1. ***Punched-tape machine.*** *Punched-tape machine* is a Turing machine equipped with an infinite two-way single-row tape, where each cell can be either *punched* or *unpunched*. Furthermore, once a cell is punched, it can never be unpunched. Alternatively, a punched-tape machine is just a Turing machine with unary alphabet consisting of a single symbol ■ (together with the empty cell symbol □), where any symbol ■ written on the tape can never be erased.

   Prove that the punched-tape machine is equivalent in power to the standard Turing machine. In other words, the languages accepted by the punched-tape machines are exactly all the computable languages.

2. ***Self-referential machines.***

   (a) Design an algorithm B that, assuming that its own source code $\langle B \rangle$ is conveniently given as input when starts, outputs the *source code* of another algorithm A that prints the source code of B on execution. (The algorithm should be described using pseudocode, not an explicit Turing machine.)

   (b) Design an algorithm that outputs its own source code without any input. (Remember that your algorithm does not have access to its own source code. The executable of the algorithm alone has to generate its source code back.)[1]

   [Hint: As a warm-up, can you construct a Turing machine M that takes no input, and output a sequence of 1s on the tape, where the number of 1s is equal to two times the number of transitions (edges) in M?]

[1] Algorithm with such property is called a ***quine***. Wait, you never wrote a quine before? Put the homework on pause and write a quine in your favorite programming language. It might sound like a silly trick, but in fact, this is how compilers can be written in the same language they are trying to compile.