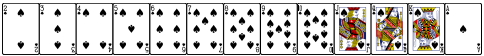
COMP 2011: Data Structures

Lecture 4. Queues and Linked Lists

Dr. CAO Yixin

yixin.cao@polyu.edu.hk

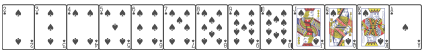
September, 2021



Review of Lecture 3

- A review of arrays.
- The memory organization in general and in Java.
- Adjustable arrays.
- Stacks and their use in parentheses matching.

What's the meaning of == applied to primitive types and objects?
link 1 link 2



Exercises on stacks

Suppose that integers 0 through 9 are pushed onto an originally empty stack in order. There are ten successful pop operations, and when a value is popped it is printed out.

Which sequence(s) cannot be the output?

1. 4 3 2 1 0 9 8 7 6 5
2. 4 6 8 7 5 3 2 9 0 1
3. 2 5 6 7 4 8 9 3 1 0
4. 4 3 2 1 0 5 6 7 8 9
5. 1 2 3 4 5 6 9 8 7 0
6. 0 4 6 5 3 8 1 7 2 9
7. 1 4 7 9 8 6 5 3 0 2
8. 2 1 4 3 6 5 8 7 9 0

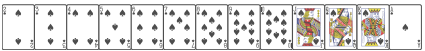
push/pop

Exercises on stacks (cont'd)

1. What is the running time of `push` and `pop`?

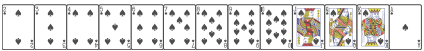
2. Suppose that P, O, L, Y, U are pushed onto an originally empty stack in order.
There are four successful `pop` operations.

In the end, there is one letter remaining in the stack.
Which letter is it? (Hint: Consider all the possibilities.)



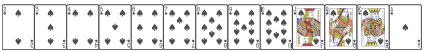
linear structures (continued)

Queues



Queues

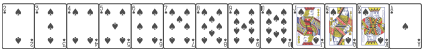
- You see it everyday at food courts.
- Cars waiting to go through the tunnel
- Files to be printed by a printer



Specification of the queue

A queue's state is modeled as a sequence of elements, initially empty.

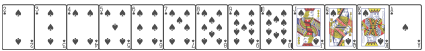
- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.



Specification of the queue

`enqueue(Peppa)`
adds "Peppa" to the rear, and returns nothing.

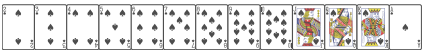
- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.



Specification of the queue

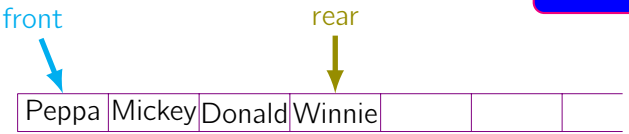
```
enqueue(Mickey)
enqueue(Donald)
enqueue(Winnie)
```

- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.

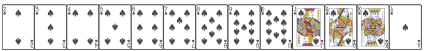


Specification of the queue

```
enqueue(Mickey)
enqueue(Donald)
enqueue(Winnie)
```



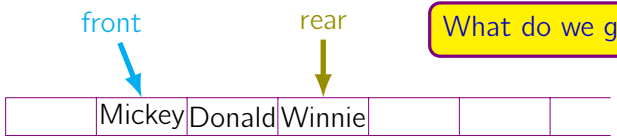
- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.



Specification of the queue

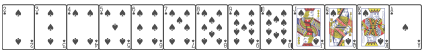
`dequeue()`
returns the first element (at the front).

What do we get by a second `dequeue`?



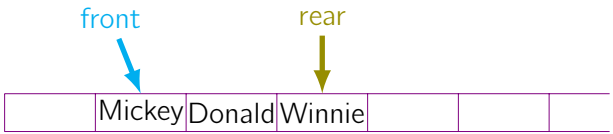
Peppa

- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.



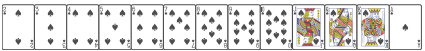
Specification of the queue

First-In First-Out (FIFO)

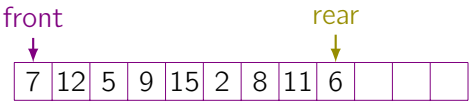


Peppa

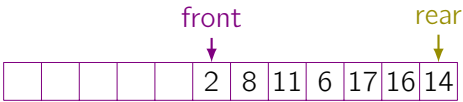
- An `enqueue(`x`)` operation inserts x at the rear (or tail) of the queue.
- An `isEmpty` operation tests if this queue is empty.
- A `dequeue` operation removes the item at the front of the queue and returns the element that was deleted.
- A (nonstandard) `peek` operation looks at the object at the front.

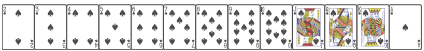


Implemented using an array

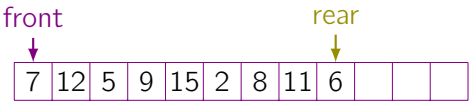


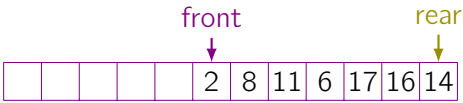
after five `dequeue` operations
and three `enqueue` operations





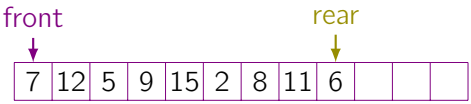
Implemented using an array

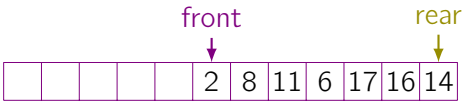




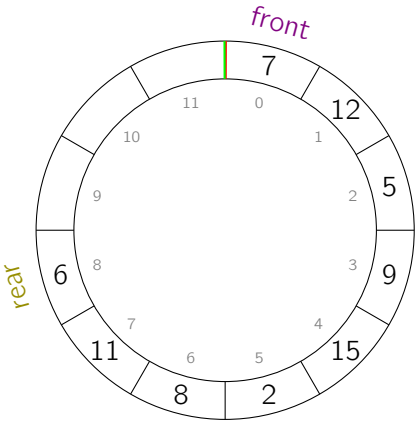
19 wants to join the queue, but...

