AQA Computer Science A-Level

4.2.1 Data structures and abstract data types

Concise Notes

Specification:

4.2.1.1 Data structures:

Be familiar with the concept of data structures.

4.2.1.2 Single- and multi-dimensional arrays (or equivalent):

Use arrays (or equivalent) in the design of solutions to simple problems.

4.2.1.3 Fields, records and files:

Be able to read/write from/to a text file.

Be able to read/write data from/to a binary (non-text) file.

4.2.1.4 Abstract data types/data structures:

Be familiar with the concept and uses of a:

- Queue
- Stack
- Graph
- Tree
- Hash table
- Dictionary
- Vector

Be able to distinguish between static and dynamic structures and compare their uses, as well as explaining the advantages and disadvantages of each.

Describe the creation and maintenance of data within:

- Queues (linear, circular, priority)
- Stacks
- Hash tables

4.2.2.1 Queues:

Be able to describe and apply the following to linear queues, circular queues and priority queues:

- add an item
- remove an item
- test for an empty queue
- test for a full queue

4.2.3.1 Stacks:

Be able to describe and apply the following operations:

- push
- pop
- peek or top
- test for empty stack
- test for stack full

4.2.4.1 Graphs:

Be aware of a graph as a data structure used to represent more 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.

4.2.5.1 Trees (including binary trees):

Know that a tree is a connected, undirected graph with no cycles.

Know that a rooted tree is a tree in which one vertex has been designated as the root. A rooted tree has parent-child relationships between nodes. The root is the only node with no parent and all other nodes are descendants of the root.

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.

4.2.6.1 Hash tables:

Be familiar with the concept of 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.

4.2.7.1 Dictionaries:

Be familiar with the concept of a dictionary

Be familiar with simple applications of dictionaries, for example information retrieval, and have experience of using a dictionary data structure in a programming language.

4.2.8.1 Vectors:

Be familiar with the concept of a vector and the following notations for specifying a vector:

- [2.0, 3.14159, -1.0, 2.718281828]
- 4-vector over $\mathbb{R}$ written as $\mathbb{R}4$
- function interpretation
- $0 \mapsto 2.0$
- $1 \mapsto 3.14159$
- $2 \mapsto -1.0$
- $3 \mapsto 2.718281828$
- $\mapsto$ means maps to

That all the entries must be drawn from the same field, eg $\mathbb{R}$.

Dictionary representation of a vector.

List representation of a vector.

1-D array representation of a vector.

Visualising a vector as an arrow.

Vector addition and scalar-vector multiplication.

Convex combination of two vectors, u and v.

Dot or scalar product of two vectors.

Applications of dot product.

Data structures

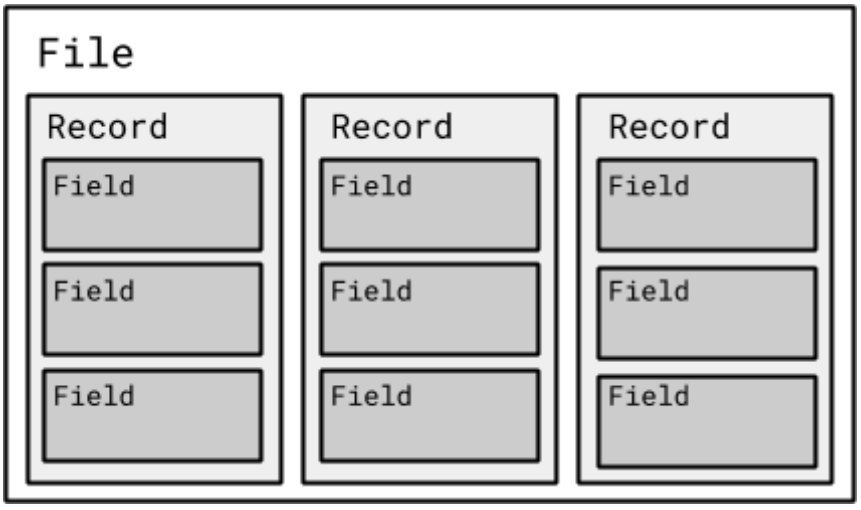
- The containers within which information is stored
- Some are better suited to different types of data than others
- Programmers must decide which of the data structures available is the best to use

Arrays

- An indexed set of related elements
- Must be finite and indexed
- An array's index often starts from zero
- Must only contain elements with the same data type
- Arrays can be created in many dimensions

Fields, records and files

- Information is stored by computers as a series of files.
- Each file is made up of records which are composed of a number of fields.



