

Copyright

© 2016 PG Online Limited

The contents of this unit are protected by copyright.

This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it are supplied to you by PG Online Limited under licence and may be used and copied by you only in accordance with the terms of the licence. Except as expressly permitted by the licence, no part of the materials distributed with this unit may be used, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise, without the prior written permission of PG Online Limited.

Licence agreement

This is a legal agreement between you, the end user, and PG Online Limited. This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it is licensed, not sold, to you by PG Online Limited for use under the terms of the licence.

The materials distributed with this unit may be freely copied and used by members of a single institution on a single site only. You are not permitted to share in any way any of the materials or part of the materials with any third party, including users on another site or individuals who are members of a separate institution. You acknowledge that the materials must remain with you, the licencing institution, and no part of the materials may be transferred to another institution. You also agree not to procure, authorise, encourage, facilitate or enable any third party to reproduce these materials in whole or in part without the prior permission of PG Online Limited.

Data Structures

Teacher’s Guide

Introduction

This teacher’s guide contains a detailed lesson plan to accompany the set of PowerPoint slides and worksheets for each lesson.

The lessons are designed to form a basis for ideas for the teacher and should be adapted to suit the teaching style and preferences of the individual teacher, and the resources and nature of the individual school or Computing / ICT department.

The material supplied for this unit includes:

* 7 PowerPoint presentations, each designed to cover one lesson
* 7 worksheets
* 6 homework sheets
* An end-of-unit test for assessment purposes
* Python and VB.net programs to demonstrate implementation of some of the data structures and algorithms

Summary

The unit is subdivided into seven topics plus a test. It covers all of Section 4.2 of the AQA A-Level specification 7517. (Arrays, records and files are covered in AS Unit 1.) The unit gives practical and worked examples of each of the different abstract data structures including queues, stacks, lists, graphs, trees, hash tables and dictionaries. The function and practical application of each data type is discussed, with pseudocode and coded program solutions for relevant algorithms in VB and Python. Vectors and dot products and their application are covered in a final topic before a comprehensive examination-style assessment of the understanding across the whole unit.

Learning Outcomes for the unit

**At the end of this Unit all students should be able to:**

* describe the concept and uses of a queue, stack, list, graph, tree, hash table, dictionary and vector
* describe typical uses of these data structures
* define a rooted tree and a binary tree
* be able to apply a simple hashing algorithm
* describe what is meant by a collision and how collisions are handled using rehashing
* perform vector addition and scalar multiplication

**Most students will be able to:**

* describe the creation and maintenance of data within queues, stacks and hash tables
* describe the characteristics of an array-based queue, circular queue and priority queue
* know how an adjacency matrix and an adjacency list may be used to represent a graph
* be able to compare the use of adjacency matrices and adjacency lists
* be able to describe and apply a number of different hashing algorithms
* describe different notations for specifying a vector
* calculate the dot product of two vectors

**Some students will be able to:**

* describe and apply the following operations to a linear, circular and priority queue: add an item, remove an item, test for empty queue, test for full queue
* describe and apply the following operations to a stack: push, pop, peek or top, test for empty stack, test for full stack
* describe the convex combination of two vectors
* generate parity given two vectors u and v over GF(2)

Previous Learning

Students need to have studied the first year of the A Level course and have some programming experience.

Suggested Resources

The textbook *AQA A Level Computer Science (Year 2)* or *AQA A Level Computer Science* (AS and A Level Year 2 in one volume).



A complete course text that provides a comprehensive understanding of each topic in both years of the new AQA A Level Computer Science specification. It is presented in an accessible and interesting way, with many in-text questions to test students’ understanding of the material and ability to apply it.

The complete book is divided into 12 sections, each containing roughly six chapters. Each chapter covers material that can comfortably be taught in one or two lessons. It will also be a useful reference and revision guide for students throughout the AS and A Level courses.

Two short appendices contain A Level content that could be taught in the first year of the course as an extension to related AS topics.

Each chapter contains exercises, some new and some from past examination papers, which can be set as homework. Answers to all these are available to teachers only in a Teachers Supplement which can be ordered from our website www.pgonline.co.uk.

**This book has been selected for AQA’s official approval process.**

Vocabulary

Elementary data type, composite data type, abstract data type, encapsulation, information hiding, static data structure, dynamic data structure, heap, overflow, underflow

Queue, circular queue, priority queue, First In, First Out (FIFO), enqueue, dequeue

Append, push, pop, stack, Last In, First Out (LIFO), call stack, stack frame, parameter, return address

Hashing, hash table, collision, mid-square method, folding method, dictionary

Graph, edge, arc, vertex, node, directed graph, digraph, undirected graph, weighted edge, adjacency matrix, adjacency list, Page Rank algorithm

Tree, root, child, parent, subtree, leaf node, binary search tree, pre-order, in-order and post-order traversal

vector, scaling, convex combination of vectors, dot product, Galois field, GF(2)

Assessment

Assessment will be by means of regular homework and a final assessment with examination style questions.