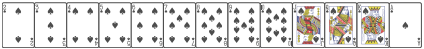
Implemented using an array circularly



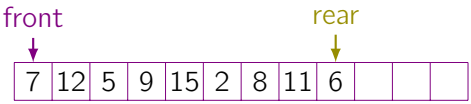


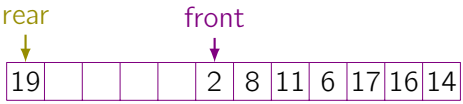
19 wants to join the queue, but...

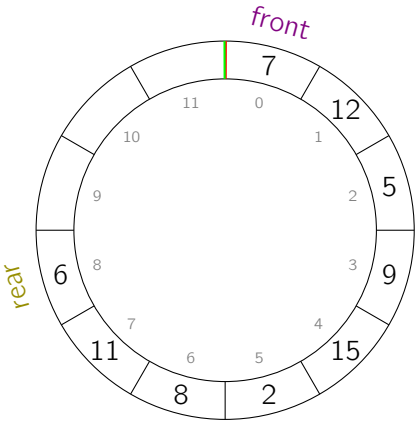


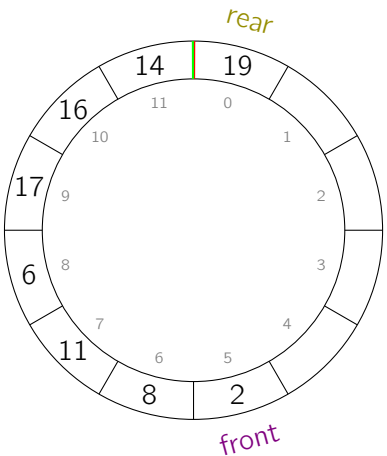


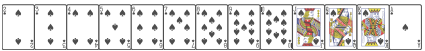
Implemented using an array circularly











Implementation details: indices

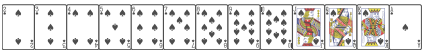
```
public class Queue<T> {
    private Object[] data;
    private int front, rear;

    public Queue(int size){
        data = new Object[size];
        front =    ;
        rear =    ;
    }

    public boolean isEmpty() {}
    public void enqueue(T e) {}
    public T dequeue() {}
}
```

initial value of `front`?

initial value of `rear`?

Observations

- enqueue changes only `rear`.
- dequeue changes only `front`.



Implementation details: indices

```
public class Queue<T> {
    private Object[] data;
    private int front, rear;

    public Queue(int size){
        data = new Object[size];
        front =    ;
        rear =    ;
    }

    public boolean isEmpty() {}
    public void enqueue(T e) {}
    public T dequeue() {}
}
```

initial value of `front`?

initial value of `rear`?

Observations

- enqueue changes only `rear`.
- dequeue changes only `front`.

`front = rear = 0`
after the first insertion.

Implementation details: indices

```
public class Queue<T> {
    private Object[] data;
    private int front, rear;

    public Queue(int size){
        data = new Object[size];
        front =    ;
        rear =    ;
    }

    public boolean isEmpty() {}
    public void enqueue(T e) {}
    public T dequeue() {}
}
```

initial value of `front`?

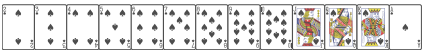
initial value of `rear`?

How about `isEmpty`?

Observations

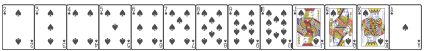
- `enqueue` changes only `rear`.
- `dequeue` changes only `front`.

`front` = `rear` = 0
after the first insertion.



```
enqueue(1) enqueue(2) enqueue(3) enqueue(4) enqueue(5) enqueue(6)
dequeue() dequeue() dequeue() dequeue() dequeue() dequeue()
```

How many different sequences we can get?

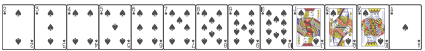


Exercises

1. What is the running time of `dequeue` and `enqueue`?

2. Suppose that P, O, L, Y, U are inserted into an originally empty queue in order.
There are four successful `dequeue` operations.

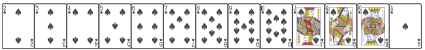
In the end, there is one letter remaining in the stack.
Which letter is it?



Overflow

- There must be an error/exception when underflow.
- For overflow, we have two choices:

1. Increase the stack/queue size when it's full.
2. Let the user to check before inserting.

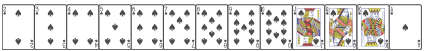


Summary of stacks and queues

- In stacks and queues, only one data item can be accessed.

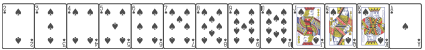
  in a stack, the last item inserted; in a queue, the first item inserted .
- The operations have special names: `push/pop` and `enqueue/dequeue`.
- Both can be implemented with (fixed-length) arrays. (there're alternatives)
- For a queue, the indices wrap around from the end of the array to the beginning.
- It's nontrivial to implement queues, but the use of stacks is more difficult.

- Arithmetic expressions are typically evaluated by translating from infix notation ($3-1$, used by human) to posfix notation ($31-$, used by computers) and then evaluating the postfix expression.
- Both need stacks.



Priority queues

Have you visited the emergency service of Queen Elizabeth Hospital?

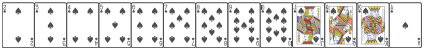


Priority queues

An urgent level is assigned to each patient after pre-evaluation.

Then the most urgent patients are taken care of first.

- A priority queue allows access to the smallest (or sometimes the largest) item.
- operations: inserting and removing the item with the smallest key.