Abstract data types/data structures

- Don't exist as data structures in their own right
- Make use of other data structures to form a new way of storing data

Queues

- An abstract data structure based on an array
- The first item added to a queue is the first to be removed
- Referred to as "first in, first out" (FIFO) abstract data structures
- Used by computers in keyboard buffers and in the breadth first search algorithm

Linear queues

- Has two pointers, a front pointer and a rear pointer
- These can be used to identify where to place a new item in a queue or to identify which item is at the front of the queue
- The item at the front of the queue has been in the queue for the longest
- Operations that can be performed on a queue include `Enqueue`, `Dequeue`, `IsEmpty` and `IsFull`
- Emptiness can be detected by comparing the front and rear pointers

Circular queues

- A type of queue in which the front and rear pointers can move over the two ends of the queue
- More memory efficient than a linear queue

Priority Queues

- Items are assigned a priority
- High priority items are dequeued before low priority items
- In the case that two or more items have the same priority, the items are removed in the usual *first in, first out* order
- Frequently used in computer systems.

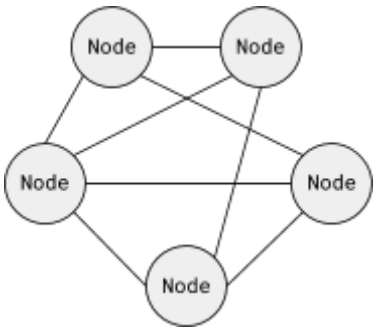
Stacks

- A first in, last out (FILO) abstract data structure
- Often based on an array
- Have just one pointer: a top pointer.
- The item at the bottom of a stack has been in the stack for the longest
- Operations that can be performed on a stack include:
  - `Push` (add an item)
  - `Pop` (remove the item at the top)
  - `Peek`
- `Peek` returns the value of the item at the top of the stack without actually removing the item.
- The functions `IsFull` and `IsEmpty` can also be executed on stacks
- If a `push` command is executed on a full stack, a stack overflow error is thrown
- A similar error, called a stack underflow, can be caused by attempting the `pop` command on an empty stack

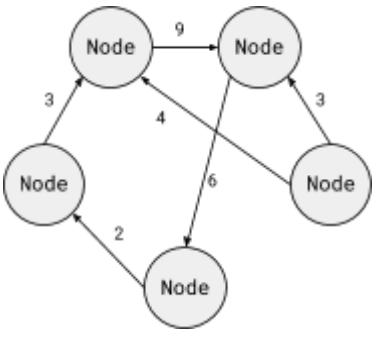
Graphs

- An abstract data structure used to represent complex relationships between items within datasets
- Can be used to represent networks such as transport networks, IT networks and the Internet.
- Consists of nodes (sometimes called vertices) joined by edges (sometimes called arcs)
- A weighted graph is one in which edges are assigned a value, representing a value such as time, distance or cost
- Can be represented in two different ways:
  - Adjacency matrices
  - Adjacency lists

Unweighted, undirected graph



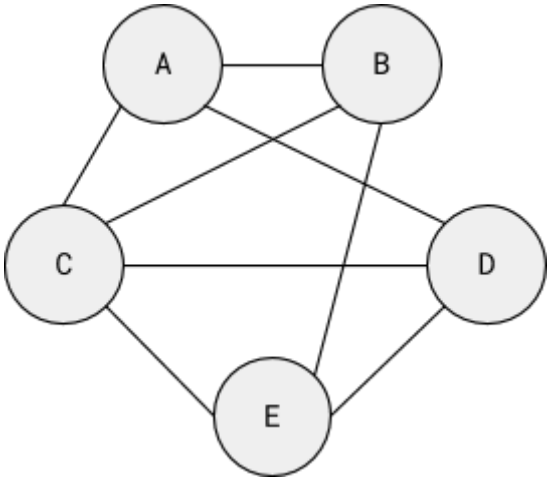
Weighted, directed graph



Adjacency matrices

- A tabular representation of a graph
- Each of the nodes in the graph is assigned both a row and a column