A few points to note:

These are not live assessment questions. They have all been created from scratch for this scheme of work. We cannot guarantee the areas covered in the test will cover all areas that could come up in any given exam paper. That being said, when producing the test the following have been carefully taken into account:

* the range of questions is designed to elicit the understanding of students from E-A\* grade.
* appropriate command words and language is used across the range of questions (list, describe, state, discuss, explain…)

Topic plans

|  |  |  |
| --- | --- | --- |
| Topic1 | Queues |  |
| Preparation: Print out several copies of **Bus Cards.doc**. This one-page document contains three sets of cards that can be cut out and used (i.e. enough sets for 3 students). The items represent buses to towns around the country, an empty queue and pointers to the Front and Rear of the queue. Buses queue up in order of arriving and leaving the station.  Print out and cut up one set of cards for each student – 9 buses, 2 pointers and one empty queue.  Learning Objectives:   * Understand the concept of an abstract data type * Be familiar with the concept and uses of a queue * Describe the creation and maintenance of data within a queue (linear, circular, priority) * Describe and apply the following to a linear, circular and priority queue   + add an item   + remove an item   + test for an empty queue   + test for a full queue | | |
| Content | | Resources |
| **Starter:**  **Introduction to data structures**  Show the first slide and discuss the three categories of data type (elementary, composite and abstract). Introduce the concept of abstraction, and remind them of the terms encapsulation and data hiding which they should have met during the first year.  **Main:**  Introduce the concept of queues with some examples from real life, and the operations that can be performed on queues. Students should attempt to identify the four different operations that need to be implemented for a queue ADT, namely   * Add item to the rear of the queue * Remove item from the front of the queue * Check if the queue is empty * Check if the queue is full   **Exploring four operations on simple array queue**  The **bus cards** can be used to model a queue. Show the slide telling student which buses to add and remove, and discuss the state of the queue after 3 buses have been added and two removed.  The space in this queue is only used once. When the end of the queue is reached, it becomes unusable.  The work of the **front** and **rear** pointers should be discussed. In the examples; **front** points to the first item in the queue (i.e. the next item to remove) and **rear** points to last item in the queue.  When an item leaves the queue, it is removed and then **front** is incremented.  When an item is added, **rear** is first incremented and then the item is added to the space pointed at by **rear**.  Discuss where the pointers will be when the queue is initially empty. Where should the pointers be after the first item is added? Which pointer was updated? The answer is that **rear should be initialised to -1, and front to 0**. After one item is added, both **front** and **rear** point to the item.  Note that this is just one possible implementation – in an alternative implementation, **rear** points to the next free space in the queue.  Discuss what happens when the queue is empty or full. We can’t add to a full queue or remove an item from an empty queue. The maximum size **maxSize** needs to be specified, and a variable **size** holding the number of items in the queue updated whenever an item is added or removed  **Worksheet 1 Task 1**  Give out Worksheet 1 and ask students to complete Task 1, individually or in pairs. Give help where needed and go over the answers when they have finished  **Problems with implementing a queue as an array**  Students should be in a position to answer the questions designed to help them identify the shortcomings of this particular data structure. They may well identify the need for a circular queue or a dynamic data structure such as a **list**.  **Modelling a circular queue**  The bus cards can be used to model a queue.  The pointers wrap round back to the beginning to reuse any freed space. A simple scenario of add/remove is illustrated on the slides.  As before, **front** points to the next item to remove and **rear** points to the last item in the queue. Discuss the role of the MOD function in making the pointers wrap round, guaranteeing that an index outside the range of the array cannot be generated.  **Worksheet 1 Task 2**  This task allows students to manipulate a circular queue.  **Pseudocode**  Let the students try and develop the pseudocode for **isEmpty** and **isFull** for a queue **q** implemented as an array of size **maxSize**.  This could be done individually or in groups before giving the pseudocode on the following slide. Discuss any alternatives that students have come up with. A shorter alternative to the IF statement in SUB isEmpty is  SUB isEmpty  RETURN (size = 0)  ENDSUB  (or in Python, return (size == 0))  And in SUB isFull,  RETURN (size = maxSize)  Students can then look at the pseudocode subroutine for enqueueing an item and try writing the equivalent one for dequeueing.  SUB dequeue  IF size = 0 THEN  print “Queue empty”  item = Null  ELSE  item 🡨 q[front]  front = (front + 1) MOD maxSize  size = size - 1  ENDIF  RETURN item  ENDSUB  Discuss alternative algorithms/pseudocode.  **Modelling a priority queue**  Examples –   * + Hospital triage – those with more serious injuries or illness see the doctor more quickly   + Amazon Prime – membership with monthly subscription ensures quicker delivery   + Airport check-in – those with flights leaving sooner are processed more quickly   The bus cards can be used to model a priority queue. For this, it is much simpler to assume it is implemented using a **built-in dynamic data structure** such as a **list** rather than a circular queue. When an item is deleted from the front of the list, the new front item will automatically be at q[0]. There is no limit to the size of a list (barring memory restrictions) but it would of course be possible to limit the size of the queue using **maxSize** and **size**.  Lists will be covered in the next lesson but students will probably already be familiar with them from practical programming work.  There is a Python implementation of a queue using a list, in the folder.  **Worksheet 1 Task 3**  This task allows students to manipulate a priority queue. Discuss answers when students have completed the task.  **Plenary**  Recap with a comparison of the simple array queue, a circular queue, and a priority queue.  Give out **Homework 1**.  **Extension**  Some students may like to look at the following Internet sites, which explore queueing theory and a recent research outcome.  <http://sciencenordic.com/queues-move-faster-if-last-person-served-first>  <http://www.bbc.co.uk/news/magazine-34153628> | | PowerPoint presentation Data Structures T1 Queues.pptx  Bus Cards.doc  Data Structures Worksheet 1 Queues  Data Structures Worksheet 1 Answers  circularQueue.py  dynamicQueue.py  DynamicQueue (vb programs)  StaticQueue (vb programs)  Data Structures Homework 1  Data Structures Homework 1 Answers |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Topic 2 | Lists | |  | |
