David Croft

Arravs

Linked lists Array example

Data

Abstract data

· ·

Stacks

Othe

_

Recar

Data structures and Abstract Data Types

David Croft

Coventry University

david.croft@coventry.ac.uk

2017





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

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

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



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.





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

$$\vdash | \mathsf{A} |$$

$$| \rightarrow |$$
 \leftarrow

$$\leftarrow$$

$$\stackrel{
ightarrow}{\leftarrow}$$



Data structures

Abstract dat types

Queues

Stacks

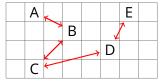
Othe

Tree

Recap

Not in contiguous memory.

- Each node is separate.
- Scattered.
- C++ 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
```

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

public:
    int value;
};
```



Linked lists
Array example
LL example

Data structures

Abstract data types

Queues

~ .

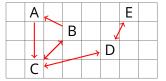
Othe

Tree

Reca

Not in contiguous memory.

- Each node is separate.
- Scattered.
- C++ 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
```

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

public:
    int value;
};
```



Linked lists
Array example
LL example

Data structures

Abstract data

Queues

Stacks

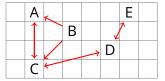
Othe

Tree

Reca

Not in contiguous memory.

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



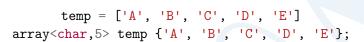
- Why would we use linked lists instead of arrays?
 - Can change size.
 - Can quickly insert and delete elements.

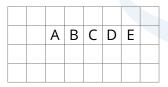
```
class Node:
    __prev = None
    __next = None
    value = None
```

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

public:
    int value;
};
```







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



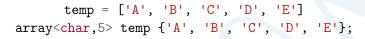
Data structure

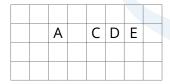
Abstract data

Queue

...

Recap





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



Data structure

Abstract data

Queue

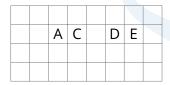
Oth.

1100

Recap



```
temp = ['A', 'B', 'C', 'D', 'E']
array<char,5> temp {'A', 'B', 'C', 'D', 'E'};
```



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



Abstract dat

OHAHAS

0.1

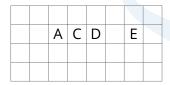
....

recup

Coventry University



```
temp = ['A', 'B', 'C', 'D', 'E']
array<char,5> temp {'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 data

Queue

...

Reca

Coventry University



```
temp = ['A', 'B', 'C', 'D', 'E']
array<char,5> temp {'A', 'B', 'C', 'D', 'E'};
```



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

Abstract dat types

Queue

O+b

_

Coventry University

Removing array elements

C

```
temp = ['A', 'B', 'C', 'D', 'E']
array<char,5> temp {'A', 'B', 'C', 'D', 'E'};
```



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

Array examp LL example

Data structure

Abstract dat types

Queue

Ctack

Otha

Recap

/	4				Е	
		В		1	7	
	/	7_	<u>_</u>	D		
(C –					

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





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



Data

Abstract dat

Queue

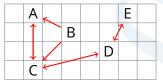
nth a

_

Recap

Removing linked list elements





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



Data structure

Abstract dat

Queue

. .

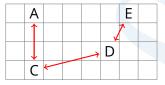
Otho

rree

Recap







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



David Croft

Linked list:

Array example

structures

Abstract dat types

Queue

Stack

Othe

Tree

Reca

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

Abstract dat

21

Stack

Othe

Tree

Recap

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.



Array example

Data structure

Abstract data types

OHAHAS

c. .

Otho

_

Recap

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

Software engineering principal.



Array example

Data structure

Abstract data types

Queues

. .

...

Troc

Recap

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



Array example

Data structure

Abstract data types

Queues

~----

Otho

Troc

Recar

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



Arraye

Array example

Data structures

Abstract data types

Queue

Ctacke

Othe

Tree

Reca

- 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.



Array example

Data structures

Abstract data types

Queue

Stacks

Othe

Traa

Reca

- 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.



Abstract data

Imagine an ADT like a car.

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

Linked lists
Array example

Data structures

Abstract data types

турсз

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.



Linked lists
Array example

Data structures

Abstract data types

Oueue

Stack

Othe

Tree

Recap

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.



