David Croft

Allays

Linked lists

LL example

structures

types

Queue:

Stacks

Juck.

...

....

## 122com Data structures and types

**David Croft** 

Coventry University david.croft@coventry.ac.uk

2016



## Overview

- 1 Arrays
- 2 Linked lists
  - Array example
  - LL example
- 3 Data structures
- 4 Abstract data types
- 5 Queues
- 6 Stacks
- 7 Sets
- 8 Other
- 9 Trees
- 10 End

David Croft

#### Arrays

Linked list

Array example

LL example

Abstract dat

types

Queues

Stacks

J ca crit

Otho

Tree

End

A series of objects all of the same size and type.

```
char array[] = {'A', 'B', 'C', 'D', 'E'};
```

- Stored in contiguous blocks of memory.
- Python lists are functionally closest.
  - But are not arrays.
- Can't be resized.



David Croft

#### Linked lists

The challenger for array's crown.

- Series of nodes, each of which points to the next element.
  - And to the previous element if it's a doubly linked list.

$$D \rightarrow$$

Doubly linked 
$$\leftarrow A \leftarrow B \leftarrow C \rightarrow D$$

$$\leftarrow A$$

$$\rightarrow$$
  $\leftarrow$ 

$$\stackrel{\leftarrow}{\rightarrow}$$

$$\stackrel{\leftarrow}{\rightarrow}$$



#### Arrays Linked lists

Array example

Data structure

Abstract data

Queues

~ -----

Stacks

Othe

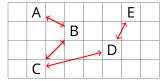
Tree

End

## Coventry University

### Not in contiguous memory.

- Each node is separate.
- Scattered.
- Dynamic memory (pointers!).



- Why would we use linked lists instead of arrays?
  - Can change size.
  - Can quickly insert and delete elements.

```
class Node:
   __prev = None
   __next = None
   value = None
```

Linked lists II

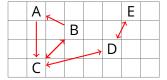
```
class Node
{
private:
    Node *prev;
    Node *next;

public:
    int value;
};
```

## Coventry University

### Not in contiguous memory.

- Each node is separate.
- Scattered.
- Dynamic memory (pointers!).



- Why would we use linked lists instead of arrays?
  - Can change size.
  - Can quickly insert and delete elements.

```
class Node:
    __prev = None
    __next = None
    value = None
```

Linked lists II

```
class Node
{
private:
    Node *prev;
    Node *next;

public:
    int value;
};
```



#### Arrays Linked lists

Array example

Data structure

Abstract data

Queues

. .

Sets

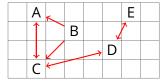
Othe

Tree

## Coventry University

### Not in contiguous memory.

- Each node is separate.
- Scattered.
- Dynamic memory (pointers!).



- Why would we use linked lists instead of arrays?
  - Can change size.
  - Can quickly insert and delete elements.

```
class Node:
   __prev = None
   __next = None
   value = None
```

Linked lists II

```
class Node
{
private:
    Node *prev;
    Node *next;

public:
    int value;
};
```

Data structures

types

Queues

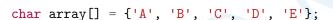
Stack

\_\_\_\_

Other

Tree

Fnd





- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?



OHAHAS

Stack

....

char array[] = {'A', 'B', 'C', 'D', 'E'};



- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?
  - 1 Remove element from the array.



structures

Abstract dat types

Queues

c. .

Sets

Otha

Tree

End



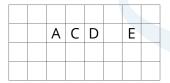


- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?
  - 1 Remove element from the array.
  - 2 Move next element to occupy the empty space.



# Removing array elements

char array[] = {'A', 'B', 'C', 'D', 'E'};



- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?
  - 1 Remove element from the array.
  - Move next element to occupy the empty space.
  - Repeat.

Abstract dat types

Queue

Stack

Sets

Othe

Tree

End

## Coventry University

# Removing array elements

C



- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?
  - Remove element from the array.
  - 2 Move next element to occupy the empty space.
  - Repeat.





- Array in memory, multiple elements in a contiguous block.
- How do we remove elements from the middle?
  - 1 Remove element from the array.
  - Move next element to occupy the empty space.
  - Repeat.
- Is very slow with large arrays.

Ougues

Ctoole

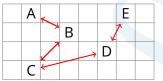
Sets

Othe

Tree

End





- Linked list, separate elements scattered in memory.
- Each pointing to the next/prev element.
- How do we remove elements?

