Breadth-first search can be a useful algorithm, particularly when the step costs are all equal or you care about the path length and not the path cost. This activity is in two parts:

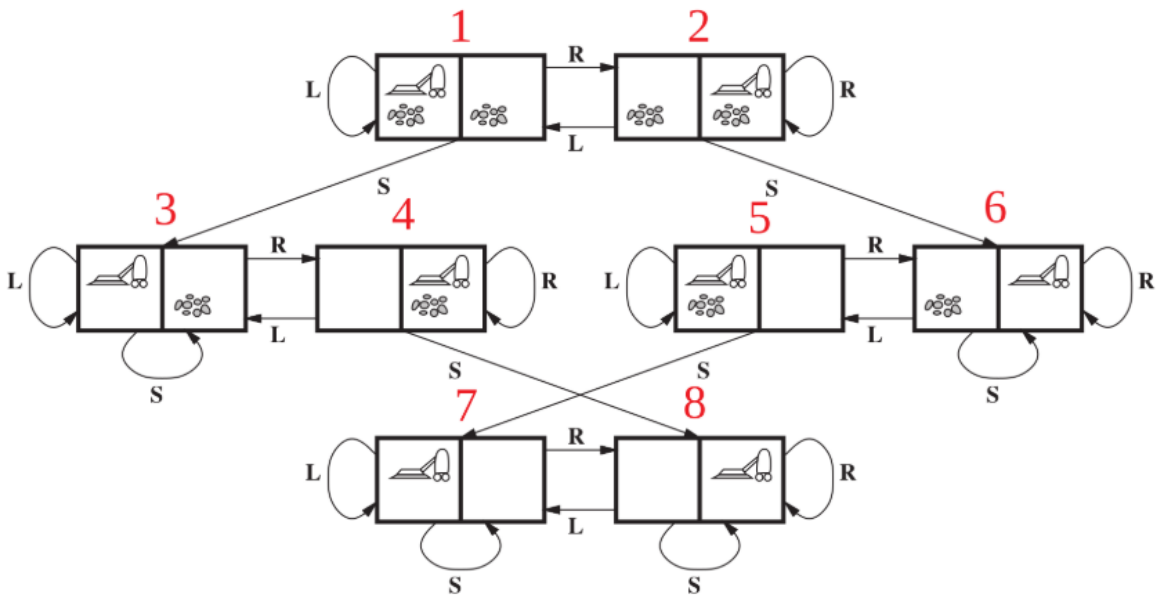
1. Implement breadth-first search yourself for a small example problem;
2. Share and discuss your implementation with other students in a small group, then improve your own implementation (if necessary) using the best ideas from the group.

This activity will help to reinforce your knowledge of uninformed search strategies. It will also help you with the first practical assessment, in which you will need to implement another algorithm that is closely related to breadth-first search.

Implement breadth-first search (as described in AIMA Chapter 3 Section 3.4.1) in Java to solve the small problem described below. The implementation does not need to be a generic breadth-first search, it can be specific to this problem. In particular, a state may be represented with a single integer, and the expansion of a node can be implemented with a simple look-up table, indexed by an integer *s* and containing a list of integers (representing the states adjacent to *s*).

Your implementation of the frontier does not need to be the most efficient choice. You do not need to extract the solution at the end: just verifying that you have reached a goal state is enough. Finally, there is no need to store the names of the actions. It should be possible to implement your program with just one Java class.

The problem scenario is the **vacuum world** that you saw in lesson 1. The diagram of all 8 states is shown below. Each state has a number in red. You will need to represent the states and transitions shown in the diagram in your program in some form. For example, state 1 expands to states 1, 2 and 3 (by the actions Left, Right and Suck).



State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dirt Left	True	True	False	False	True	True	False	False
Dirt Right	True	True	True	True	False	False	False	False
Vacuum Position	Left	Right	Left	Right	Left	Right	Left	Right
Action – Left	1	1	3	3	5	5	7	7
Action – Right	2	2	4	4	6	6	8	8
Action – Suck	3	6	3	8	7	6	7	8