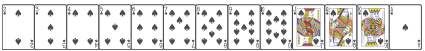


Exercise: Implementation of priority queues

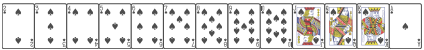
Store a priority queue as an array:

- put the new element in a proper position; (ordered)
- search the element with the smallest key when removing. (unordered)

In the ordered way, do you want to sort them increasingly or decreasingly?



Linked Lists



How does our brain store information, e.g.,

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z?

What is the 12th letter in the English alphabet?

What is the next (13th)? the previous (11th)?

What is the next (13th)? the previous (11th)?

Try to write down the lyrics of your favorite song.

Hey Jude, don't be afraid

mv

既是同舟在獅子山下且共濟

mv

We are the champions, my friends

mv

来日纵使千千阕歌飘于远方我路上

mv

_____

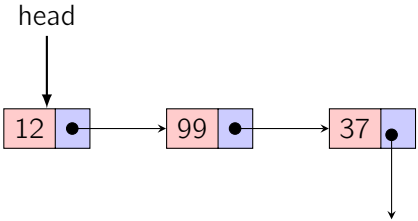
_____

You got blood on your face, you big disgrace

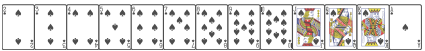
_____

_____

Linked lists

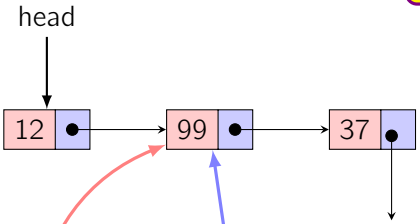


- Separate blocks of storage (nodes) are linked together using reference/pointers.
- Each node has two fields:
  a data component and a link that points to its succeeding node.
- `head` points to the first node, and its is `null` if and only if the list is empty.
- Linked lists are an important alternative to arrays.

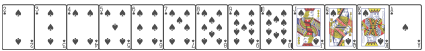


Linked lists

How to find the previous node?

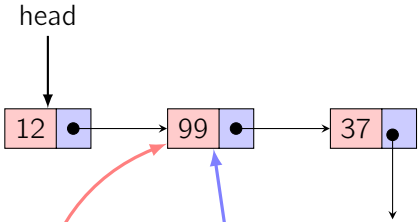


- Separate blocks of storage (nodes) are linked together using reference/pointers.
- Each node has two fields:
  a data component and a link that points to its succeeding node.
- `head` points to the first node, and its is `null` if and only if the list is empty.
- Linked lists are an important alternative to arrays.



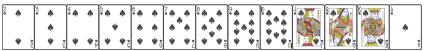
Linked lists

Operations on a linked list?

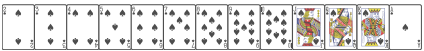


sequential representations
vs.
reference representations

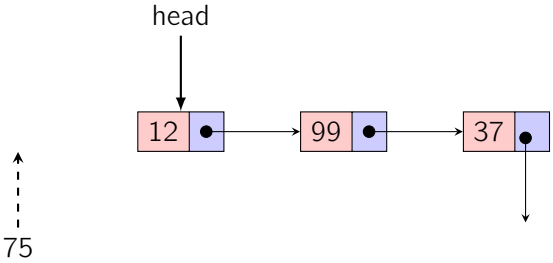
- Separate blocks of storage (nodes) are linked together using reference/pointers.
- Each node has two fields:
  a data component and a link that points to its succeeding node.
- `head` points to the first node, and its is `null` if and only if the list is empty.
- Linked lists are an important alternative to arrays.



Insertion



1. Inserting at the front

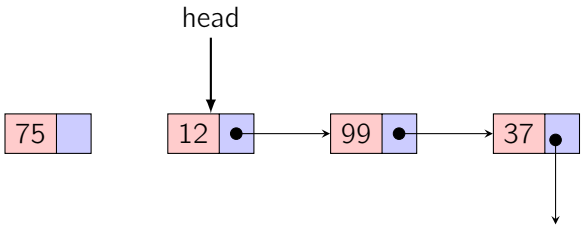


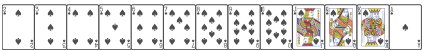
1. Inserting at the front

First, make a new node with data 75.

Then make it the new head, and let its link point to the old head.

which one first?



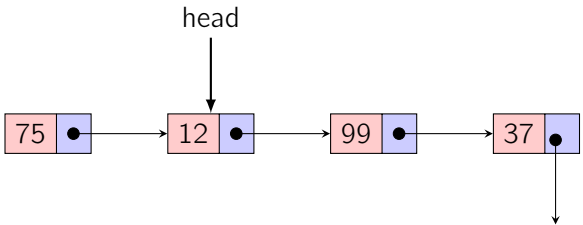


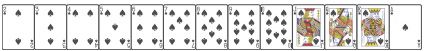
1. Inserting at the front

First, make a new node with data 75.

Then make it the new head, and let its link point to the old head.

which one first?



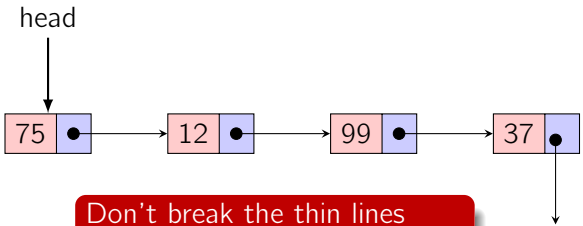


1. Inserting at the front

First, make a new node with data 75.

Then make it the new head, and let its link point to the old head.

which one first?

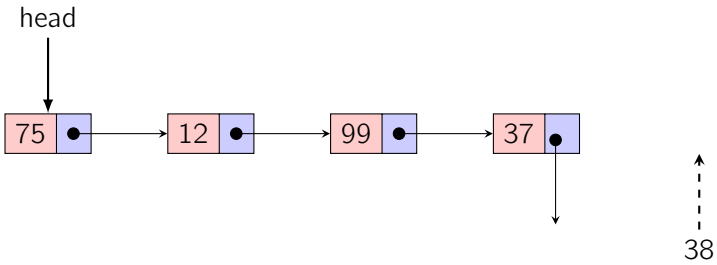


Don't break the thin lines

Once you lose access to your data, you never get it back!