71

Queue

Stack

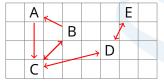
Sets

Othe

Tree

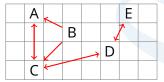
End





- Linked list, separate elements scattered in memory.
- Each pointing to the next/prev element.
- How do we remove elements?
  - Change pointers.





- Linked list, separate elements scattered in memory.
- Each pointing to the next/prev element.
- How do we remove elements?
  - Change pointers.

### Linked li

Array example

Abstract dat

types

Queue

Stack

. .

Othe

Troo

End



- Linked list, separate elements scattered in memory.
- Each pointing to the next/prev element.
- How do we remove elements?
  - Change pointers.
  - Delete old element.



#### David Croft

Linked lis

structures
Abstract da

types

Queue

Stack

Sets

Othe

Tree

End

# **Coventry** University

### Advantages

- Inserting and deleting elements is very fast.
  - **O**(1).
- No size limits, can keep adding new elements.
- Doesn't waste memory.

### Disadvantages

- Not indexed.
  - Can't ask for the 20th element etc.
  - Have to step through the list (slow).
- Needs more memory than an array to store the same number of elements.
  - Have to keep track of where the next/prev nodes are.

#### Array:

Array example

## Data structures

types

Oueues

~----

Stacks

...

Troo

End

Arrays and linked lists are data structures.

- A specific way of storing data.
- Can see how the various elements of the structure are laid out in memory.
- Direct access to the underlying memory.



Queues

Stack

Sets

Othe

Troo

As we move to storing more complex information in our software we well start to encounter Abstract Data Types (ADTs).

Software engineering principal.



Abstract data types

structures

Abstract data

Abstract data types

Queue:

Stack

sets

Otne

Tree

- Software engineering principal.
- Keep what a data type can do...



#### Array

Array example

structures

Abstract data

types

Queues

Stacks

J ( a C .

Tree

- Software engineering principal.
- Keep what a data type can do... ...and how it does it separate.



Arrays

Array example LL example

Abstract data

types

Queues

Stack

sets

Othic

Trees

- Software engineering principal.
- Keep what a data type can do... ...and how it does it separate.
- Unlike data structure ADTs only concerned with the interface.



### Arrays

Array example LL example

structures
Abstract data

types

Queues

Stacks

.

Othe

Troo

Tree

- Software engineering principal.
- Keep what a data type can do... ...and how it does it separate.
- Unlike data structure ADTs only concerned with the interface.
- Internals of ADTs can vary widely between implementations.





structs & types

David Croft

122com Data

Arrays

Linked lists

Data

Abstract data

Queue

Stack

Sets

Othe

Tree

Imagine an ADT like a car.

It has a set of supported operations, go faster, go slower, turn left, turn right.





structs & types

David Croft

122com Data

Arrays

Linked lists
Array example

Data structure

Abstract data types

Queues

Stack

Cata

Othe

Tree

Imagine an ADT like a car.

- It has a set of supported operations, go faster, go slower, turn left, turn right.
- Don't care how it achieves these.



#### Arrays

Array example

Data structure:

Abstract data types

Queues

Stack

Sets

Othe

Tree

### Imagine an ADT like a car.

- It has a set of supported operations, go faster, go slower, turn left, turn right.
- Don't care how it achieves these.
- Don't care if, internally, it's using a combustion engine or an electric motor.



122com Data

Davia Crc

### Arrays

Array example

Data structure

Abstract data

types

Queues

Stacks

Sets

Othe

Tree

Imagine an ADT like a car.

- It has a set of supported operations, go faster, go slower, turn left, turn right.
- Don't care how it achieves these.
- Don't care if, internally, it's using a combustion engine or an electric motor.
- Only care about the result.



### Arrays

Linked lists
Array example
LL example

Abstract dat

types

Queues

Stacks

- oto

Othe

Tree

### A First In First Out (FIFO) ADT.

- Ends of the queue called the front and back.
- New elements added to back of queue only.
  - Pushing push(value)
- Old elements removed from front of queue only.
  - Popping pop()
- No cutting in.



David Croft

#### Arrays

Linked lists
Array example
LL example

Abstract data

O. . . . . .

Queues

Stacks

Sets

Othe

Tree

Data

## A FIFO ADT.

- Ends of the queue called the front and back.
- New elements added to back of queue only.
  - Pushing push(value)
- Old elements removed from front of queue only.
  - Popping pop()
- No cutting in.
- Buffer to hold items for processing in the order in which they arrive.