| Learning Objectives:   * Explain how a list may be implemented as either a static or dynamic data structure * Show how items may be added to or deleted from a list | | | | |
| Content | | | Resources | |
| **Starter:**  Start by asking students to give you examples of elementary, composite and abstract data types (ADT), which were described in the last lesson.  (Answers could be – Elementary: integer, real, char; Composite: string, array, record; Abstract: list, queue, stack)  Remind students of the ADT **queue** and the methods needed to implement it. Discuss with the aid of the slide headed “List – an ADT” what is meant by an ADT and the principles of **data hiding** and **encapsulation**; the user can use the data structure without knowing any details of how it is implemented. For example, when an item is removed from the front of a list, what actually happens? Are items shuffled along, or are pointers moved? We don’t generally need to know.  The only restriction on the size of a list may be the size of memory – sooner or later a program will crash with an “out of memory” error if too many items are added (possibly because the program is in a loop). The **heap** is briefly mentioned here but is beyond the scope of this specification.  **Main**  Ask students for examples of lists in everyday life and in computer systems. Some suggestions are given on the next slide.  A **list** in a language such as Python is a dynamic data structure, meaning that it can grow as needed. For this reason, lists are very useful for implementing other ADTs such as stacks, queues and trees. What operations do we need to be able to perform on a list in order to implement a queue, for example?  Students should be able to suggest testing for an empty list, adding to the end of a list and removing from the front of a list. The next slide lists some operations that can typically be performed on an inbuilt list structure.  Give out **Worksheet 2** and ask students to do **Task 1**. If they are able to implement these instructions in a programming language, that would be good practice. Are there any other list methods or functions they know of? (**sort**() is one possible operation, **reverse**() is another.)  Go over the answers when students have finished. All these operations are commonly available in a programming language with a built-in dynamic list ADT, and the programmer does not need to know how they are implemented.  **Implementing a queue as a list**  A dynamic data structure such as a list makes it easy to implement a queue. The queue can be allowed to grow for as long as there is space in memory – alternatively a maximum **maxSize** may be specified. It is not necessary to have a variable **size** since the length of the list can be found with a function such as **len(aList)**.  If **maxSize** has been defined, then a function **isFull** would be useful.  Ask the students to write pseudocode for queue operations implemented using a list. Answers on next slide. Note that the statement  RETURN (len(q) = maxSize)  will return True or False depending on whether len(q) is equal to maxSize. Ask students what will be returned by the statement RETURN (len(q) = 0)  (Answer: True or False)  If some students would like a challenge they can try implementing a priority queue using a list.  There is a Python program called **Priority queue.py** in the folder.  **Operations on lists**  Merging, sorting and searching, are all very common list operations.  **Task 2, questions 2 and 3 of the Worksheet** gives students some practice with manipulating lists. Understanding how a merge works will be useful for the next unit on Algorithms, where the merge sort is covered.  The bus cards can be used as a kinaesthetic activity to do a manual merge operation with two sorted lists before writing the algorithm.  **Plenary**  Go over the answers to the worksheet and briefly review the operations that are possible on a built-in **list** ADT in a programming language the students are familiar with.  Give out **Homework 2**. | | | PowerPoint presentation Data structures T2 Lists.pptx  Data structures Worksheet 2 Lists  Data structures Worksheet 2 Answers  Priority queue.py  countNumbers 80-100.py  countNumbers80-100.vb  MergeLists.py  mergeLists.vb  Data structures Homework 2 Lists.docx  Data structures Homework 2 Answers.docx  Ordered list (homework qu 2).py  Ordered\_List.vb | |
| Topic 3 | | Stacks | |  |
| Learning Objectives:   * Be familiar with the concept and uses of a stack * Be able to describe the creation and maintenance of data within a stack * Be able to describe and apply the following operations: push, pop, peek (or top), test for empty stack, test for full stack * Be able to explain how a stack frame is used with subroutine calls to store return addresses, parameters and local variables | | | | |
| Content | | | | Resources |