2. Inserting at the end

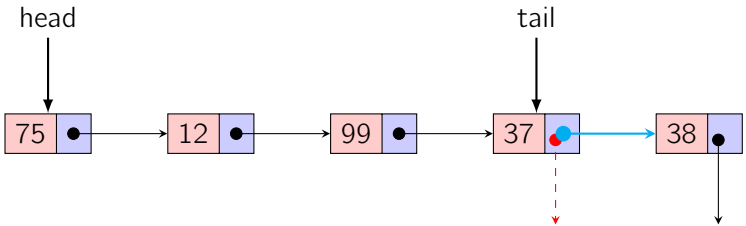
The first step is _____?



2. Inserting at the end

The first step is _____?

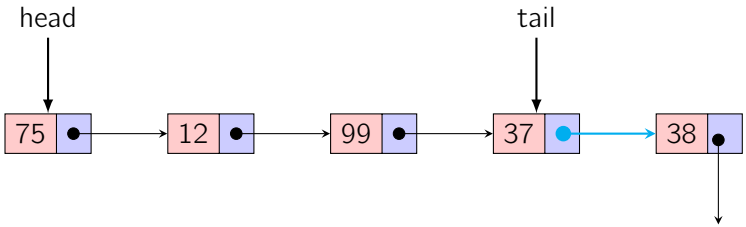
Then find the "tail" (last node), make it point to the new node.



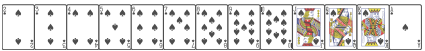
2. Inserting at the end

The first step is _____?

Then find the "tail" (last node), make it point to the new node.



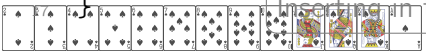
What is the running time?
What if the list was empty?



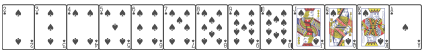
```java
public class LinkedList<T> {
    static class Node<T> {T element; Node<T> next;}
    Node<T> head;
    public void insertFirst(T a) {
        Node<T> newNode = new Node<T>(a);
        newNode.next = head;
        head = newNode;
    }
    public void insertLast(T a) {
        if (head == null) {insertFirst(a); return;}
        Node<T> cur = head;
        while (cur.next != null) cur = cur.next;
        Node<T> newNode = new Node<T>(a);
        newNode.next = null;
        cur.next = newNode;
    }
}
```

A boundary case (easy to handle).

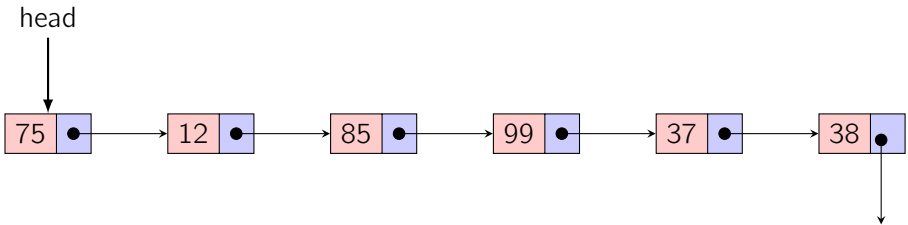
Inserting in the middle (after certain node) is similar as at the end.



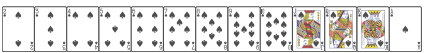
Deletion



Removing the first node (and returning it element)



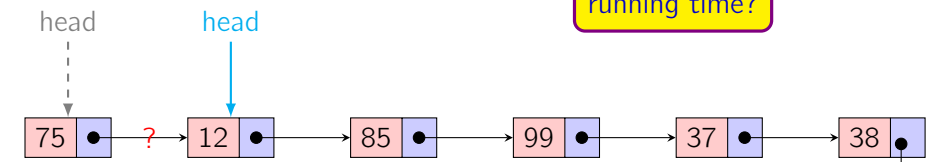
What are the boundary cases?



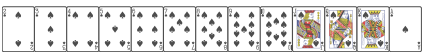
Removing the first node (and returning it element)

Let the head point to the next node, and return the first element (75).

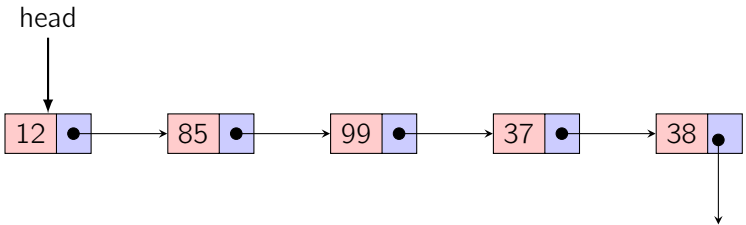
running time?



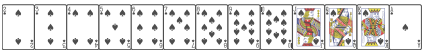
Do we need to remove this link?



Removing the last node (and returning it element)

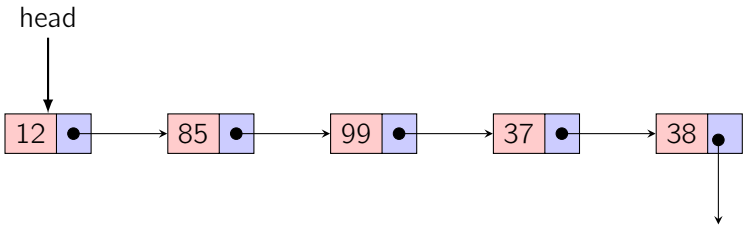


What are the boundary cases?