**Queues** 

#### A FIFO ADT.

- Ends of the queue called the front and back.
- New elements added to back of queue only.
  - Pushing push(value)
- Old elements removed from front of queue only.
  - Popping pop()
- No cutting in.
- Buffer to hold items for processing in the order in which they arrive.
- Which would be better for a queue? An array or a linked list?



**David Croft** 

### Arravs

Linked lists
Array example
LL example

Abstract dat

Ougues

Queues

Stacks

sets

Othe

ree

# End

# Coventry University

### A FIFO ADT.

- Ends of the queue called the front and back.
- New elements added to back of queue only.
  - Pushing push(value)
- Old elements removed from front of queue only.
  - Popping pop()
- No cutting in.
- Buffer to hold items for processing in the order in which they arrive.
- Which would be better for a queue? An array or a linked list?
  - Linked list.

Data structures

types

Queues

Stack

Tree

# Coventry University

# Array as a queue.

1

 $front \Rightarrow$ 

- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.

Abstract data types

Queues

Stack

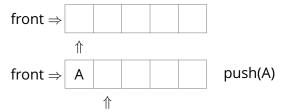
\_ .

Othe

Tree

Coventry University

# Array as a queue.



- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.

Abstract data

Queues

Stacks

Coto

Othe

Tree

rpes d Croft Array as a queue.

I

front  $\Rightarrow$   $\uparrow$  front  $\Rightarrow$  A

push(A)

 $\uparrow$ 

front  $\Rightarrow$  A B

push(B)

 $\uparrow$ 

- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.



Queues

 $front \Rightarrow$ push(A) front  $\Rightarrow$  A  $\uparrow$ 

push(B)

В

front  $\Rightarrow$  A

push(C) front  $\Rightarrow$ В

 $\uparrow$ 

Array as a queue.

- Very similar to stacks.
  - Keep track of next free space.
    - Limited size.



Abstract data

Queues

Stacks

C-4-

Othe

Tree

## Coventry University

front  $\Rightarrow$ 

## Array as a queue.

I

front  $\Rightarrow$ front  $\Rightarrow$ front  $\Rightarrow$ A

push(A)

front  $\Rightarrow$ A

B

push(B)

1

В

- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.
- What happens when we pop()?
  - Have to shuffle every element forward one space.
  - Inefficient.

push(C)

Queues

# Covento

## Array as a queue.

 $front \Rightarrow$ 

front  $\Rightarrow$ 

- front  $\Rightarrow \mid A$ 1
- push(A)
- В
- push(B)

- push(C) front  $\Rightarrow$ В
- $front \Rightarrow$ Α В 1
  - pop()

Inefficient.

- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.
- What happens when we pop()?
  - Have to shuffle every element forward one space.

Data structures

Abstract data types

Queues

Stack

Sets

Othe

Tree

End



## Array as a queue.

Very similar to stacks.

space.

- front ⇒ ↑
- front  $\Rightarrow A$ 
  - $\uparrow$
- front  $\Rightarrow$  A B push(B)
- front  $\Rightarrow$  A B C push(C)
- front  $\Rightarrow$  A B

Limited size.

push(A)

pop()

- What happens when we pop()?
  - Have to shuffle every element forward one space.

Keep track of next free

Inefficient.

Queues



## Array as a queue.

- front  $\Rightarrow$
- front  $\Rightarrow \mid A$

push(A)

1

front  $\Rightarrow$ В

push(B)

pop()

push(C) front  $\Rightarrow$ В

front  $\Rightarrow$ В 1

- Very similar to stacks.
  - Keep track of next free space.
  - Limited size.
- What happens when we pop()?
  - Have to shuffle every element forward one space.
  - Inefficient.

Data structures

Abstract dat types

Queue:

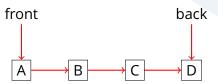
Charles

**T...** 

End

## Linked list as a queue.







#### Array

Linked lists

LL example

structures

pop()

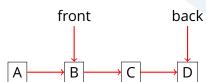
Abstract data types

Queues

. .

