COMP3702/COMP7702 Artificial Intelligence
Semester 2, 2020
Assignment 0

The purposes of this assignment are to refresh some of the foundational mathematical concepts and methods used in AI, and introduce you to some of the basic programming tools and functionalities you need to know for your assignments. **Assignment 0 does not contribute to your final grade**, however you are strongly encouraged to do them for your own learning and prepare you for the remainder of the course. We also aim to make it autogradable through Gradescope, so that you can self-assess your progress; more detail on this will be given later. This assignment is composed of three parts: logic and sets, graphs, and coding basics.

1 Logic and sets

This section will review some key basic terminology, concepts and notation from logic and sets. Logic will be addressed in much greater detail in Module 2 of COMP3702/COMP7702, while sets are used extensively in almost all scientific programming; a solid understanding of how both are used in programming tasks is essential to excelling in this course.

Loosely speaking, **logic** is a formal, scientific method of examining or thinking about ideas. The tools and rules of mathematical logic allow us to assess the validity of statements and assertions and to make inferences. Some basic logic notation follows; given statements A and B:

- $A \wedge B$ for (A and B), is the *conjunction* of A and B,
- $A \vee B$ for (A or B or (A and B)), is the *disjunction* of A and B,
- $A \Rightarrow B$ for (A *implies* B), which means (if A then B), and so is automatically true when A is false,
- $A \Leftrightarrow B$ to mean (A iff B), which abbreviates (A if and only if B), and expresses the logical *equivalence* of A and B, and
- $\neg A$ for (not A), is *negation* of A, and is true iff A is false.

Statements may contain variables (or unknowns, or place-holders), as in $(x \leq 3) \wedge (y \leq 7)$ which is true when the integer variables x and y have values less than or equal to 3 and 7 respectively, and false otherwise. A statement like $P(x, y)$, which involves variables x, y, is called a *property* or *predicate* (or *relation*, or *condition*) and it only becomes true or false when the pair x, y stand for particular things.

A **set** is an (unordered) collection of objects, called its *elements* or *members*. We write $a \in X$ when a is an element of the set X. We read $a \in X$ as "a is a member of X" or "a is an element of X" or "a belongs to X", or just "a in X". Sometimes we write $\{a, b, c, \ldots\}$ for the set of elements $a, b, c, \ldots$, using curly braces. Some important sets:

- $\emptyset$: the *empty set* with no elements.
- $\mathbb{N}$, the set of *natural* numbers $\{1, 2, 3, \ldots\}$, and $\mathbb{N}_0$, the *natural* numbers with zero $\{0, 1, 2, 3, \ldots\}$
- $\mathbb{Z}$, the set of *integers*, both positive and negative, including zero, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$,
- $\mathbb{Q}$, the set of *rational* numbers, given by ratios of integers, and
- $\mathbb{R}$, the set of *real* numbers.

A set X is said to be a *subset* of a set Y, or X is *included* in Y, written $X \subseteq Y$, iff every element of X is an element of Y, that is: $X \subseteq Y \leftrightarrow (z \in X) \wedge (z \in Y)$. A set is determined solely by its elements (not their order) in the sense that two sets are equal iff they have the same elements.

Now assume a set U (for "universe") exists. Subsets of U support operations closely related to those of logic. The key operations are:

- Union $A \cup B = \{x : x \in A \vee x \in B\}$
- Intersection $A \cap B = \{x : x \in A \wedge x \in B\}$ and
- Complement $A^c = \{x \in U : x \notin A\}$.

Question 1.1.

(a) Write a (python) script that completes the set operations of union and intersection on an arbitrary pair of set inputs.

(b) Two sets A and B are said to be *disjoint* when $A \cap B = \emptyset$. Write a script that tests if an arbitrary pair of set inputs are disjoint, and prints either "DISJOINT" or "INTERSECTING" depending on the outcome.

Set generation: We take the existence of the empty set $\emptyset$ for granted, along with the sets of basic elements listed above. Beyond this, sometimes a set is determined by a logical *property*, in that the set's members are precisely those which satisfy the property. Then we write $X = \{x : P(x)\}$, meaning the set X has as elements all the x such that the property $P(x)$ is true (i.e. read ":" as "such that" or "which satisfy"). Alternatively, if Y is a set and $P(y)$ is a property, we can form the set $\{y \in Y : P(y)\}$, or equivalently $\{y : y \in Y \wedge P(y)\}$. This is the subset of X consisting of all elements y of Y that satisfy $P(y)$. When we write $\{a_1, \ldots, a_n\}$, it can be interpreted as the set $\{x : x = a_1 \vee \ldots \vee x = a_n\}$. These rules are for generating sets are called *comprehensions*.