Removing the last node (and returning it element)

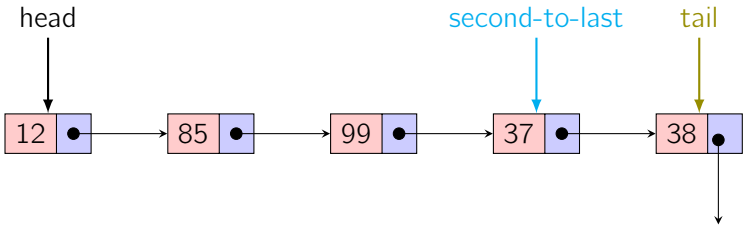
1. find the last node of the list;
2. let the link of its previous node point to `null`.

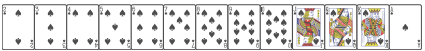


Removing the last node (and returning it element)

1. find the last node of the list;
2. let the link of its previous node point to `null`.

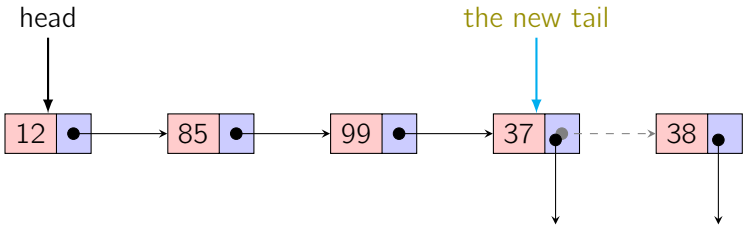
So, we need to keep track of two nodes during traversal.





Removing the last node (and returning it element)

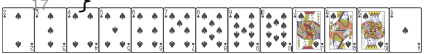
1. find the last node of the list;
2. let the link of its previous node point to `null`.



What are the boundary cases?

1. the list is empty; or
2. it has only one element.

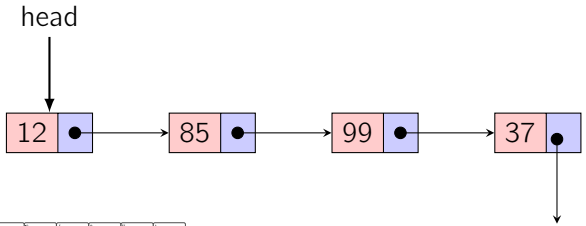
```java
public T removeFirst() {
    if (head==null) {System.out.println("underflow"); return n
    Node<T> temp = head;
    head = head.next;
    temp.next = null;       // optional but suggested.
    return temp.element;
}

public T removeLast() {
    if (head == null || head.next == null) return removeFirst(
    Node<T> secondToLast = head;
    Node<T> last = secondToLast.next;
    while (last.next != null) {
        secondToLast = secondToLast.next; last = last.next;}
    secondToLast.next = null;   // very important
    return last.element;
}
```

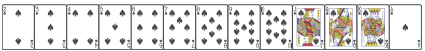


Complexity and improvements

insert		delete	
at the front	at the rear	from the front	from the rear
$O(1)$	$O(n)$	$O(1)$	$O(n)$

Can you improve them?

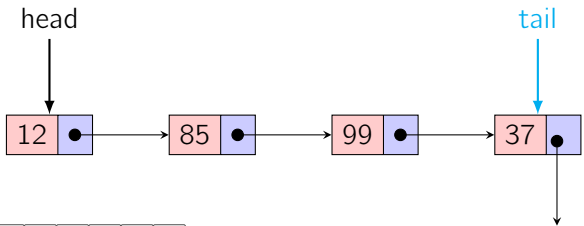


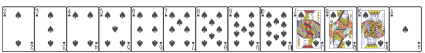


Complexity and improvements

insert		delete	
at the front	at the rear	from the front	from the rear
$O(1)$	$O(n)O(1)$	$O(1)$	$O(n)$

Can you improve them?



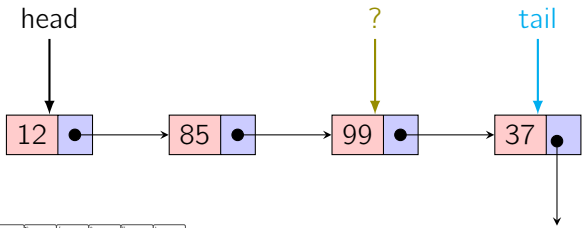


Complexity and improvements

insert		delete	
at the front	at the rear	from the front	from the rear
$O(1)$	$O(n)O(1)$	$O(1)$	$O(n)$

Anything else that may help?

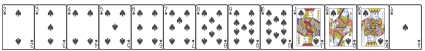
How about a second reference?



Augmenting a data structure

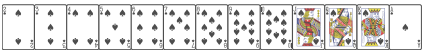
- More often than not, textbook data structures are not sufficient.
- We've seen previously nonstandard operations (`peek`).
- We can also add data fields: One single reference (`tail`) can save big!
- Another field that might be helpful is the `size`.

- However, with the convenience of them comes the trouble of maintaining them.
  - It's almost trivial to maintain `size`: one line in insert and one line in delete.
  - But it's tricky to maintain `tail`.
    try to do it as an exercise.



Pitfalls of linked lists

- Be careful with boundary cases (depending on your action):
  - the empty list,
  - the list with one element (in some cases),
  - manipulating the first/last node, and
  - .....
- Don't lose access to needed objects: There is usually only one correct order to change the references (pointers).
- Make sure a link is *not* `null` before following it.
- Mark end of list by setting the `next` link of the last node point to `null`.
- Diagrams help. Draw it when you're not certain.

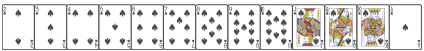


Pretty much everything is a (nontrivial) combination of arrays and lists.

In data structures...it is more important to really understand the basic material than have exposure to more advanced concepts.

Steven S. Skiena
"The Algorithm Design Manual," Page 65

So I'll spend more time on "easy" things than others usually do.

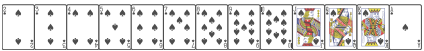


Pretty much everything is a (nontrivial) combination of arrays and lists.

