MT2505 Computing Project 1: Binary Operations

MRQ

Semester 2, 2019–20

Overview

This is the first part of the Computing Project for MT2505. A second part will be made available later in semester when the relevant material has been covered in lectures. You should make a single submission consisting of your attempts for both parts of the Computing Project in a single file. Answers to the numbered **Question**s should all be included in your submission. The additional document "Guidance on what to submit" (`guidance.pdf`) will give you information on the format of your submission.

This document contains some steps labelled "**Check and Debug Your Code**". These give instructions on programming tasks to do that will enable you to check whether the functions you have produced do what you intend. It would not be a bad idea to also check your code on additional examples. A successful solution to **Question 2** will enable you to generate many such additional examples.

Python resources

The project will be completed in Python. For more information about Python see the following sources:

- Notes from the Computing Workshop from Semester 1: `https://mms.st-andrews.ac.uk/mms/module/2019_0/Y1/MT2000-Workshop/Student+Area/Lecture%20Notes/?subview=icon`
- The official Python tutorial: `https://docs.python.org/3/tutorial/`. Chapter 3, and particularly Section 3.1.3 concerning lists, might be useful when completing this part of the project.

If you are reading this document in a suitable programme (e.g., Acrobat Reader), the above are clickable links that will take you to the appropriate page.

1 Binary operations

In this part of the project you will write some Python code to handle binary operations and to check whether they satisfy certain properties introduced in Chapter 1 of the lecture notes.

For the purposes of this project, a binary operation will be represented by its *Cayley table* of the form:

$*$	0	1	2
0	0	1	0
1	0	0	0
2	0	1	1

If A is the set $\{0, 1, 2\}$ with binary operation $*$ defined by the above table, then, for example,

$$1 * 0 = 0 \qquad \text{and} \qquad 0 * 1 = 1.$$

To be more precise, the entry in the row labelled x and the column label y is equal to the value $x * y$.

You can define the binary operation given by the table above in Python by setting:

```
X = [[0, 1, 0], [0, 0, 0], [0, 1, 1]]
```

The table is then stored in the variable `X`. If you then want to find the product of the elements 0 and 2 (that is, the value $0 * 2$), then you can do:

```
X[0][2]
```

We shall only be considering binary operations defined on the set $A = \{0, 1, \ldots, n-1\}$ for some positive integer n. (In particular, binary operations defined on other sets are *not* to be considered in this part of the project.)

Question 1: Write a function in Python called `is_binary_operation` which checks if a given variable `X` (in Python) really represents a binary operation in the above format.

Your function should check whether the input variable is a list with n entries (for some positive integer n) and each entry is itself a list of length n whose entries are integers between 0 and $n-1$ (inclusive).

Your function should have one argument `X` and should return `True` or `False`. It might look something like:

```
def is_binary_operation(X):
  if not isinstance(X, list):
    return False

# your code: goes here!
```

The Python command `isinstance` checks whether the first parameter is an instance of the variable type given in the second parameter. In the above code, `isinstance` is being used to check that the input argument `X` is actually a list.

Check and Debug Your Code: Check that your function `is_binary_operation` returns the correct value with each of the following inputs[1]:

(a) `"banana"`

(b) `[["a", "b", "c"], ["b", "c", "a"], ["c", "b", "a"]]`

(c) `[[0, 1, 1, 3], [3, 2, 1, 1], [3, 3, 0, 3], [0, 3, 0, 3]]`

(d) `[[1, 1, 0], [1, 1], [0, 1, 1]]`

(e) `[[1, 1, 2], [2, 3, 1], [0, 0, 1]]`

(f) `[[1, 1, 2, 1], [2, 3, 1, 3], [0, 0, 1, 0], [0, 0, 1, 0], [1, 2, 3, 0]]`

(g) `[1, 1, 0, 0, 1, 2, 1, 2, 2]`

(h) `[[1, 1, 2],[0, 1, 1, [1, 1, 0]]]`

(i) `[[1, 0, 1], [0, 0, 1], [0, 1, 2]]`

[1]The answers should be: (a) `False`; (b) `False`; (c) `True`; (d) `False`; (e) `False`; (f) `False`; (g) `False`; (h) `False`; (i) `True`. If you obtain different answers, or if your function does not yield an answer, then debug your function!

