Q= {q0, q1 …. q12}

T= {1, ., =, X, Y}

B= ‘B’

Q0 = q0

F = q12

State table =

State: q0 Edges: (q0,1,1,r) (q1,.,.,r)

State: q1 Edges: (q1,1,1,r) (q2,B,.,l)

State: q2 Edges: (q2,1,1,l) (q3,.,.,r)

State: q3 Edges: (q3,X,X,r) (q4,1,X,l) (q12,.,=,l)

State: q4 Edges: (q4,X,X,l) (q5,.,.,l)

State: q5 Edges: (q5,Y,Y,l) (q6,1,Y,r) (q11,B,B,r)

State: q6 Edges: (q6,Y,Y,r) (q7,.,.,r)

State: q7 Edges: (q7,1,1,r) (q7,X,X,r) (q8,.,.,r)

State: q8 Edges: (q8,1,1,r) (q9,B,1,l)

State: q9 Edges: (q9,1,1,l) (q10,.,.,l)

State: q10 Edges: (q10,1,1,l) (q10,X,X,l) (q5,.,.,l)

State: q11 Edges: (q11,Y,1,r) (q3,.,.,r)

State: q12 Edges: (q12,X,1,l)

1. See above for the logic. I believe I used a relatively small number of character, the only way I could use less symbols is by removing the = and replacing it with a ’.’ (this would also remove a step). I chose this method because it seemed the easiest to implement and understand from a conceptual level.

3. Keep going right until you reach a blank

Turn the blank into a .

Go left until you reach a c

Skip all Xs

Convert a one on the right side into and X

Skip all X,Y and Z, moving left

Convert a one on the left side into and Y

Skip all X,.,Y and ones, moving right to the end of the tape

Convert a blank into a one

Move left until you get back to the left side

Go back to state 5

When at state 5 skip blanks moving right

Convert all Ys into ones moving right

When you reach the ., move to the right and go back to state 3

When the multiplication is done, convert . into an = and all the Xs into ones

It works perfectly with multiply by zero. I believe the maximum numbers before the stack gets too large is around 10\*10 – this is likely due to the queue based implementation. I wanted it to more closely resemble a physical tape/head instead of using a simple, more memory efficient array.

1. See turingMachine.js and README.MD. I JavaScript because:

* I am most familiar with it.
* It is an object oriented language, making the creation of the graphs/the implantation of the components easier
* It has meant I can create a nice user interface for it easily: see index.html or https://oliverbrotchie.github.io/FoundationsTwoCoursework/

5/6. See test.txt

7. In terms of time complexity the program is relatively fast, it does however take more memory to store than if I had implemented it using an array and it also does not clear up junk on the tape behind the output blank.

8. Not Implemented