Linked lists
Array example

Data structures

Abstract data types

Queue

Stack

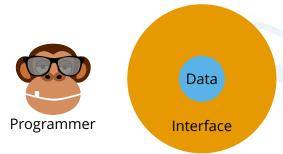
Othi

1166

кеса

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.
- Keep people away from the internal workings/data.





Linked lists
Array example
LL example

Data structures

Abstract data

Queues

Queue

Stacks

Othe

Tree

Recap

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.



Arrays Linked list

Linked lists Array example LL example

Data structure

Abstract dat types

Queues

Othe

Tree

Reca

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.



Oueues

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 gueue 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?



Arrays Linked list

Linked lists Array example LL example

structures
Abstract da

Abstract dat types

Queues

Queuc

Othe

Tree

Reca

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

Abstract data types

Queues

Queue.

0.1

Troc

кесар







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



Array:

Linked lists
Array example

Data structures

Abstract data types

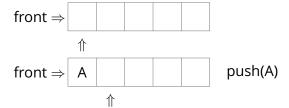
Queues

Stacks

Othe

Tree

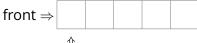
Recap



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



Queues





front \Rightarrow A

push(A)

 \uparrow

front \Rightarrow A В \uparrow

push(B)

Keep track of next free space.

Limited size.

Very similar to stacks.



Linked lists
Array example

Data structures

Abstract data types

Queues

Othe

Troc

Reca

front \Rightarrow

front \Rightarrow front \Rightarrow A push(A)

front \Rightarrow A B push(B)

В

 \uparrow

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

push(C)



Arrays

Linked lists
Array example

Data structures

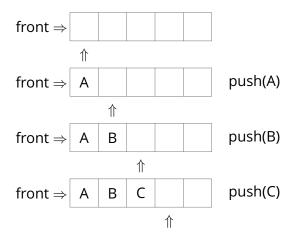
Abstract data

Queues

O+1- --

Troo

Recap



- 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.



Arrav:

Linked lists
Array example

Data structures

Abstract data

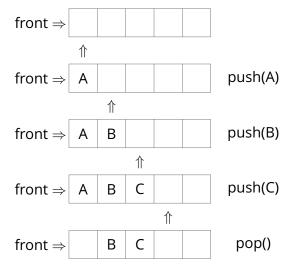
Queues

Queues

Othe

Troc

Reca



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.



Arravs

Linked lists
Array example

Data structures

Abstract data

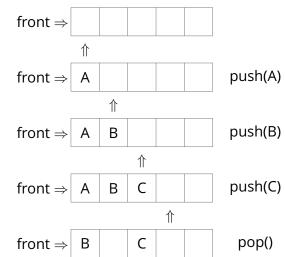
Queues

Queue.

Other

Troc

Reca



 \uparrow

- 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.



Arravs

Linked lists

Data structures

Abstract data types

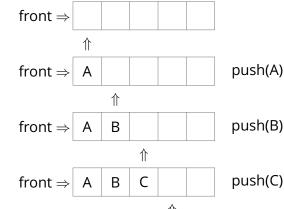
Queues

queues

Other

Tree

Reca



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.

pop()



front \Rightarrow

Arrays

Linked lists Array example

Data

Abstract data

Queues

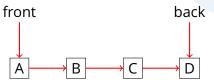
queue.

Stacks

Othe

Tunn

Recan





Array:

Linked lists
Array example

Data structures

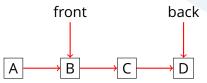
Abstract data types

Queues

Othe

Recar

pop()





Array:

Linked lists
Array example

Data structures

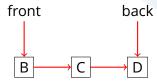
Abstract data types

Queues

Othic

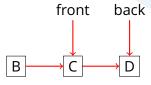
Recar







pop(), pop()





Arrays

Array example

Data structure

Abstract data types

Queues

Queue.

0.1

1100

pop(), pop()





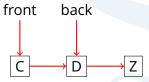
Data structures

Abstract data types

Queue:

0.1

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





Arrays

Linked lists Array example

Data

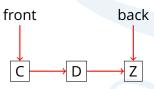
Abstract data

Queue:

Queue.

0.1

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





Arrays Linked list

Linked lists Array example LL example

Data structures

Abstract data

Queue

Queue

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.



Arrays Linked list

Array example
LL example

structures
Abstract da

Abstract dat types

Queue

Stacks

Othe

Tree

Recap

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?



