Module Code
COM00015C

BEng, BSc, MEng and MMath Degree Examinations 2021–22

Department Computer Science
Title Software 1
Issued 09:00am on Wednesday 12^{th} January 2022
Submission due 09:00am on Thursday 13^{th} January 2022
Feedback & marks due Thursday 17^{th} February 2022 at noon
Time Allowed 24 Hours (NOTE: papers late by up to 30 minutes will be subject to a 5 mark penalty; papers later than 30 minutes will receive 0 marks).

Notification of errors in the paper may be made up to **one hour** after the start time. If you wish to raise a possible error it must be done through ⟨`cs-exams@york.ac.uk`⟩ with enough time for a response to be considered and made within the first hour.

Time Recommended THREE hours

Word Limit Not Applicable

Allocation of Marks:
Question 1 is worth 20%, questions 2 and 3 are worth 15% each, question 4 is worth 10%, and question 5 is worth 30%. The remaining 10% are allocated to style, clarity, and quality of code.

Instructions:
Candidates should answer **all** questions using Python 3.8. Failing to do so will result in a mark of 0%. All questions are independent and can be answered in any order. Download the paper and the required source files from the VLE, in the "Assessment> January Exam" section. Once downloaded, unzip the file. You **must** save all your code in the `ClosedExamination` folder provided. **Do not** save your code anywhere else other than this folder.

Submit your answers to the Department's Teaching Portal as a single **.zip** file containing the `ClosedExamination` folder and its sub-directories. Failing to include the `ClosedExamination` folder will result in a loss of 10 marks. Other archival format such as .rar and .tar files will not be accepted and will result in a mark of 0%.

If a question is unclear, answer the question as best you can, and note the assumptions you have made to allow you to proceed. Please inform ⟨`cs-exams@york.ac.uk`⟩ about any suspected errors on the paper immediately after you submit.

A Note on Academic Integrity

We are treating this online examination as a time-limited open assessment, and you are therefore permitted to refer to written and online materials to aid you in your answers.

However, you must ensure that the work you submit is entirely your own, and for the whole time the assessment is live you must not:

- communicate with departmental staff on the topic of the assessment
- communicate with other students on the topic of this assessment
- seek assistance with the assignment from the academic and/or disability support services, such as the Writing and Language Skills Centre, Maths Skills Centre and/or Disability Services. (The only exception to this will be for those students who have been recommended an exam support worker in a Student Support Plan. If this applies to you, you are advised to contact Disability Services as soon as possible to discuss the necessary arrangements.)
- seek advice or contribution from any third party, including proofreaders, online fora, friends, or family members.

We expect, and trust, that all our students will seek to maintain the integrity of the assessment, and of their award, through ensuring that these instructions are strictly followed. Failure to adhere to these requirements will be considered a breach of the Academic Misconduct regulations, where the offences of plagiarism, breach/cheating, collusion and commissioning are relevant: see AM1.2.1 *(Note this supercedes Section 7.3 of the Guide to Assessment)*.

1 (20 marks) Basic Programming Structure

The code must be written in the provided file `question_1.py`.

The aim of this question is to compute the position of the centre of mass C of a set of spheres $\{S_1, ..., S_n\}$. A sphere $S_{i,\ i\in\{1..n\}}$ is defined by its centre of mass (x_i, y_i, z_i), its radius r_i and its unit mass μ_i. The total mass m_i of the sphere S_i is given by:

$$m_i = \frac{4}{3}\mu_i \pi r_i^3$$

The centre of mass $C = \begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix}$ is given by the formulae:

$$c_x = \frac{\sum_{i=1}^{i=n} x_i\ m_i}{\sum_{i=1}^{i=n} m_i}$$
$$c_y = \frac{\sum_{i=1}^{i=n} y_i\ m_i}{\sum_{i=1}^{i=n} m_i}$$
$$c_z = \frac{\sum_{i=1}^{i=n} z_i\ m_i}{\sum_{i=1}^{i=n} m_i}$$