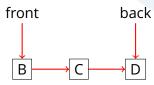
\_\_\_\_

End



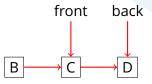


pop()





pop(), pop()





Abstract data types

Queues

Stacks

041- -

Troo

End

## Linked list as a queue.

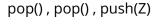
pop(), pop()

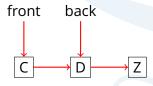














Queues

Stacks

Otho

Troo

End

## Linked list as a queue.

pop(), pop(), push(Z)





David Croft

**Stacks** 

A First In Last Out (FILO) ADT.

- Ends of the stack are called the top and bottom.
- New elements add to top of stack only.
  - Pushing push(value)
- Old elements removed from top of stack only.
  - Popping pop()
- No cutting in.



**Stacks** 

### Arrays

Linked list Array example LL example

Abstract data

types

Queues

Stacks

Jeack

5005

1100

### A FILO ADT.

- Ends of the stack are called the top and bottom.
- New elements add to top of stack only.
  - Pushing push(value)
- Old elements removed from top of stack only.
  - Popping pop()
- No cutting in.
- Which would be better for a stack? An array or a linked list?



**Stacks** 

types

Queue

Stacks

Sets

Othe

Tree

End

#### A FILO ADT.

- Ends of the stack are called the top and bottom.
- New elements add to top of stack only.
  - Pushing push(value)
- Old elements removed from top of stack only.
  - Popping pop()
- No cutting in.
- Which would be better for a stack? An array or a linked list?
  - Doesn't matter performance wise.
  - Linked list if n is unknown.



types

Queue

Stacks

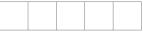
sets

Otne

End

Array as a stack.







- Keep track of position of the next free space in the array.
- Arrays have a fixed size.
  - Can't hold more values than we have space for.



### Arrays

Array example LL example

structures

types

Queues

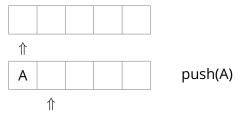
Stacks

5005

End

## Array as a stack.





- Keep track of position of the next free space in the array.
- Arrays have a fixed size.
  - Can't hold more values than we have space for.



Data structures

Abstract data types

Queues

Stacks

Othe

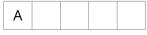
Tree

End

Array as a stack.







push(A)





push(B)



Keep track of position of the next free space in the array.

- Arrays have a fixed size.
  - Can't hold more values than we have space for.

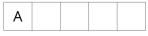


**Stacks** 









push(A)





push(B)





push(C)



- Keep track of position of the next free space in the array.
- Arrays have a fixed size.
  - Can't hold more values than we have space for.



#### Arrays Linked lis

Linked lists
Array example
LL example

structures

Abstract data types

Queue:

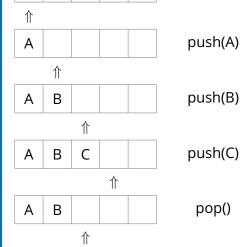
Stacks

\_ .

Othe

Tree

End







- Keep track of position of the next free space in the array.
- Arrays have a fixed size.
  - Can't hold more values than we have space for.



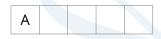
- An unordered ADT.
  - Items ordered by the set.
  - You have no control over it.
- Sets contain unique elements.
  - Can't contain duplicates.
- Can add items to a set.
- Can remove items from a set.
- Can see if an item is in a set.
- Can't get the  $n^{th}$  element.
  - It's unordered remember.



Sets C

- An unordered ADT.
  - Items ordered by the set.
  - You have no control over it.
- Sets contain unique elements.
  - Can't contain duplicates.
- Can add items to a set.
- Can remove items from a set.
- Can see if an item is in a set.
- Can't get the  $n^{th}$  element.
  - It's unordered remember.





add(A)

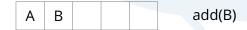
Sets

Coventř

- An unordered ADT.
  - Items ordered by the set.
  - You have no control over it.
- Sets contain unique elements.
  - Can't contain duplicates.
- Can add items to a set.
- Can remove items from a set.
- Can see if an item is in a set.
- Can't get the *n*<sup>th</sup> element.
  - It's unordered remember.







Sets

Sets

An unordered ADT.

- Items ordered by the set.
- You have no control over it.
- Sets contain unique elements.
  - Can't contain duplicates.
- Can add items to a set.
- Can remove items from a set.
- Can see if an item is in a set.
- Can't get the *n*<sup>th</sup> element.
  - It's unordered remember.