In data structures...it is more important to really understand the basic material than have exposure to more advanced concepts.

Steven S. Skiena
"The Algorithm Design Manual," Page 65

So I'll spend more time on "easy" things than others usually do.

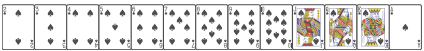


Pretty much everything is a (nontrivial) combination of arrays and lists.

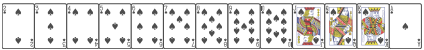
In data structures...it is more important to really understand the basic material than have exposure to more advanced concepts.

Steven S. Skiena
"The Algorithm Design Manual," Page 65

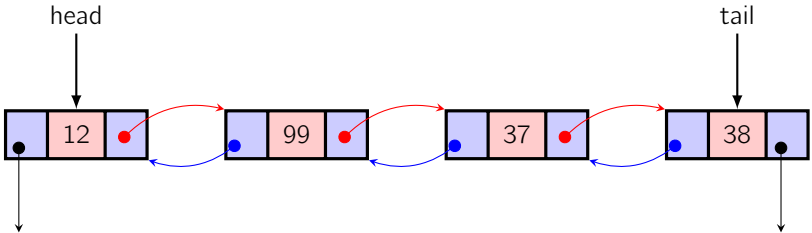
So I'll spend more time on "easy" things than others usually do.

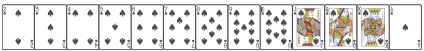


Doubly Linked Lists



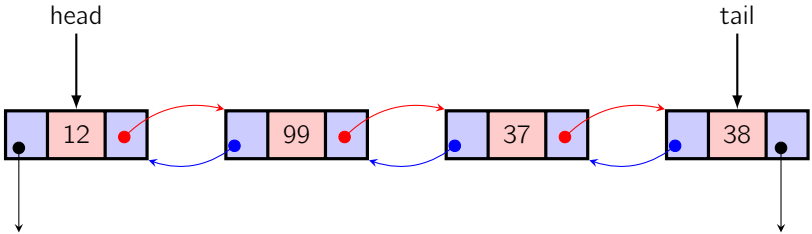
Doubly linked lists





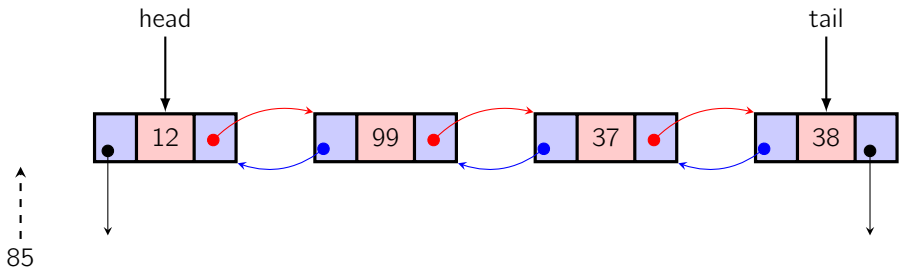
Doubly linked lists

Complexity of reversing a doubly linked list?

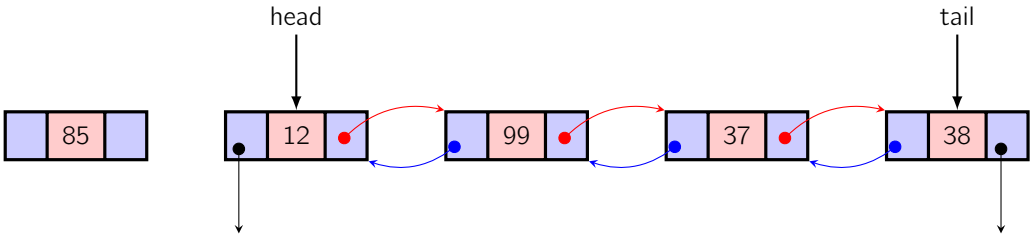


Inserting at the front

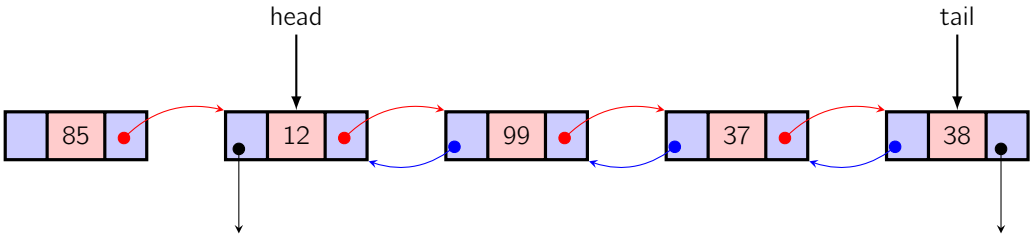
Insert 85 at the front?