(i) [10 marks] Write a function `centreOfMass(spheres)` that returns the position of the centre of mass of a set of spheres. The parameter `spheres` is a **set** of tuples, where each tuple represents the information of a given sphere.
The tuples are of the form `(x, y, z, radius, unitMass)`, where `x`, `y` and `z` are the coordinates of the centre of mass of the sphere, `radius` is the radius of the sphere, and finally `unitMass` is the mass of 1 volume unit for that sphere. The returned value is the tuple (c_x, c_y, c_z).

The function must raise a `ValueError` if at least one of the following occurs:

- the set `spheres` is empty,
- the radius of a sphere is less or equal to 0,
- the unit mass of a sphere is less or equal to 0.

(ii) [10 marks] Write the **docstring** for this function. Note, the docstring must follow the Google documentation style.

2 (15 marks) List Built-in Data Structure

The code must be written in the provided file `question_2.py`.

To perform an efficient collision detection between objects in a scene, an object is sometime approximated by a sphere-tree as shown in Figure 1. Two objects in a scene collide if any sphere from an object's sphere-tree intersects a sphere from the other object's sphere-tree.

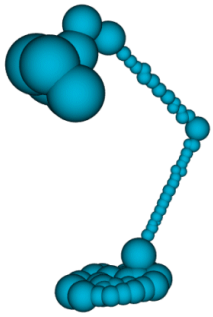


Figure 1: An example of a lamp approximation using spheres.

For simplicity, we will represent an object as a set of spheres rather than a sphere-tree. The collision problem can be formulated as "Two objects collide if any sphere from the object's set of spheres intersect a sphere from the other object's set of spheres".

A sphere S_i can be defined by four values (x_i, y_i, z_i, r_i) where x_i, y_i, and z_i are the coordinates of the centre of the sphere, and the value r_i is the radius of the sphere. Two spheres S_1 and S_2 collide if the sum of their radius is strictly greater than the distance between their centres. The distance δ between the two centres is given by:

$$\delta = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Write a function `collide(objectA, objectB)` that returns `True` if `objectA` collides with `objectB`, `False` otherwise. The parameters `objectA` and `objectB` are sets of tuples, where the tuples are of the form `(x, y, z, radius)`. The values `x`, `y` and `z` are the coordinates of the centre of a sphere, and `radius` is the radius of a sphere. The function must raise a `ValueError` if the format of a tuple is invalid, that is it does not contain exactly 4 floats or the radius is not strictly positive.

Implement a brute force approach, that consists of testing for each sphere in the set `objectA` if it intersects a sphere from the other set `objectB`.

3 (15 marks) Recursion

The code must be written in the provided file `question_3.py`.

Warning: if the implemented function is not recursive, a mark of 0 will be awarded.

T9, which stands for Text on 9 keys, was a predictive text technology used in the late 1990s for mobile phones, specifically those that contained a 3×4 numeric keypad as shown in Figure 2. T9's objective was to make it easier to type text messages. It allowed words to be entered by a single key press for each letter. For example, to type the word `FACES`, the user had to type 5 keys 3-2-2-3-7 with predictive text technology.

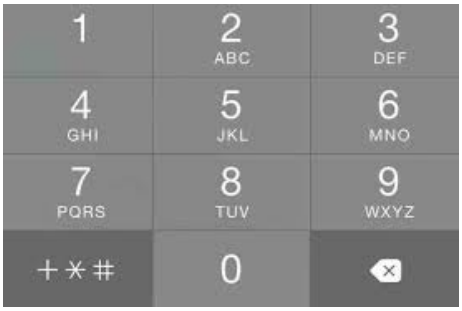


Figure 2: An example of a T9 keypad.