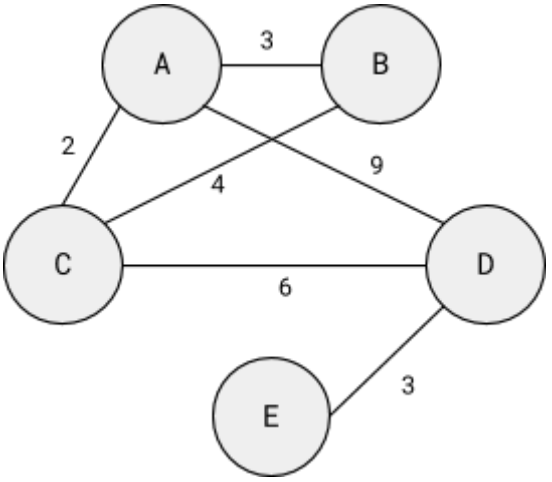
Example: Unweighted graph



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	1	0	1
C	1	1	0	1	1
D	1	0	1	0	1
E	0	1	1	1	0

- 1 is used to show that an edge exists between two nodes
- 0 indicates that there is no connection.
- Adjacency matrices have a characteristic diagonal line of 0s (shown in blue)
- Adjacency matrices display diagonal symmetry, as shown by the green cells

Example: Weighted graph

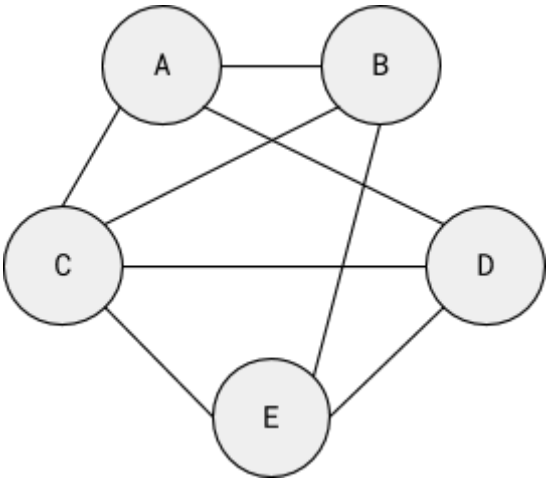


	A	B	C	D	E
A	∞	3	2	9	∞
B	3	∞	4	∞	∞
C	2	4	∞	6	∞
D	9	∞	6	∞	3
E	∞	∞	∞	3	∞

- Adjacency matrices for weighted graphs contain the weight of a connection between two nodes
- If no connection exists, an arbitrarily large value is used
- This value is often expressed as infinity

Adjacency lists

- Represent a graph using a list
- For each node, a list of adjacent nodes is created
- Only records existing connections in a graph

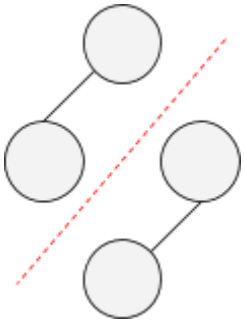


A B, C, D
B A, C, E
C A, B, D, E
D A, C, E
E B, C, D

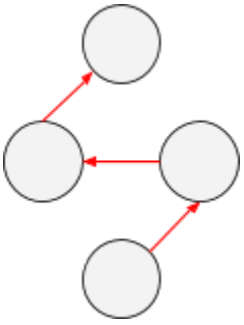
Adjacency matrix	Adjacency list
Stores every possible edge between nodes, even those that don't exist. Almost half of the matrix is repeated data. **Memory inefficient.**	Only stores the edges that exist in the graph. **Memory efficient.**
Allows a specific edge to be queried very quickly, as it can be looked up by its row and column. **Time efficient.**	Slow to query, as each item in a list must be searched sequentially until the desired edge is found. **Time inefficient.**
Well suited to dense graphs, where there are a large number of edges.	Well suited to sparse graphs, where there are few edges.