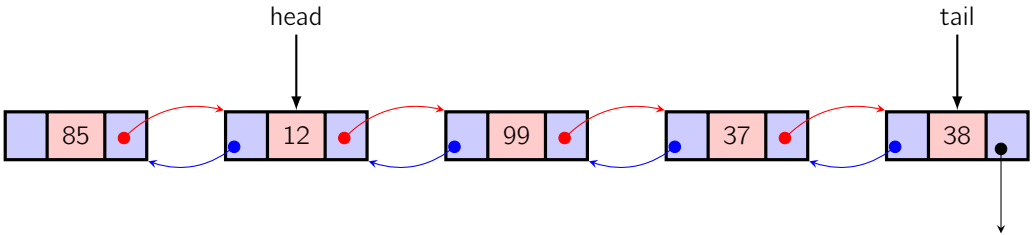
Inserting at the front



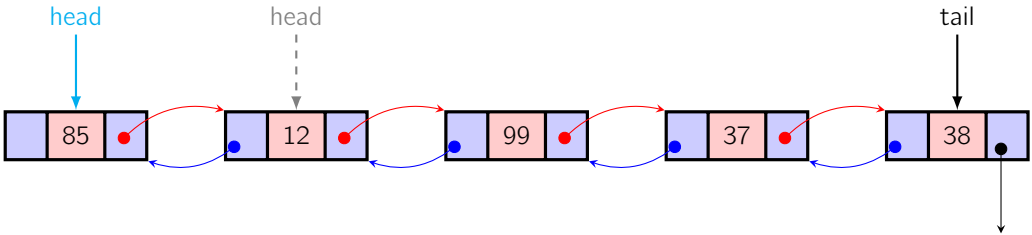
Inserting at the front



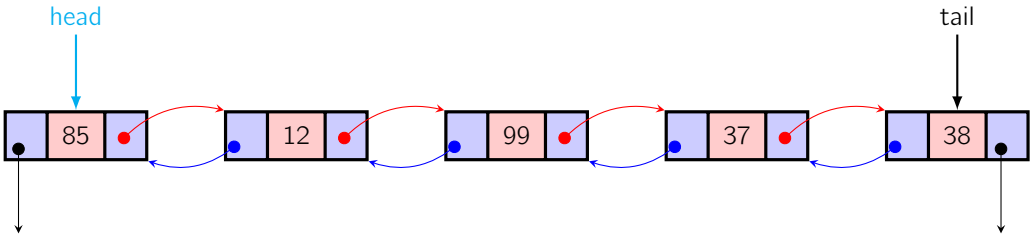
Inserting at the front



Inserting at the front

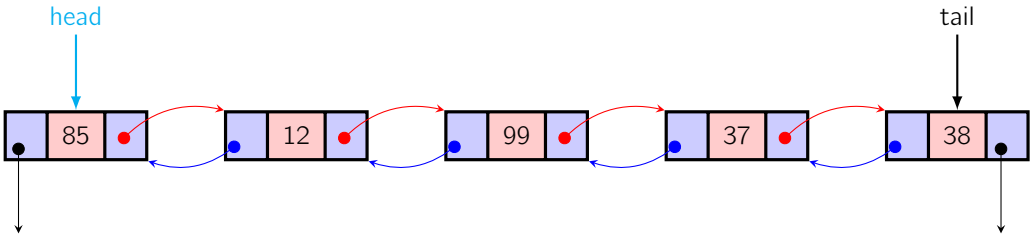


Inserting at the front



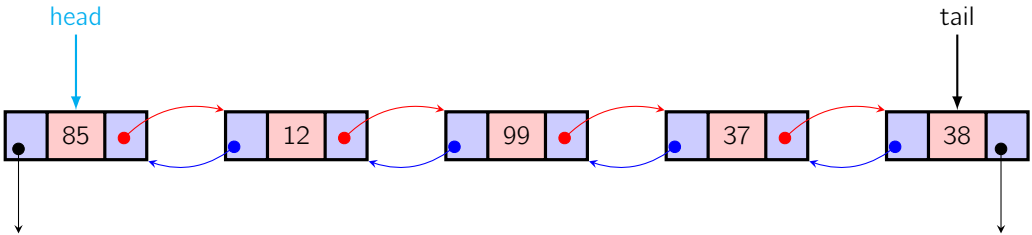
Inserting at the front

Which order is the best, and why?



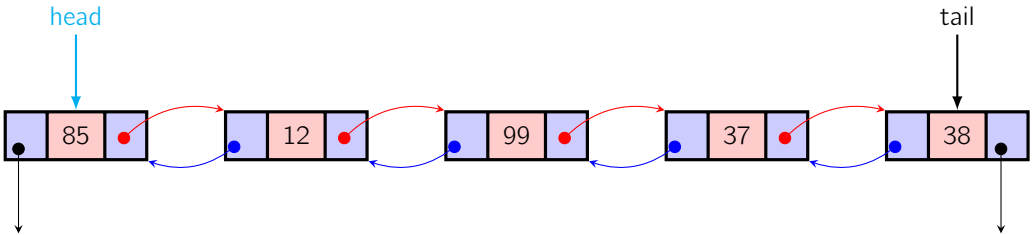
Inserting at the front

How about insert at the end?

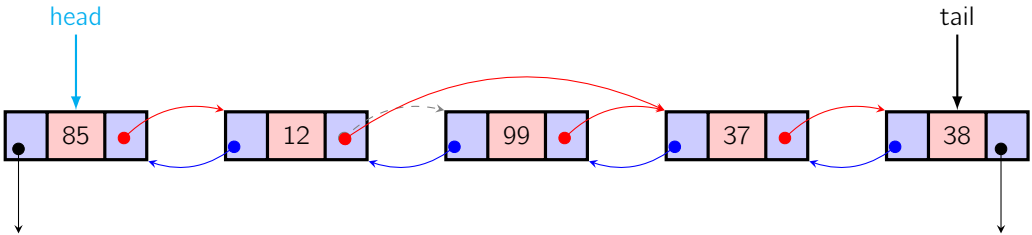


Inserting at the front

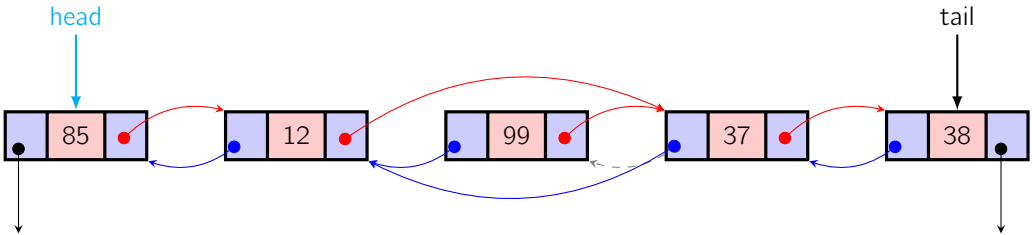
Deletion?