| **Starter:**  Remind students about the concept of abstraction and that stacks are an abstract data type. The methods used to implement each of these data structures can be used without knowledge of how they work.  The data (e.g. the pointers) used in the implementation of the queue are hidden from the user.  Introduce the concept of a stack. Plates or a pile of books are a good example.  Contrast LIFO operation of stacks with FIFO of queues.  **Main**:  Based on an understanding of a stack, students should be able to identify the operations that need to be performed on a stack:   * Push * pop * isFull * isEmpty   **peek** and **size** are two other operations which could be mentioned.  **Using a stack ADT**  Ask the students to write an algorithm or pseudocode to reverse the elements of a list using a stack. (Answer on next slide).  How can a stack be implemented? The simplest way is to use the built-in list ADT, as we did for a queue. The list operations **append** and **pop** correspond to push and pop for a stack.  Give out **Worksheet 3** and ask students to do the first two questions.  Go over the answers when they have finished.  **Overflow and underflow**  Discuss with students the terms **overflow** and **underflow**.  Ask students what should be checked in the code for pop() and push() before actually attempting these operations.  **Call stack**  The call stack provides the mechanism for passing parameters and return addresses to subroutines, and keeping track of local variables within subroutines. The programmer using a high-level language is unaware of the use of a call stack since all the details are abstracted away. When we look at high-level code, much of the work being done is hidden from us.  **Subroutine calls and execution**  Discuss with students the high-level logic of subroutine calls and returning from a subroutine.  The stack frame holds **parameters**, **return address** and **local variables**.  The ordering of the pushing/popping is important.  Coming out of a subroutine is the reverse of going in.  **Parameter** **passing**  Students are going to work on an example to help them understand how the call stack works. The slide headed **Local variables** shows some code. The objective is to describe what happens to the stack when line 148 is executed.  The next slide, **The call stack**, describes what happens.     * **When the subroutine on line 148 is called:**   - Parameters to subroutine call are pushed onto the stack  *Value of num1 is pushed onto stack*  *Value of num2 is pushed onto stack*  *Stack = (12) (7)*  - The address of the line to return execution to (return address) is pushed.  *149 is pushed onto the stack*  *Stack = (12) (7) (149)*  - Execution passes to the findMax subroutine   * **When the findMax subroutine is entered**   - Local variables are given stack space  *max is pushed onto the stack*  *Stack = (12) (7) (149) (0)*   * **When findMax executes**   - Local variables are updated  *Line 81 causes max to be updated to the value of p1*  *Stack = (12) (7) (149) (12)*  - Return value of max is set  *The value of max is popped and held*  *Stack = (12) (7) (149)*   * **When the findMax subroutine completes**   - The return address is popped and held  *149 is popped*  *Stack = (12) (7)*  - The parameters for the findMax subroutine are popped  *Value p2 is popped*  *Value p1 is popped*  - Execution is directed back to the return address that is being held   * **Execution resumes at line 149**   The detail is not important here – the main thing that the students need to get from this is the function of the call stack when subroutines are called, and the contents of a stack frame.  **Optional extension for Visual Studio users:** If you want to go deeper into the function of the call stack, you can use Visual Studio to show the equivalent assembly code for each line of high-level line of code. Put a breakpoint on the line. When the debugger hits the line, press Ctrl-Alt-D. This will show you something like this:  The address in the ‘call’ instruction is the address of the subroutine. The move instructions before the ‘call’ are the manipulation of the parameter. Unfortunately, VB.Net doesn’t compile using push/pop, but these moves are equivalent. The return address that is put on the stack is the address of the line after the ‘call’ instruction.  This image shows the returning of a value from a subroutine. Notice the two ‘mov’ instructions after the call. These move the contents of a register (the result) into a location in memory.    Note: Please highlight that a high-level programming language is not translated to assembly and then translated to machine code. The compiler creates the machine code (object code). In order to show the assembly code equivalent of a line, the IDE disassembles the object code.  **Nested subroutines**  Ask students what happens when subroutine calls are nested. For example, sub1 calls sub2 calls sub3. The stack will continue to build, with a new stack frame being added with each call. Once a subroutine completes, items are popped from the top stack frame.  Ask students to complete **Task 2** on **Worksheet 3**.  Go over the answers.  **Plenary**  Go over the operations necessary to implement a stack, and the role of a stack in holding return addresses, parameters and local variables when subroutines are called.  Give out **Homework 3.** | | | | PowerPoint presentation Data structures T3 Stacks.pptx  Data structures Worksheet 3 Stacks  Data structures Worksheet 3 Answers  Palindrome.py  Palindrome.vb  Data structures Homework 3 Stacks  Data structures Homework 3 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 4 | Hash tables and dictionaries |  |
| Learning Objectives:   * Be familiar with a hash table and its uses * Be able to apply simple hashing algorithms * Know what is meant by a collision and how collisions are handled using rehashing * Be familiar with the concept of a dictionary * Be familiar with simple applications of a dictionary | | |
| Content | | Resources |
| **Starter**  Show the first two slides on searching, point out what a common operation searching is and ask for examples of large data sets that need to be searched for a particular item very quickly. The TV Licencing Authority, NHS, police databases, vehicle registrations, exam boards, banks, supermarkets, gas, water, electricity suppliers, Inland Revenue, etc., all have huge databases which need to provide more or less instant access to particular records.  **Main**  How is the data held so that items can be quickly retrieved? Ask students for suggestions. Sequential search is obviously hopeless for a large database. Binary search is much quicker but the list has to be kept in sequence, which is itself a time consuming operation.  **Hashing**  Introduce the concept of hashing:   * Large data sets can be organised so that each record is assigned a unique address * This unique address can be calculated using an algorithm * The algorithm uses some part of the record (usually the primary key) to map the record to a unique destination address * The algorithm to implement the mapping is known as a **hashing algorithm**   Hashing enables searching and updating any entry very quickly, with one calculation and one physical access  For example, suppose exam centre numbers are being stored.   * The centre numbers are 5 digits long * The maximum number will be 99999 * The dataset will contain only some of the possible numbers; we don’t know which are there and which are not * After looking at the data, we determine that only one in 10 of the possible centre numbers are ever assigned. This still gives one thousand distinct source records * The first estimate for a suitable size for the hash table might be 1000 x 1.5 = 1500 possible slots   **A simple hashing algorithm**  Move on to discuss the example of a simple hashing algorithm. Students will see that two keys may hash to the same address. This is known as a **collision** and unless there is space for every possible key, which is generally impractical, collisions will always occur.  Ask for suggestions as to how collisions can be minimized.   * increase the number of spaces available in the data set (the more crowded the database is, the more likely a collision will occur) * try a different hashing algorithm   **Dealing with collisions**  Discuss the methods of putting an item in the next free space, or incrementing the ‘skip’ value. The last sentence on the slide says:   * Note that the size of the skip value must be such that all slots in the table will eventually be reached, or part of the table will be unused   Can students explain this? The point is that if an attempt is made to store a record and no free space can be found, the table is effectively “full” even if there are many free spaces left. The table will then have to be enlarged and reorganised.  Give out **Worksheet 4** and ask students to complete **Task 1**.  Show the slide headed **Avoiding problems** and discuss answers when completed.  **Searching the hash table**  Let students work out where key 70 should be stored. (Slot 4) It’s not there, so where is it? This time it’s in the next free slot.  Where should item 58 be? (Slot 3) Where is it? Once you reach a free space, you know that the item is not in the table.  Go through the pseudocode algorithm on the next slide.  **Other hashing algorithms**  Go through the mid-square and folding methods. In an exam, students may be required to work out a hash address but the hashing method will always be described, so no memorisation is required.  The mid-square and folding methods are briefly described.  Alphanumeric keys can be hashed by converting them to ASCII or Unicode and applying any of the hashing algorithms described.  Ask students to complete **Task 2** on the worksheet, which gives practice in different hashing methods.  **The dictionary ADT**  A dictionary is a collection of **Key:data** pairs, useful for any application where you have a key and need to find the corresponding data. For example, word definitions or translations, host names (e.g. www.bbc.co.uk) to IP addresses, user IDs and associated passwords  Note that the data in the **key:data** pairs can be any type of elementary, composite or abstract data type, including a list, array or dictionary.  Ask students what operations are required to implement the dictionary ADT. Answers on next slide. Languages such as Python and VB have a built-in dictionary ADT.  Ask students to complete **Task 3** on the worksheet.  **Plenary**  Show the final slide to make sure students have grasped the main points of the lesson.  Deleting from a built-in dictionary data structure is handles by the system, so there will be a built-in “delete” function such as del dict(key). If the key cannot be found, an error will result.  Students should recall that deleting from a hash table will normally involve marking the item as deleted rather than actually deleting it. The space can then be reused if another key hashes to the same address.  Give out **Homework 4.** | | PowerPoint presentation Data structures T4 Hash Tables & Dictionaries.pptx  Data structures Worksheet 4 Hashing  Data structures Worksheet 4 Answers  Data structures Homework 4 Hashing  Data structures Homework 4 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 5 | Graphs |  |
| Learning Objectives:   * Be aware of a graph as a data structure used to represent complex relationships * Be familiar with typical uses for graphs * Be able to explain the terms: graph, weighted graph, vertex/node, edge/arc, undirected graph, directed graph * Know how an adjacency matrix and an adjacency list may be used to represent a graph * Be able to compare the use of adjacency matrices and adjacency lists | | |
| Content | | Resources |
| **Starter**  Ask students what they understand by the term ‘graph’. They will probably describe a line graph or a bar chart. Explain that the graphs described in this topic are not related in any way to this type of graph – perhaps they could come up with a better term than ‘graph’ for what is being described, for example the connection between Facebook friends or geographic towns.  Show the slide and ask, who are Theo’s friends? Who are Mark’s friends?  **Main**  Graphs have many applications in Computing.  Discuss with students the different uses of graphs to solve problems. See how many the students can identify for themselves. They will probably get the mapping application because of the popularity of GPS devices, such as Garmin and TomTom. Graph applications are listed on a slide near the end of the presentation.   * Maps – nodes are landmarks; edges are routes; a game character takes the shortest path between landmarks * Computer networks – nodes are routers; edges are connections; routing network packets * Project management – nodes are tasks; edges represent order; represent different approaches to ordering the tasks to complete the project * Chemistry – nodes are atoms; edges are connections; modelling of chemical molecules and their interaction * Web pages and links – see textbook for an explanation of Google’s Page Rank algorithm   Start with the example of how a satnav finds a route from A to B. This example can be used to introduce the terms **node**, **edge**, **weight**.  Move on to the diagram of a directed graph, which shows a direction indicated by an arrow. A definition of a graph is given on the next slide.  **Implementing a graph ADT**  **Adjacency matrix**   * The adjacency matrix works for both directed and undirected graphs. The next slides introduce these as exercises for the students. * The adjacency matrices will look slightly different for different cases * Undirected (bi-directional) will be symmetrical [0,1] == [1,0] * Directed will be only half-filled, at most   Use the slide to show how a directed, unweighted graph is represented using an adjacency matrix.  As a class exercise, complete the adjacency matrix for a weighted, undirected graph (answer on next slide)  Give out **Worksheet 5** and ask students to complete **Task 1**.  