**Turing Machines**

| *s* | *r* | *w* | *d* | *t* |
|---|---|---|---|---|
| 1 | 0 | 1 | R | 2 |
| 1 | 1 | 0 | R | 2 |
| 2 | 0 | 1 | L | 1 |
| 2 | 1 | 1 | L | HALT |

*figure 1*

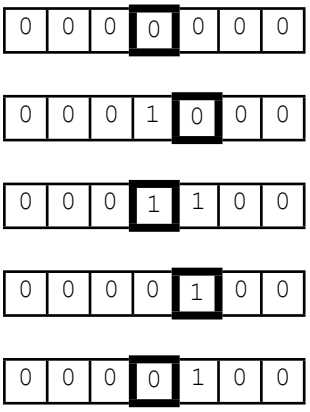


*figure 2*

A *Turing Machine* is a simple type of programmable computer. It consists of a *tape* which can be read from and written to, and a set of instructions which tell it how to behave. The tape is an infinitely long line of *cells*, each of which can store a `0` or a `1`. The machine is always positioned at a cell on the tape and is in one of several numbered *states*.

All instructions are of the form "if we are in state *s* and there is an *r* in the current cell, write *w* in the current cell, then move one cell in direction *d* (left or right) on the tape and finally change into state *t* or `HALT`". Every state has two rules, one for what to do if there is a `0` in the current cell and another if there is a `1`. There is a special `HALT` condition, after which the machine does no more work.

An example set of instructions is shown in figure 1 and the corresponding way in which those instructions affect the tape is shown in figure 2 (the highlighted box indicates the current position on the tape). In this latter figure, the machine is started in state 1 on a tape containing all `0`s and on the bottom line the tape has halted.

We could represent a positive integer *n* on the tape by *n* adjacent `1`s.

Question 1

Produce a Turing machine (i.e. write down the instruction table) that, if it is given a tape that is all `0`s except for a section representing *n*, with the machine's initial position on the left-most `1`, makes the tape represent *n+1* and then halts.

Question 2

Produce a Turing machine that, if it is given a tape that is all `0`s except for a section representing *x* followed by a single `0` and then a section representing *y*, with the machine's initial position on the left-most `1`, makes the tape represent *x+y* and then halts. Explain briefly in English the algorithm you have produced.

Turing machines are an important theoretical concept, because a sufficiently complicated machine can be built to perform any algorithm. We can consider, for a fixed number of states, how many different machines can be built.

Question 3

How many different machines can be built with a single state?

**Train Tracks**

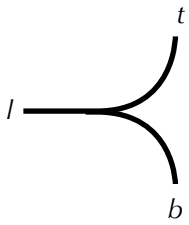


figure 3

Consider a railway that consists of lengths of track that are connected together by simple *points*, such as that shown in figure 3. A train entering a point along one of the curved pieces of track must exit along the straight portion; i.e. a train entering from *t* or *b* will exit via *l*. A train entering via *l* will exit via *t* or *b* depending on the point.

We will start by considering two types of points:

On a *lazy point*, entering from a curved portion sets the point so that, until the point is set again, a train entering from the straight portion will exit on the set curved portion. E.g. If a train enters the track from *t* (hence leaving via *l*) whenever it then enters the point from *l* it will exit from *t*, until the point's setting is altered by the train entering from *b*. Initially, when building the railway, we can choose how the point is set.

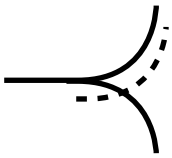

figure 4

A *sprung point* is built so that a train entering from the straight portion will always exit along the same curved portion. When drawing a sprung point we will add an additional dotted line to indicate how it is configured. Figure 4 shows an example where a train entering from the straight portion will always follow the upwards curve.

There are bridges available to the railway builder so that, when necessary, portions of the track can cross over themselves; i.e. if necessary we can draw crossing tracks. In these cases, a train cannot jump between different tracks.

Question 4

The lazy and sprung points split (or combine, depending on direction) a single track into two. Why is it not necessary to have points that split a single track into more than two tracks?

A simple train, that can only move forward, moves over the track.

Question 5

The train can only move forward but it is possible, thanks to the points, that it might get turned around and hence run along a piece of track in both directions. Draw a simple section of the railway that turns around a train.