Implement a **recursive** function `t9Words(keysPressed)` that returns a set containing all possible combinations of letters given the set of keys pressed on the phone keypad. The set of keys pressed is passed via the parameter `keysPressed`, which is a string of digits. The keys 0 and 1 are simply ignored as they do not contain alphabet letters. For simplicity, you can assume that the input is a valid string containing only digits. For example:

```
>>> t9Words('23')
{'AD','AE','AF','BD','BE','BF','CD','CE',CF'}
>>> t9Words('123')
{'AD','AE','AF','BD','BE','BF','CD','CE',CF'}
```

Note that in the second example, the returned set is the same as the first example as the key 1 does not have any corresponding alphabet letter.

You can implement your own recursive algorithm or you can implement the algorithm provided in Algorithm 1.

Algorithm 1: A recursive algorithm to get all possible words from a series of digits using T9 predictive text.

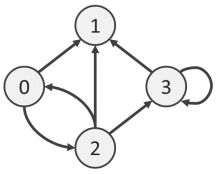
```
Procedure getWords(keysPressed: string, current: string, words:Set)
    if keysPressed is empty then
        add current to words;
        return
    letters ← keypad letters for digit keysPressed[0];
    remove first digit from keysPressed;
    if letters is empty then
        getWords(keysPressed, current, words);
    else
        foreach c ∈ letters do
            getWords(keysPressed, current + c, words);
        end
    end
end

Function t9Words(keysPressed: string): Set
    words ← empty set of strings;
    current ← empty string;
    getWords(keysPressed, current, words);
    return words
end
```

4 (10 marks) Python I/O

The code must be written in the provided file `question_4.py`.

The aim of this question is to read an adjacency matrix of an unweighted digraph from a CSV file. The adjacency matrix is represented by a 2D list of `int` containing only 0s and 1s. The CSV file format for a digraph containing n vertices is n lines, where each line is composed of n values, each separated by a comma. A value 1 in the first line second column means there is an edge from vertex 0 to vertex 1.



(a)

```
0, 1, 1, 0
0, 0, 0, 0
1, 1, 0, 1
0, 1, 0, 1
```

(b)

Figure 3: An example of (a) a digraph and (b) its representation in a CSV file.

Implement a function `readAdjacency(filename)` that reads the CSV file `filename` containing an adjacency matrix, and returns a 2D list representation of the adjacency matrix. The parameter `filename` is a string containing the path to the CSV file. Given the digraph and the CSV file from Figure 3, the returned 2D list would be:

```
[[0, 1, 1, 0],
 [0, 0, 0, 0],
 [1, 1, 0, 1],
 [0, 1, 0, 1]]
```

The function must raise a `FileNotFoundError` if the file path given in the parameter does not exist. In addition, the function must raise an `InvalidFileFormatException` if:

- The file contains values other than 0s and 1s.
- The number of columns and lines are not the same, that is the matrix is not square.

5 (30 marks) User Defined Data Structure

The code must be written in the provided file `question_5.py`.

The ASCII (American Standard Code for Information Interchange) encoding uses 7 bits to represent 128 symbols used for information interchange. Alphabet characters, digits and punctuation are represented using this encoding. A subset of the character and their binary representation is shown in Table 1. Even though the ASCII encoding use only seven bits per symbol, when writing such a symbol in a text file, an entire byte is used (a byte contains 8 bits).

Table 1: Subset of the ASCII (American Standard Code for Information Interchange) character code chart.

Symbol	Binary
A	01000001
a	01100001
?	00111111
2	00110010

If a text of a thousands characters is encoded using ASCII, the file size will be 8000 bits (1000 bytes). There is a more efficient way to store such a text using a bespoke encoding. Given letters in an English text have different frequencies, we could use less bits to encode a letter with high frequency such as 'e', and more bits for a letter with low frequency such as 'z'. Some frequencies are shown in Table 2.

Table 2: Subset of the English Letter Frequency table.