Question 2: Complete the following function so that it returns a random binary operation on the set $\{0, 1, \ldots, n-1\}$ for any positive integer n:

```
def random_binary_operation(n):
    out = []
    for i in range(n):
        out.append([])
        for j in range(n):
            # your code: append a random integer between 0 and
            # n-1 to the table
    return out
```

[Hint: You might consider using the `random` module. The advanced users of Python among you may choose to create a more sophisticated version of the function `random_binary_operation` which does not use any of the suggested code above.]

Check and Debug Your Code: Check your code does produce what you expect. You can now use your command `random_binary_operation` to check that other functions you write later in this project are working properly.

2 Associativity

An operation $*$ on a set A is *associative* if

$$(x * y) * z = x * (y * z) \qquad \text{for all } x, y \text{ and } z \text{ in } A.$$

If one is asked to determine whether a binary operation $*$ on a set A is associative or not, then one must either find a counterexample to show it is not or provide a proof that it is. A counterexample is just a single instance of $x, y, z \in A$ where $(x * y) * z \neq x * (y * z)$. When A is finite, a proof could just consist of a direct computation that $(x * y) * z = x * (y * z)$ holds for every possible value of $x, y, z \in A$. If $|A| = n$, there are a total of n^3 triples to check. For each triple, we need to calculate four products and so we have a total of $4n^3$ computations to complete. This would be very tiresome by hand, even for very small examples. If $|X| = 3$, as in the example

$*$	0	1	2
0	0	1	0
1	0	0	0
2	0	1	1

then we have to, in the worst case, check 27 equalities and perform 108 calculations of elements!!!

In this part of the project, you will write Python code to check if the binary operation $*$ on the set $\{0, 1, \ldots, n-1\}$, represented by a list of lists as above, is associative or not.

Question 3: Complete the following function `is_associative_operation` so that it returns `True` if the argument `X` defines an associative binary operation and `False` if it does not.

```
def is_associative_operation(X):
    if not is_binary_operation(X):
        return False
    # your code: goes here
```

[Recall that the product `i * j` of `i` and `j` is found using `X[i][j]`.]

Check and Debug Your Code: Which of the following multiplication tables is associative[2]?

[2]The first, third, and fifth tables are associative, the others are not.

$*$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	2	1	0	1
5	0	0	0	2	1	1	2
6	0	0	0	0	1	1	2

$*$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	2	1	0	1
5	0	0	0	2	1	1	2
6	0	0	0	0	1	1	6

$*$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1
2	0	0	2	2	2	2	0
3	0	0	2	3	2	2	0
4	0	0	2	2	4	4	0
5	0	0	2	2	4	5	0
6	0	1	0	0	0	0	6

$*$	0	1	2	3	4
0	0	1	2	3	4
1	1	0	4	2	3
2	2	3	0	4	1
3	3	4	1	0	2
4	4	2	3	1	0

$*$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	4	0	3
2	2	4	3	1	0
3	3	0	1	4	2
4	4	3	0	2	1

$*$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	0

The binary operations defined by the above tables can be loaded into Python in the following way:

- Save the file `sampleops.py` found on MMS into your own folder.
- In Python, use the following code:

```
from sampleops import assoc_example
```

- The six binary operations, stored as lists of lists, are then:

`assoc_example[0]`, `assoc_example[1]`, ..., `assoc_example[5]`

3 Identity

Let A be a set and $*$ be an operation on A. An element e of A is an *identity* for $*$ if

$$e * x = x * e = x \qquad \text{for all } x \in X.$$

Consequently, for $A = \{0, 1, \ldots, n-1\}$ as assumed on this project sheet, $e \in A$ is an identity for $*$ if and only if the row and column in the Cayley table labelled e are equal to $[0, 1, \ldots, n-1]$ in that order.

For example, in

$*$	0	1	2
0	**0**	**1**	**2**
1	**0**	0	0
2	**0**	1	1

the first row is $[0, 1, 2]$ but the first column is not (and so 0 is *not* an identity for this operation), and in

$*$	0	1	2
0	0	**0**	2
1	**0**	**1**	**2**
2	0	**2**	1

the element 1 is an identity.