Discuss the advantages and disadvantages of adjacency matrices in implementing a graph.  **Adjacency list**  Introduce adjacency lists as a way to save the space used by the empty cells.  An adjacency list consists of:   * A list of nodes, with each node pointing to a list of all adjacent nodes   A dictionary could be used to represent a weighted graph, with the key being the node. The value could itself be a dictionary, the list of adjacent nodes could be implemented as a dictionary {node:weight, node:weight, …}  Ask students to complete **Task 2** on the worksheet  The shape of the graph is not important – the connections need to be correct.  **Plenary**  Review uses of graphs. If there is time, students could look up the history of the Page Rank algorithm.  Review terms used.  Give out **Homework 5**. | | PowerPoint presentation Data structures T5 Graphs.pptx  Data structures Worksheet 5 Graphs  Data structures Worksheet 5 Answers  Data structures Homework 5 Graphs  Data structures Homework 5 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 6 | Trees |  |
| Learning Objectives:   * Know that a tree is a connected, undirected graph with no cycles * Know that a binary tree is a rooted tree in which each node has at most two children * Be familiar with typical uses for rooted trees | | |
| Content | | Resources |
| **Starter**  Compare a tree in nature with the ADT tree. Students have already studied graphs, and a tree is a particular type of undirected graph that has no cycles. (A cycle is a closed path, a route which loops back to the start point.)  The left-hand tree on the slide ‘Definition of a tree” has cycles so is not a tree. Ask: “In a tree, is there more than one possible path from one node to another?” The answer is No, however large the tree is.  **Main**  A rooted tree has a root, which is the only node in the tree without a parent. The diagram shows a tree in which J is the root, though strictly speaking it should not have arrows.  The next slide defines terminology relevant to a tree: node, edge, root, child, parent, subtree, leaf. Make sure that students can identify examples of each of these. B, X, T, F. N are all leaves. D, B, M, T, X form one subtree, M, X a second and L, N a third. D, M and L are both parents and children.  Give out **Worksheet 6** and ask students to do the questions in **Task 1.**  Students should enjoy filling in the tree for Grundy’s game. Trees can be used in gaming when the computer is playing a human – the computer can check each path after the opponent has moved to calculate the best strategy. Discuss whether this approach would work for chess or draughts, for example. (In a complex game there are too many possible moves to make this a practical approach. Grundy’s game can be played with any number of coins in the starting pile, and even 8 coins considerably increases the size of the tree.)  **Binary trees**  Move on to binary trees and methods of traversal. The slide headed ‘Traversing a tree’ gives an opportunity to test the students’ understanding of each traversal. The answers are on the next slide – the sub-trees have been ringed in red and as each node is reached, unless it is a leaf it is considered to be the root of a new sub-tree and continues down the left-hand side until a leaf is reached.  The next slide shows another method of figuring out the order in which nodes are visited for each traversal.  Pre-order, the node is visited as it is passed on the left  In order, the node is visited as it is passed underneath  Post-order, the node is visited as it is passed on the right.  You can make up some more trees for the students to practise on if they are having any difficulty.  **Binary search trees**  Efficient searching is one of the major applications of trees. Students should be able to figure out that an in-order traversal will visit the nodes in alphabetic order provided that the tree has been correctly constructed in the first place.  The next slide, **Building a binary search tree**, gives the rules for doing this, with a tree to practise on. (Answer on next slide).  Ask students to complete **Task 2, questions 3 and 4** on the worksheet.  **A balanced binary tree**  Note in passing that a balanced binary tree will give the optimal search efficiency for any item. A data set that is already sorted will produce a completely one-sided tree. Ask students to draw the tree for 7, 12, 14, 18, 22.    **Implementing a binary tree**  Each node has a left pointer, pointing to the node on its left, the data, and a right pointer pointing to the data on its right. A dummy value of -1 is given to any pointer which does not point to a node.  To implement the tree, a two dimensional list, a dictionary, an array of records, or three separate arrays are all possibilities.  There is an error in the left pointer of A[3], it should be -1.  Ask students to complete **Task 3, question 5** on the worksheet.  **Plenary**  Go over the answers to the question.  Show the final slide. Remind students that a tree is a connected, undirected graph with no cycles.  Is every node a root node? Yes:   * Choose any node in a tree and ignore any nodes that are not its descendants. * The structure remains a tree. * This holds even for leaf nodes, where the descendants are all null. * This is an important concept which will be explored in Unit 8.   Data may be organised into a tree to provide an efficient searching method  Ask students to identify and briefly describe pre-order, in-order and post-order traversals.  Give out **Homework 6**. | | PowerPoint presentation Data structures T6 Trees.pptx  Data structures Worksheet 6 Trees  Data structures Worksheet 6 Answers  Data structures Homework 6 Trees  Data structures Homework 6 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 7 | Vectors |  |
| Learning Objectives:   * Be familiar with the concept of a vector and notations for specifying a vector as a list of numbers, as a function or as a geometric point in space * Represent a vector using a list, dictionary or array data structure * Perform operations on vectors: addition, scalar vector multiplication, convex combination, dot or scalar product * Describe applications of dot product of two vectors | | |
| Content | | Resources |