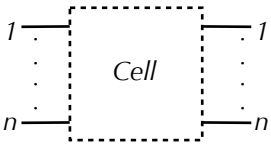


figure 5

It is possible to build a Turing machine using tracks, points and a simple train. We will build this by making, and connecting together, cells (see figure 5) which not only represent a `0` or `1` but also contain the instruction set logic. Each cell on the tape will contain the same configuration of tracks, with some of its internal points indicating the current value stored in the cell.

The position of our machine on the tape will be the cell that currently contains the train. When the train enters or leaves the cell it will leave on a track that represents the current state; so a cell in an *n* state machine will have *n* tracks on both the left-side and on the right-side.

The train will be able to enter or leave a cell from either direction, since we may have come from either the left or right cell, and instructions can be to move left or right. Within our cells we will want a more regimented 'flow' for the train (so that we can use the same tracks whether we have arrived from the left or from the right). We will do this by building a block inside the cell (we'll call it *B)* that has *n* inputs (one for each input state), and *2n* outputs (one for each output state going to the left and one for each state going to the right). The railway will be designed so that the train can only go along input and output tracks in a single direction.

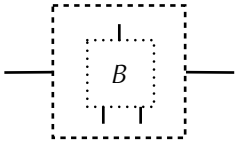


*figure 6*

Question 6

Figure 6 shows the outline of a cell for a 1 state machine with *B* inside; the top line from *B* is its input, the bottom left line is the output when traveling to the left and the bottom right line is for output to the right. Illustrate how appropriate tracks and points can be added to join *B* to the rest of the cell.

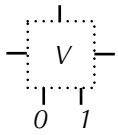


*figure 7*

One configuration that is going to be useful is shown in figure 7; we'll call it *V* since it is related to the value of the cell. We will design our railway so that trains only approach from either the top or the left. When a train enters from the top, if the cell is currently set to `0` it will exit from the bottom track labelled *0*; similarly if the cell is set to `1`. If the train enters from the left side, we will toggle the value of the cell, and exit from the right side.

To build *V* we will introduce a new item, a *flip-flop* point, which we can denote (as in figure 8) by adding a short bar through the curved tracks. For this point, every time a train enters along the straight portion of the track it alternates which curved track it follows. Initially, when building the railway, we can choose how the point is set. It is not possible to build a flip-flop point using a finite combination of lazy and sprung points.

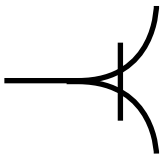

*figure 8*

Question 7

Using all three types of points, illustrate how to construct *V*.

Question 8

Consider a component of the railway that behaves in a similar manner to *V* but, rather than a single left and right track for toggling the cell's state, has two left and two right tracks. Entering on the first left track would set the cell to `0` before leaving on the first right track. Entering on the second left track would set the cell to `1` before leaving on the second right track. Is it possible to build such a component without using a flip-flop point? Justify your answer.

Since several instructions may require us to toggle the set value of a cell, it will be useful to create a *subroutine* to perform this function. As with a conventional subroutine, when its work is finished we will want to be able to return to where we originated.

Question 9

Suppose that we have a subroutine which is entered and exited by the same piece of track, and that this subroutine can be approached from several different tracks. Illustrate how those pieces of track can be connected to the subroutine track so that, when the train exits the subroutine it will return along the correct track.

Question 10

Is your solution to question 9 robust; i.e. is it possible for the points to become set in such a way that the entry / exit of the subroutine will not work as planned? Justify your answer.

We will build *B* by having each input track (recall there is one per state) lead in to its own *V*. The value written on the cell is represented by all the *V*s, so we need to ensure that each of the *V*s in the same cell represents the same value.

Question 11

Outline how to put together a subroutine to toggle the *V*s.

The instruction set for the Turing machine is defined by what happens after the tracks labelled *0* and *1* in figure 7.

Question 12

For each possible form of instruction, illustrate how the track should be constructed after a *V*.

Congratulations! You've built a Turing machine from a railway.

In constructing our railway we have allowed the use of bridges so that tracks can cross over each other. Bridges are not actually necessary; we can construct a network of points that has the same effect.

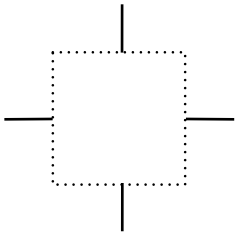

*figure 9*

Question 13

Construct a roundabout for the tracks shown in figure 9, so that a train entering on any of the tracks exits on next track clockwise.

Question 14

By using a roundabout as an inner core (or otherwise) construct an equivalent to a bridge, that has no crossing tracks, so that a train entering on any of the tracks exists on the opposite track.