Trees

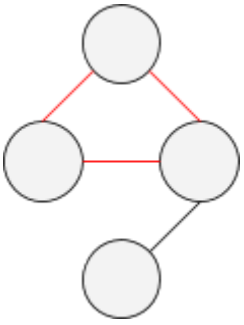
A tree is a connected, undirected graph with no cycles.



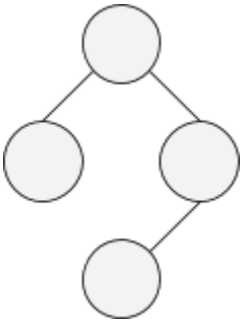
✘ Connected
✔ Undirected
✔ No cycles



✔ Connected
✘ Undirected
✔ No cycles



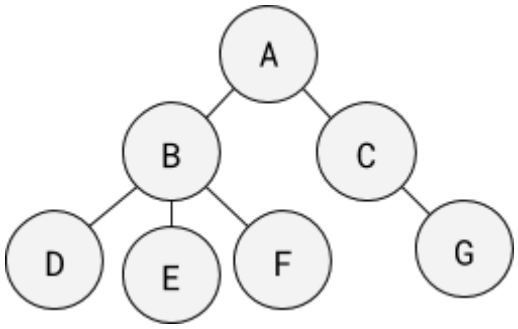
✔ Connected
✔ Undirected
✘ No cycles



✔ Connected
✔ Undirected
✔ No cycles

Rooted trees

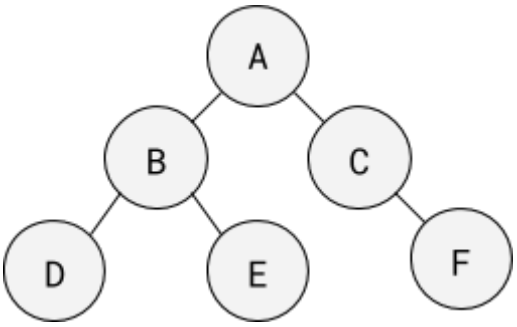
- Have a root node from which all other nodes stem
- The root node is usually displayed at the top of the tree in diagrams
- Nodes from which other nodes stem are called parent nodes
- The root node is the only node with no parent
- Nodes with a parent are called child nodes
- Child nodes with no children are called leaf nodes



Root: A
Parent: A, B, C
Child: B, C, D, E, F, G
Leaf: D, E, F, G

Binary trees

- A rooted tree in which each parent node has no more than two child nodes.



Hash tables

- A way of storing data that allows data to be retrieved with a constant time complexity
- A hashing algorithm takes an input and returns a hash
- A hash is unique to its input and cannot be reversed to retrieve the input value
- A hash table stores key-value pairs
- The key is sent to a hash function, which produces a hash
- Resulting hash is the index of the key-value pair in the hash table
- When an element is to be looked up in a hash table, the key is first hashed
- Once the hash has been calculated, the position in the table corresponding to that hash is queried and the desired information located

Collisions

- Sometimes different inputs produce the same hash
- Well designed hash tables get around collisions by using rehashing
- One simple rehashing technique is to keep on moving to the next position until an available one is found

Dictionaries

- A collection of key-value pairs
- Values are accessed by their associated key
- One application of dictionaries is in dictionary-based compression

Synoptic Link

Dictionary-based compression is covered in more detail under **fundamentals of data representation.**