Arrays Linked lists Array example

Data structures

Abstract dat types

Queue

Stacks

Othe

Tree

Recap

A FILO ADT.

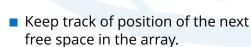
- Ends of the stack are called the top and bottom.
- New elements add to top of stack only.
 - Pushing push(value)
- Old elements removed from top of stack only.
 - Popping pop()
- No cutting in.
- Which would be better for a stack? An array or a linked list?
 - Doesn't matter performance wise.
 - Linked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



Stacks

Array as a stack.





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



Arravs

Linked lists
Array example
LL example

Data structure:

Abstract dat types

Queue

Stacks

0.1

_

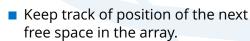
Recap



push(A)

 \uparrow

Α



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



Data structure

Abstract dat types

Queue

Stacks

.....

push(A)

A B 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.



Array as a stack.

Stacks



Α

push(A)

 \uparrow

Α В

push(B)

Α В

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.



Arravs

Array example

Data structure

Abstract dat

Oueue

Stacks



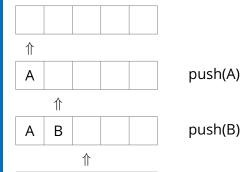
Α

Α

В

В

Array as a stack.



push(C)

pop()

- 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
- Data structures
- Abstract data
- Ougue
- Stacke
- Other
- 1166
- тесар

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



Data structs & types

David Croft

Arravs

Linked lists Array example

Data

Abstract dat

Ougus

Otne

Tree

Recap

Break





Array

Linked lists
Array example
LL example

Data structures

Abstract dat types

Queues

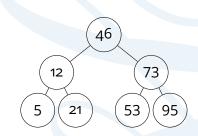
0.1

Trees

Reca

Variation on linked lists.

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





Δrrave

Linked lists
Array example

Data structures

Abstract data

Queues

Queues

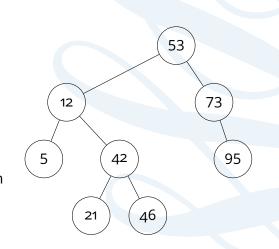
041-

Trees

Recap

 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.





Δrrave

Linked list

Data structures

Abstract data

Queues

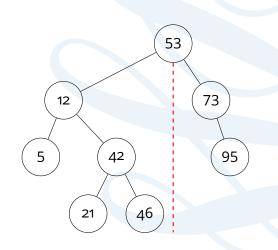
Queues

O+l- -

Trees

Reca

- 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...





Balance

Data structures

Abstract dat types

Queues

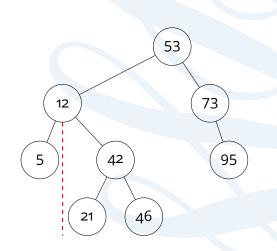
Stacks

Otne

Trees

Recap

- 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.





Arrays

Linked lists

Data structures

Abstract dat types

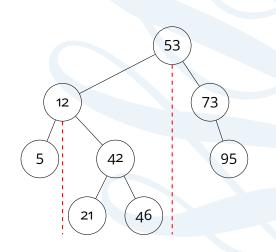
Queues

Othe

Trees

Recap

- 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.





Arrays Linkod lists

Array example

Data structure

Abstract data

Ougue

Queue

. .

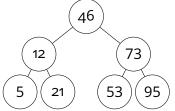
Othe

Trees

Recap

Important that you keep your BSTs balanced.





Degenerate tree.





Data structure

Abstract dat types

Oueues

Stack

Otho

rree

Recap

Everyone

- Need to understand the structures before we can pick the right one.
- Different data structures have very different characteristics.
- Huge effect on efficiency of your code.
- If you pick the right ADT it can save you a lot of code.
 - E.g. why write code to check for duplicates? Use a set and they can't exist.
 - **E**.g. why write code to find the most recent addition to a list, use a stack.



Arravs

Array example

Data structures

Abstract dat types

Queue

Ctooko

Other

Tree

Recap

- 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.

Queue.

FIFO.

Using an array as one.

Using a LL as one.



Data structs & types

David Croft

Arravs

Linked lists
Array example

Data

Abstract data

OHALIS

. .

Tree:

Recap

The End