| **Notes**  The specification assumes that students will be familiar with vectors from GCSE mathematics. It focuses on the notation used to represent vectors and the operations to manipulate vectors. These materials start further back. They assume that students will need a bit of a refresher on vectors, so revises the materials from GCSE mathematics. A good refresher website is  <https://www.mathsisfun.com/algebra/vectors.html>  There is a good page on the use of vectors in 3D graphics: <http://www.gamedev.net/page/resources/_/technical/math-and-physics/practical-use-of-vector-math-in-games-r2968>  The notation from the specification adheres to the practice of using ( ) to denote a point in space. It uses [ ] to denote a vector.  **Starter**  Remind students about abstraction and its ability to represent data and operations on that data. Ask them to name all the ADTs that they have already studied. (Queues, lists, stacks, graphs, trees, hash tables, dictionaries). The last one to be studied on this course is **vectors**.  Find out how much the students can remember about vectors from GCSE Maths.  **Main**  Students will need to understand why they’re learning this.  Discuss the examples given on the slide **Why use Vectors?**  Some types of problems within their reach will be:   * Vector graphics   + Images are defined as paths, not distinct bit patterns   + Rules of mathematics allow scaling without distortion * Computer games / simulations   + The combination of moving forces, represented by vectors, can be resolved into a single path by using the same mathematical rules.   + For example, a ship steering against the wind and sea. * Multiple dimensions   + Can represent 2, 3, or more dimensions   + The idea of more than 3 dimensions may be confusing for students, but more on that later. Just assure them, that if we can think it, we can represent it.   **Notation (2-vector** ℝ2 **), (3-vector** ℝ3 **), (4-vector** ℝ4 **), (function f:** S↦ ℝ **) S** maps to ℝ, the set of real numbers.  Students need to be familiar with the different notations that are used to represent a vector. 2- and 3-vectors are easy to visualise as an arrow or a point in space. The notation involving the set of real numbers can be deciphered easily from the superscript.  The use of the **map** function will need to be explained in terms of a location or position analogy.  Discuss the examples on the slide for representing a vector as an arrow and as a function. As a class exercise, do the exercises on the next slide. (Answers on following slide).  **Representation**  It’s fine to not know what a vector represents. Computer scientists don’t actually always need to know what they represent to be able to manipulate them in a program. However, we do have to find ways of representing them inside the computing device.  **Vector – representation in a programming language**  Examples from Python and VB.Net are given to show how a vector can be represented by a list of numbers, by a 1-D array, and by a dictionary. Note that the most obvious choice for each programming language has been discussed. Python naturally supports lists and dictionaries. VB.Net supports lists, arrays, and dictionaries.  Ask/discuss with students which choices are available in the programming languages that they’re familiar with.  **Addition of two vectors:** The vector addition rules are quite simple, so students can do the exercises on the slide. Answers provided on next slide.  If other examples are required, consider one with negative coordinates.  **Scalar multiplication of a vector:** The important fact is that multiplication by a scalar changes the magnitude (size) of the vector. This is known as scaling. Students can do the exercise on the following slide.  Give out **Worksheet 7** and ask students to do **questions 1 to 4** in **Task 1.** This is designed to make sure that the students have grasped and remembered the concepts and notation covered, and will act as revision notes.  This worksheet has marks given against each question so could be used as a self-marked ‘quiz’. Go over the answers when students have finished.  **Vector- convex combination**  There is a set of vocabulary that is important at this stage. Understanding this interaction of vectors is important in the fields of Geographical Information Systems (GIS) and games. This is highlighted on the following slide.  **Exercise – convex combination**  Students can do the exercise on the slide. The correct answer is on the following slide. Students may need a hint to do both the arithmetic and the graphical method.  **Dot product**  Introduce the dot product, first as a pure mathematical operation. Once understood, then the parity bit application can be tackled.  **Dot product – application**  Because we’re working with bits, we can’t define vectors over ℝ. We have to define our vectors over the set of binary digits {0, 1}.  The name Galois Field is given to a set that consists of a finite number of items. It is named in honour of the French mathematician Évariste Galois who developed a theorem of their existence.  Galois Field 🡺 A finite set of elements 🡺 We want 2 elements 🡺 GF(2) 🡺 The set of elements {0, 1}  It is beyond the scope of this unit to prove that multiplication maps to AND and that addition and subtraction both map to XOR.  **Exercise - parity bit**  Have students calculate the parity bit required to achieve even parity, by finding the dot product of the vectors u and v. u is a vector filled with 1s and v is the vector representing the bit pattern for which we want to find the parity bit.  e.g. u = [1,1,1,1] v = [1,0,1,1]  One way to do this is to set it out so that students can see each step:  u • v = (1\*1) + (1\*0) + (1\*1) + (1\*1)   = 1 + 0 + 1 + 1   = 1 + 1 + 1   = 0 + 1  = 1 This is the parity bit.  Another way to do this would be to use the operators AND and XOR:  *u* • *v* = (1 AND 1) XOR (1 AND 0) XOR (1 AND 1) XOR (1 AND1)  The brackets are provided for clarity but are not needed. The order of operator precedence ensures that the ANDs are done before the XORs.  To calculate odd parity, NOT the result of the even parity calculation.  Ask students to do **questions 5 and 6** in **Task 2.** If time is short, these can be done as homework. Alternatively, students can be asked to revise for the test next week.  **Plenary**   * What 3 types of notation can be used to specify a vector?   + A list of numbers, a function, as a point in space * What 3 abstract data types can be used to represent a vector?   + A list, dictionary, or array * What 4 different types of operations can be performed on vectors?   + Addition, scalar vector multiplication, convex combination, and dot product * What is one application of dot product   + Finding the parity bit | | PowerPoint presentation Data structures T7 Vectors.pptx  Data structures Worksheet 7 Vectors  Data structures Worksheet 7 Answers |