Vectors

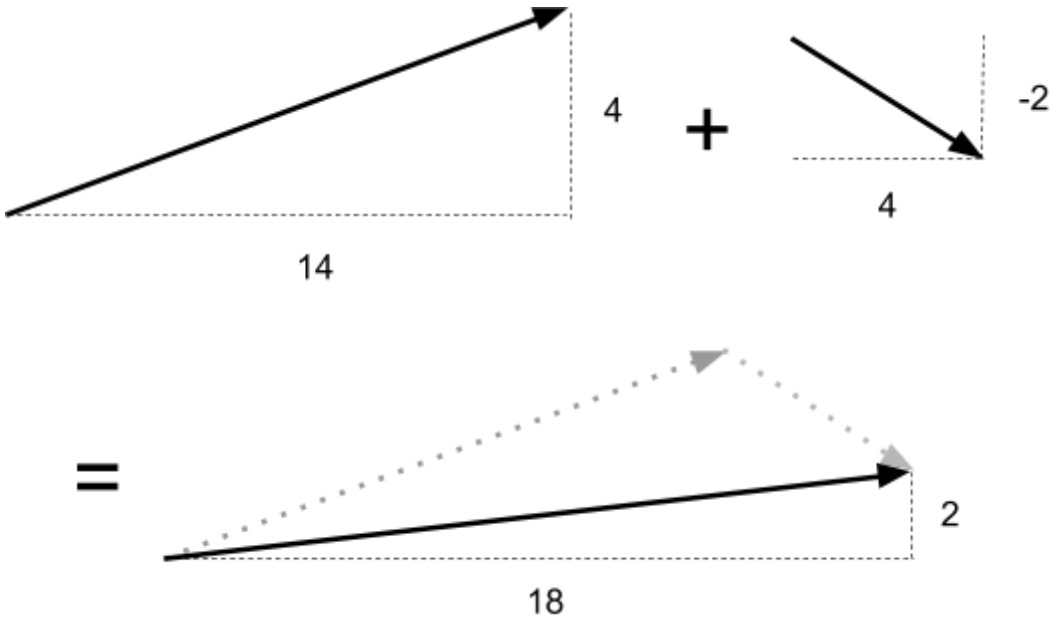
- Can be represented as:
  - lists of numbers
  - functions
  - ways of representing a point in space

As a list of numbers	`[12, 7, 3, 55]`
As a vector space over a field	4-vector over $\mathbb{Z}$ ($\mathbb{Z}^4$)
As a function	$0 \mapsto 12$ $1 \mapsto 7$ $2 \mapsto 3$ $3 \mapsto 55$
As a point in space	(12, 7, 3, 55)

- If viewed as a function, a vector can be represented by using a dictionary
- If viewed as a list, a one-dimensional array would be suitable
- Can be visually represented as an arrow

Vector addition

- Vectors can be added to achieve translation
- With arrows, vectors are added “tip to tail”



- Vectors can also be added by adding each of their components

$$\begin{aligned} & [14,\ 4] \\ + & [4,\ -2] \\ = & [18,\ 2] \end{aligned}$$

Scalar-vector multiplication

- In order to scale a vector, each of its components are multiplied by a scalar
- Scaling a vector affects only its magnitude, not its direction

$$\begin{aligned} & [14,\ 4] \\ \times & \quad 3 \\ = & [42,\ 12] \end{aligned}$$

Convex combination of two vectors

- With two vectors, a and b, a convex combination of the two would be $ax + by$ where x and y:
  - are non-zero numbers
  - are less than one
  - add to 1
- The convex combination of a and b with x and y
  - is formed on the line that would join the tips of a and b
  - splits the line in the ratio chosen for the values x and y

Dot product

- A single number derived from the components of vectors
- Can be used to determine the angle between two vectors
- The dot product of the vectors a and b is notated $a \cdot b$ (said "a dot b")

Let $a = [12, 3]$ and $b = [5, 8]$

$$a \cdot b = (12 \times 5) + (3 \times 8)$$
$$= 84$$

Static and dynamic data structures

- Every data structure is either static or dynamic
- Static data structures are
  - fixed in size
  - most frequently declared in memory as a series of sequential, contiguous memory locations
- Dynamic data structures
  - change in size to store their content
  - store each item of data alongside a reference to where the next item is stored in memory
  - require more work on the part of the computer to set up and use