LDD2-Project: Session 5

Aims of this session: To start with Boolean circuits.

We'll start by creating a new class, bool_circ, which will be a subclass of open_digraph, since Boolean circuits are special cases of open directed graphs.

Exercise 1:
Create the bool_circ subclass of open_digraph. Ensure that if g is an instance of open_digraph, then bool_circ(g) creates a Boolean circuit using g. For the moment, we don’’t need to worry about whether the graph g is a valid Boolean circuit. The point of this exercise is simply to take an open directed graph and cast it as a Boolean circuit.

In the following, we will define the logical gate at each node by means of its label. We suggest using & for the AND gate, I for the OR gate, and ~ for the NOT gate; and leaving the empty label ‘ ’ for the copy symbol. In the future, we can also have nodes labelled '0' and '1' to represent the constants 0 and 1, or ^ to represent EXCLUSIVE OR.

To be a valid Boolean circuit, all the “copy” nodes must have exactly one input (i.e. must have an input degree of 1). Since the AND and OR gates are associative (for example, in the case of &, we have $(x_0\&x_1)\&x_2 = x_0\&(x_1\&x_2)$), they can have more than 2 inputs without being ambiguous. We also accept that they can have 0 entries (this represents the neutral element of the operation) or 1 entry (this is the identity).

Exercise 2:
For node, implement the indegree, outdegree, and degree methods, which calculate the incoming, outgoing, and total degree of a node, respectively.

To be a valid Boolean circuit, the graph must also be acyclic.

Exercise 3:
Create an is_cyclic method, which tests the cyclicity of a directed graph (for example using the algorithm we saw in the "session 0" slides).

Exercise 4 (tests required):
Implement an is_well_formed method for bool_circ, which tests whether the object created is a well-defined Boolean circuit. Remember that to have a valid Boolean circuit, the corresponding graph should be acyclic and respect the constraints on the degrees given above.

Exercise 5:
Modify the init method of bool_circ so that it tests whether the given graph is indeed a Boolean circuit.

In what follows, when we are given a pair of graphs, we will need to ensure that there is no overlap between the indices of one and those of the other. For example, if we have i as the id of a node in the first graph, we would like i not to be the id of any node in the second graph. A simple solution is to translate the indices of one of the two graphs sufficiently so that there is no overlap. The following two exercises are to be performed in the open_digraph class.

Exercise 6:
Implement the min_id and max_id methods, which return the min index and max index of the graph nodes, respectively.

Exercise 7 (tests required):
Implement a shift_indices(self,n) method, which adds an integer n (possibly negative) to all the indices in the graph.