|  |  |
| --- | --- |
| Unit assessment | |
| Learning Outcomes:  Students will   * apply their knowledge in answers to a range of questions * be able to highlight areas of strength and any gaps in their understanding of computers | |
| Content | Resources |
| Students should complete the **Assessment Test**.  These tests have been designed to be printed and answered by hand. Alternatively, they could be uploaded and incorporated into an automated test as part of many modern VLEs. | Data structures Final assessment  Data structures Final assessment Answers |

Acknowledgements

The authors and publisher would like to thank all contributors for their kind permission to reproduce their photographs or images, screenshots of their websites or other copyright material in the PowerPoint Guides.

Crushed cars © Sjoerd van der Wal, www.iStockphoto.com

Google Map directions © Google 2016

Images otherwise stated © Shutterstock

Every effort has been made to trace the copyright holders and we apologise in advance for any unintentional omissions. We would be pleased to insert the appropriate acknowledgement in any subsequent edition of this publication.

This material contains links to relevant websites featured in the teacher’s guide. Every effort has been made to ensure that at the time of distribution, the links remain unbroken, the material remains up-to-date and that links are not inadvertently linked to sites that could be considered offensive. PG Online cannot be held responsible for the content of any website mentioned in this material. It is sometimes possible to find relocated sites by typing the original URL into a browser. Any errors should be reported directly to support@pgonline.co.uk and changes will be made in any subsequent editions of the material.

Artwork



Heathland Bistro © 2015 Andrew Bird

Acrylic on board, 40x40cm

www.abirdart.co.uk