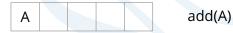


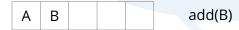


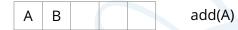


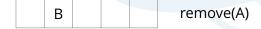
- An unordered ADT.
  - Items ordered by the set.
  - You have no control over it.
- Sets contain unique elements.
  - Can't contain duplicates.
- Can add items to a set.
- Can remove items from a set.
- Can see if an item is in a set.
- Can't get the  $n^{th}$  element.
  - It's unordered remember.











### Arrays

Linked list

Array example

LL example

structures

Abstract data types

Queue

Stack

Sets

Other

Tree

End

- Lots of other ADTs.
- Different names in different languages.
- Lists.
- Circular lists.
- Associative arrays.
  - Dictionaries/Maps.
- Double-ended queues.
- Trees.
- Graphs.



**Trees** 

#### Arrays

Array example

## Data structure

Abstract data types

Queues

Stack

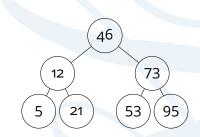
C-4-

Othe

Trees

### Variation on linked lists.

- Made of nodes and relationships.
- Root node at top.
- Each node can have > o children.
- Binary search tree.
  - Very common type.
  - Ordered.
  - Max two children.
  - Binary searching.
  - Very good for sets.



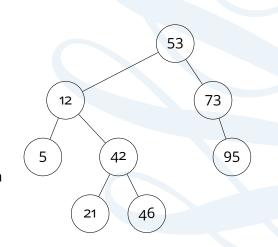


**Balance** 

Trees

Trees can be balanced or unbalanced.

- Not required for all trees.
- Going to be talking about BSTs from here on.
- Unbalanced because more than a one node difference between the two halves.

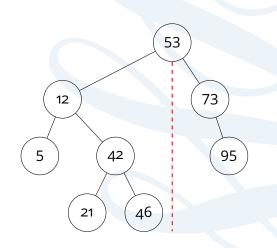




**Balance** 

Trees

- Trees can be balanced or unbalanced.
- Not required for all trees.
- Going to be talking about BSTs from here on.
- Unbalanced because more than a one node difference between the two halves.
  - For the whole tree...

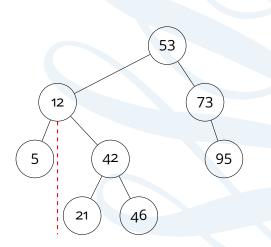




## Coventry University



- Trees can be balanced or unbalanced.
- Not required for all trees.
- Going to be talking about BSTs from here on.
- Unbalanced because more than a one node difference between the two halves.
  - For the whole tree...
  - ...and one of the subtrees.



**Balance** 

Data structures

Abstract data types

Queue:

Stacks

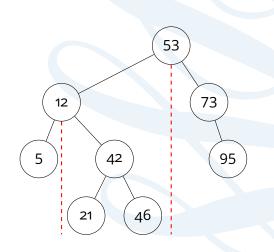
Sets

Othe

Trees

End

- Trees can be balanced or unbalanced.
- Not required for all trees.
- Going to be talking about BSTs from here on.
- Unbalanced because more than a one node difference between the two halves.
  - For the whole tree...
  - ...and one of the subtrees.







### David Croft

Arrays

Array example

Data structures

Abstract dat types

Queues

Stacks

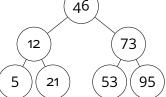
Othe

Trees

Coventry University







## Degenerate tree.



122com Data structs & types

David Croft

Allays

Linked lists

LL example

Data structures

types

Queue:

Stack

Othe

Tree

End





#### Arrave

Linked list
Array example
LL example

structures
Abstract dat

Abstract dat types

Queue

Stacks

Othe

Tree End Arrays.

Advantages/disadvantages.

Linked lists .

Advantages/disadvantages.

How to insert/delete.

 Difference between data structure and ADTs.

Stack.

FILO.

Using an array as one.

Using a LL as one.

**Q**ueue.

FIFO.

Using an array as one.

Using a LL as one.

Sets.

No duplicates.

Unordered.

Trees.

Balanced/unbalanced.



122com Data structs & types

David Croft

Arrays

Linked lists

D - t -

structures

Abstract dat types

Queues

Stacks

Jucks

...

Troo

End

# The End