One can get a row of a binary operation `X` (represented as a list of lists) using `X[i]`. Working with a column requires a little more effort:

```
[row[j] for row in X]
```

(Think carefully about why this works! You may want to check the Python documentation on lists.)

Question 4: Write a function called `identity_of_operation` that takes a binary operation (a list of lists) as an argument and returns $e \in \{0, \ldots, n-1\}$ that is an identity for the operation if it exists, and returns `-1` if there is no identity element.

Check and Debug Your Code: Find the identity of following multiplication tables if they exist[3].

$*$	0	1	2	3	4
0	0	1	2	3	4
1	1	0	4	2	3
2	2	3	0	4	1
3	3	4	1	0	2
4	4	2	3	1	0

$*$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	4	0	3
2	2	4	3	1	0
3	3	0	1	4	2
4	4	3	0	2	1

$*$	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	0	0	0

$*$	0	1	2	3	4	5	6
0	0	0	2	2	4	5	0
1	0	0	2	2	4	5	1
2	2	2	0	0	5	4	2
3	2	2	0	0	5	4	3
4	4	4	5	5	0	2	4
5	5	5	4	4	2	0	5
6	0	1	2	3	4	5	6

$*$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1
2	0	0	2	3	0	2	6
3	0	0	3	2	0	3	6
4	4	4	4	4	4	4	4
5	0	1	2	3	4	5	6
6	0	0	6	6	0	6	6

$*$	0	1	2	3	4	5	6
0	0	1	2	3	4	4	3
1	1	2	4	0	3	3	0
2	2	4	3	1	0	0	1
3	3	0	1	4	2	2	4
4	4	3	0	2	1	1	2
5	4	3	0	2	1	1	2
6	3	0	1	4	2	2	5

The binary operations defined by the above tables are also found in file `sampleops.py`. Use the code

```
from sampleops import id_example
```

to load the six binary operations, stored as lists of lists, as:

`id_example[0]`, `id_example[1]`, ..., `id_example[5]`

4 Group axioms

For the final steps in this part of the project, you will require access to every single possible binary operation on the set $A = \{0, 1, 2\}$ (with three elements). Use the code

```
from sampleops import tripleop
```

to gain access to a function `tripleop` that takes one argument (an integer between 0 and $3^9 - 1 = 19\,682$). The result of `tripleop(n)` is a binary operation, stored as a list of lists, defined upon the set $\{0, 1, 2\}$. As `n` ranges through all permitted values, every single possible binary operation is produced.

One can calculate the number of binary operations on $\{0, 1, 2\}$ that are associative can be computed using the following code:

```
sum (1 for n in range(3**9) if
  is_associative_operation(tripleop(n)))
```

[3] Your function `identity_of_operation` should return: `0`, `0`, `-1`, `6`, `5`, `-1`. If you obtain different answers, then debug your function!

(At least, this will be the case if your command `is_associative_operation` functions correctly.) Evaluate the above code to determine how many associative binary operations there are on this set.

The final question on this part of the computing project requires you to do similar computations. You will need to write suitable functions that checks additional properties. Your submission should include these functions and also the code that you execute to perform this computation. Finally include a piece of commented text, such as

```
# Returns 9876
```

to state your answer. (In this case, the claim would be that there are 9 876 binary operations on $\{1, 2, 3\}$ satisfying some particular collection of conditions.)

The additional definitions you need are:

- A binary operation on a set A is *commutative* if $x * y = y * x$ for all $x, y \in A$.
- A *group* is a set A together with a binary operation that is associative such that there is an identity element and every element has an inverse. (See Chapter 5 of the lecture notes for details.)
- A group is *abelian* if its binary operation is commutative.

Question 5: Write suitable functions to determine the answers to the following questions:

(a) How many of the binary operations on $A = \{0, 1, 2\}$ are associative but not commutative?

(b) How many of the binary operations on $A = \{0, 1, 2\}$ are commutative but not associative?

(c) How many of the binary operations on $A = \{0, 1, 2\}$ endow A with the structure of a group?

(d) How many of the binary operations on $A = \{0, 1, 2\}$ endow A with the structure of an abelian group?