Question 1.2. Write a python script that:

(a) initialises the universe of values $U = \{u \in \mathbb{N}_0 : u < 10\}$, then selects three elements of U (selection, not removal) at random and calls them set A, and selects another four elements of U at random and calls them set B,

(b) computes the *set difference* $A \setminus B$, defined as $\{a \in A \wedge a \notin B\}$; call the output C,

(c) finds the complement of B and computes $A \cap B^c$; call the output D,

(d) Checks the set equivalence of the outputs of steps (b) and (c), i.e. check if $C = D$.

The following questions walk you through some useful comprehensions for generating larger sets from smaller ones.

Question 1.3. Begin with sets $U = \{u \in \mathbb{N}_0 : u < 5\}$ and $V = \{a, b, c, d, e\}$, and write scripts to generate the following sets:

(a) A set of ordered pairs, or 2-tuples, can be be constructed from two sets X and Y by computing their *set product*: $X \times Y = \{(x, y) : x \in X \wedge y \in Y\}$. Generate $U \times V$. Comment on its size.

(b) A set consisting of the set of all subsets of a set is called a *powerset*: $\mathcal{P}(X) = \{Y : Y \subseteq X\}$. Generate the powerset of V, $\mathcal{P}(V)$. Comment on its size.

2 Graphs

We will make heavy use of graphs and trees in COMP3702/COMP7702. In particular, a *directed graph*, or *digraph*, $G = (V, E)$, comprises vertices, V, that are connected together by edges, E, in which all the edges are directed from one vertex to another.

Expressing a digraph in code is a necessary building block to many AI methods. An efficient way to do this in python is to use *tuples*, which are ordered and immutable. Typically, you can represent an edge as a 2-tuple of vertices, or a 3-tuple with two vertices followed by an edge attribute *dictionary*, e.g.:

```
e1 = (1, 3)
e2 = (2, 3, {'weight': 3.1415})
```

In the first line, the edge is directed from vertex 1 into 3; that is to say, the order of the vertices in this tuple matters. In the second line, the edge is directed from vertex 2 into 3, and an attribute of edge *weight* is set to 3.1415. Any extra information you may want to include about the edge can be stored in such a dictionary, including physical characteristics of the edge or formatting information for visualisations. Edges expressed in this format can then be collected in any iterable container, such as a list or a set, to collate the set E, for example:

```
E = {e1, e2}
```

or all in one go:

```
E = {(1, 3),
     (2, 3, {'weight': 3.1415}) }
```

Question 2.1. Write a method defining a *successor function* for the digraph representation above. The method should take the digraph edge representation specified above and a specific vertex as inputs, and return a set containing the vertices that can be reached from the given vertex. For example, for the E given above, `successor(E,1)` should return $\{3\}$, while `successor(E,3)` should return $\emptyset$.

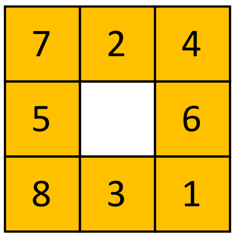
3 Coding basics

Question 3.1. Write a program that takes the filename of a text file as a single command-line argument, i.e. `cat ⟨file⟩` and outputs the content of that file to standard output.

Question 3.2. Write a program that takes a text file as input, removes a particular letter from the content of the text file, and outputs the results into another text file. The program should take 3 inputs as a single command-line argument. These inputs (in order) are:

- The filename of the input text file.
- The filename of the output file.
- The letter to be removed.

Question 3.3. The state of the 8 puzzle (as described in the supplementary Week 2 lecture) can be represented as a 9-character ASCII strings of digits 1-8 and '_' for the blank, in top-to-bottom left-to-right order. For example, 8-puzzle state below would be given by the string `7245_6831`:



Write a program that takes, as input, two states of the 8-puzzle and outputs how to reach the second state from the first by only taking a single legal action (i.e. `L`, `R`, `U` or `D`), or if not possible, outputs `IMPOSSIBLE`. In particular: The program should run from command line as:

```
which_action ⟨state1⟩ ⟨state2⟩
```

where `⟨state1⟩` and `⟨state2⟩` are 8-puzzle states. The program should output a single line to standard output, one of `L`, `R`, `U` or `D` if it's possible to take a single action to reach one state from the other, or `IMPOSSIBLE` if it isn't possible. For example, the result of running:

```
which_action 12345678_ 132_46578
```

should be `IMPOSSIBLE`, whereas the result of running:

```
which_action 12345678_ 12345_786
```

should be `U`.