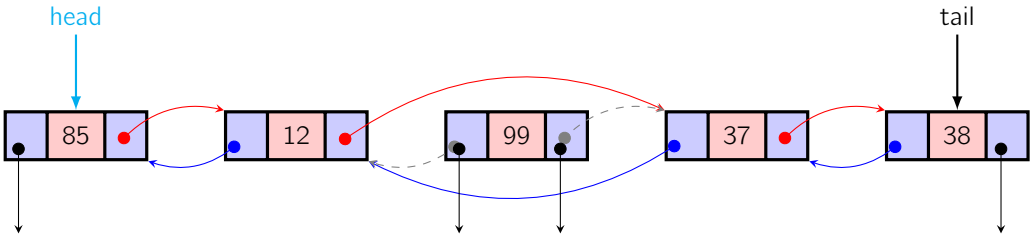
Inserting at the front



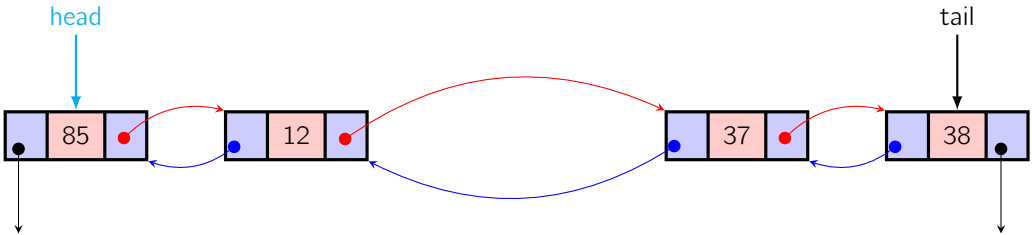
Inserting at the front



Inserting at the front

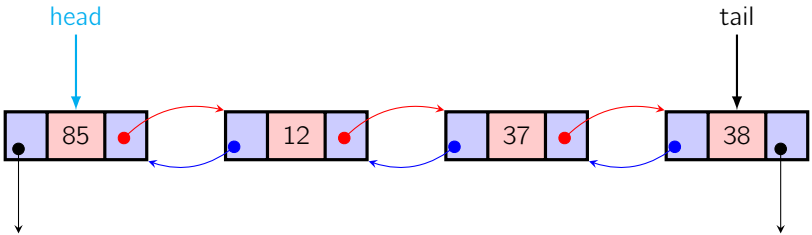


Inserting at the front



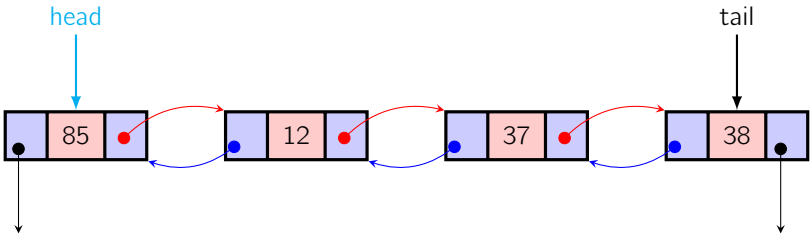
Inserting at the front

Boundary conditions?



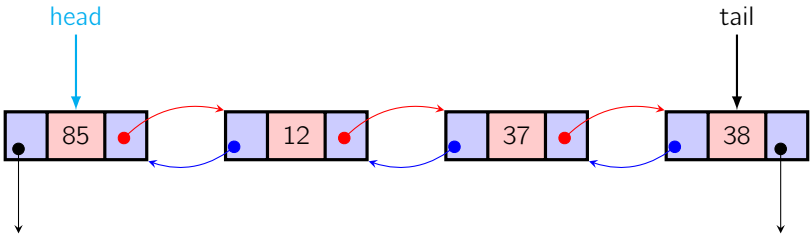
Inserting at the front

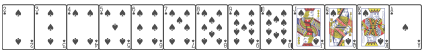
Doubly linked lists must be a gift of God:
Everything is so nice.



Inserting at the front

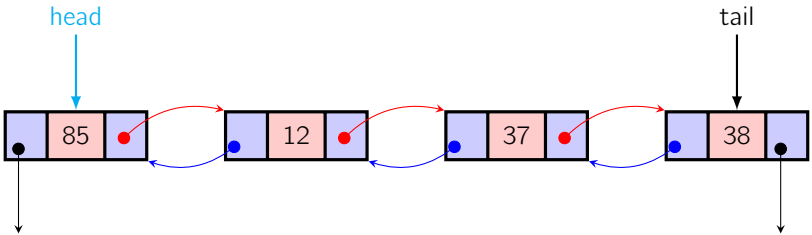
But, why don't we use doubly linked lists for queues/stacks?





Inserting at the front

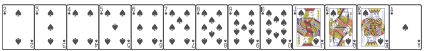
You stop thinking so when you need to swap two nodes...



Sorting linked lists

- We frequently want an array to be sorted; e.g., we can do binary search.
- Does it help search if we have a linked list sorted?
- Binary search cannot be used on linked lists! because it relies on the consecutive storage.

- For some applications, we do want to sort a linked list.
- Which sorting algorithms we have learned can be adapted to (doubly) linked lists?



Summary

- A linked list consists of one list object and a number of node objects.
- The list object contains a reference, often called `head`, to the first node.
- Each node object contains data and a reference `next` to the next node in the list.
- A `next` value of `null` indicates the end of the list.
- To traverse a linked list, you start at `head` and then go from node to node, using each nodes' `next` field to find the next node.
- A double-ended list also maintains a reference `tail` to the last node in the list.
- A double-ended list allows insertion at the end of the list efficiently.
- Stacks and queues can be implemented using either arrays or linked lists.
- A doubly linked list permits backward traversal, but it's even more time-consuming and difficult to maintain.

- As a good software engineering practice, use an iterator to traverse a list. (examples)


Happy Holidays!