Letter	Frequency
e	12.02
T	9.10
j	0.10
z	0.07

The aim of this question is to build a class `Encoding` that can use a bespoke encoding to encode and decode a text. A binary tree like the one shown in Figure 4 is used to store the bespoke encoding.

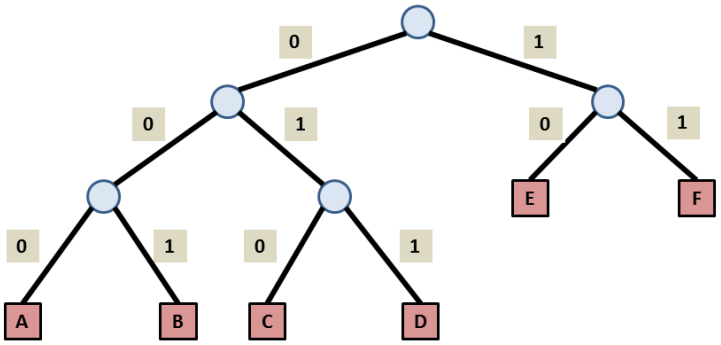


Figure 4: Example of an encoding tree containing 6 symbols.

For this problem, we have chosen a list to represent the binary tree. The root of the tree is at index 0. The left child of the node at index n is at index $2n + 1$, and the right child is at index $2n + 2$. In our structure, an internal node is represented by an empty string, a leaf is a node where the left and right children are `None`. The String stored at this node contains a single character representing a symbol of the alphabet.

The tree shown in Figure 4 is represented by the list

```
['','','','','','E','F','A','B','C','D',None,None,None,None,
 None,None,None,None,None,None,None,None]
```

(i) [5 marks] Implement a class `Encoding` with the method `__init__` taking a list representing a binary tree as parameter. A **copy** of the list passed in the parameters should be assigned to the **instance attribute** `_encoding`. For simplicity, we assume that the parameter provided has the correct format and you don't need to check it.

(ii) [15 marks] When decoding a series of bits using the encoding tree `_encoding`, as a common convention, bit '0' represents following the left child and bit '1' represents following the right child (see Figure 4). Generally speaking, the process of decoding is simply a matter of translating the stream of prefix codes to individual byte values, usually by traversing the binary tree node by node as each bit is read from the input stream. Reaching a leaf node necessarily terminates the search for that particular byte value.

For example, using the tree shown in Figure 4, the code `'1100001010'` is translated into the word `FACE`.

In the class `Encoding`, implement a method `decodeText` that decodes a binary string `coded` passed in the parameter into a plain text and returns it. In addition,

- The string `coded` must contain only 1s and 0s. The method must raise a `ValueError` if it contains another symbol.
- The method must raise a `ValueError` if the code is invalid. For example, the message has been corrupted and the coded word does not finish on a leaf of the binary tree `_encoding`. Given the tree from Figure 4, the method should raise an exception if the string passed in the parameter is `'0001'`.

(iii) [10 marks] To encode a plain text, using the `_encoding` tree is not convenient. A better approach is to build a dictionary based on the tree, where the key is a symbol from the text to encode, and the mapped value is the path to that symbol in the `_encoding` tree. Using the encoding tree shown in Figure 4, the built dictionary should be equal to `cipher`.

```
cipher = {'A':'000', 'B':'001', 'C':'010',
          'D':'011', 'E':'10', 'F':'11'}
```

Encoding the string `'ACE'` with the built dictionary is straightforward, take the first character from the string, lookup the corresponding path in the `cipher` dictionary and append it to the encoded text. The result of the process is `'00001010'`.

In the class `Encoding`, implement a method `encodeText` that encodes a plain text `text` passed in the parameter into a binary string and returns it. In addition, the method must raise a `ValueError` if the `text` contains symbols not present in the `_encoding` tree. Given the tree from Figure 4, the method should raise an exception if the `text` passed in the parameter is `'CAT'`.

End of examination paper