Turing Machine - Factoring

Héctor Briongos Merino

January 2023

Description of the task and design decisions

The task is going to be suited for a Turing machine with only one infinite tape whose alphabet can contain only the symbols {0, 1, _} and the head on the Turing machine moves in left (L) / right (R) direction or doesn't move (*), upon reading a symbol on the tape. The selection of a Turing machine with only one tape is to give an example that every task can be preformed using a one tape Turing machine, even when using more tapes (that can act as memory storage or to obtain auxiliary results) could make the implementation easier and also help to complete the task faster.

First of all, the task chosen for the implementation is factoring. The algorithm that I have implemented in the Turing machine to preform the factoring task to a given number written in binary is depicted in Fig. 1.

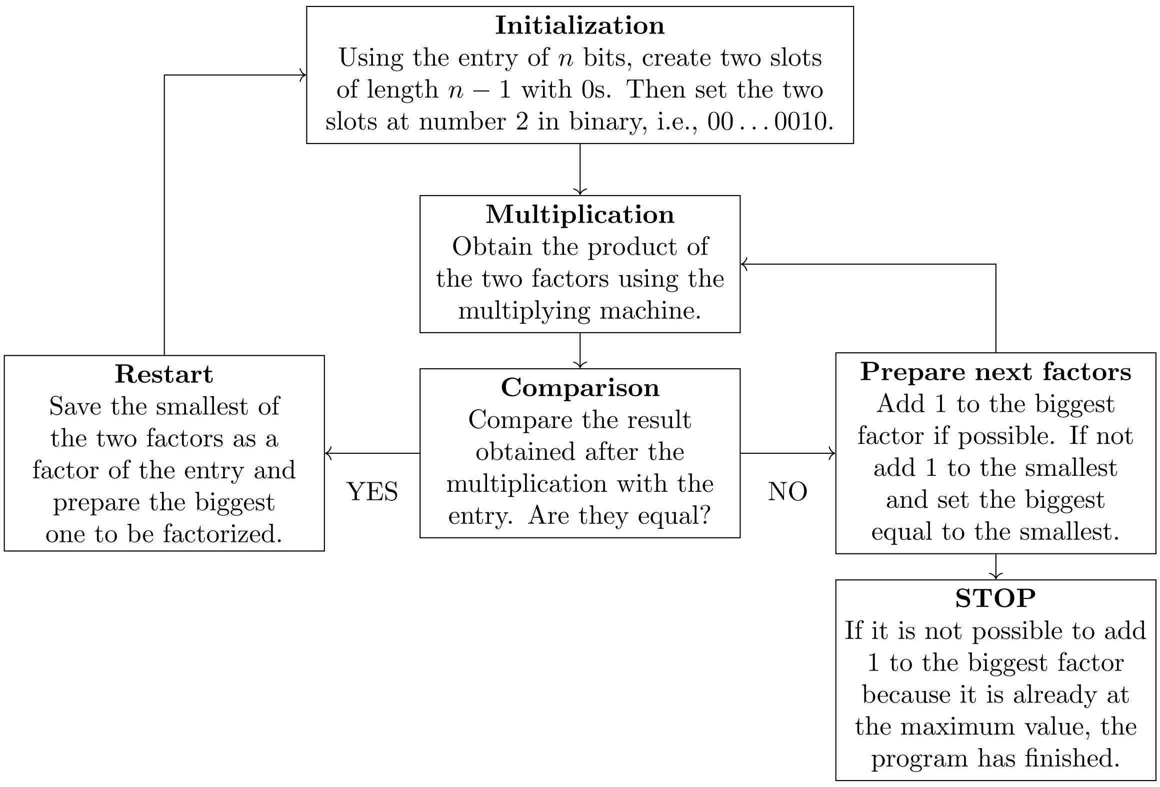
Initialization
Using the entry of n bits, create two slots of length $n-1$ with 0s. Then set the two slots at number 2 in binary, i.e., $00\ldots0010$.
Multiplication
Obtain the product of the two factors using the multiplying machine.
Restart
Save the smallest of the two factors as a factor of the entry and prepare the biggest one to be factorized.
Comparison
Compare the result obtained after the multiplication with the entry. Are they equal?
Prepare next factors
Add 1 to the biggest factor if possible. If not add 1 to the smallest and set the biggest equal to the smallest.
YES
NO
STOP
If it is not possible to add 1 to the biggest factor because it is already at the maximum value, the program has finished.

Figure 1: Algorithm of factorization used for the homework

As the Turing machine don't use any special symbols to help us to mark special parts on the tape (for example, the starting point of the initial input or where the factors we are multiplying are), I decided to use different strings of blank symbols to note different parts on the tape.

Explicit description of TM

Now, the different steps of the algorithm should implemented using instructions for the Turing machine of the form: $\delta(p, X) = (Y, D, q)$ where p is the state where the Turing machine is, X is the symbol that the head is reading, Y is the symbol that the machine is going to write, D the direction where the head is going to move after writing and q the state of the Turing machine after moving the head.

As input for the machine, one should provide the number that want to factorize in binary. If there are zeros at the left of the input they will be erased as they don't provide any value to the number. After erasing the meaningless zeros, the input will be of length n. To describe the steps of the algorithm, the instructions for each task will be given and commented to help the understanding of the method.

Note: the name of the states

An explanation about the name of the different states implemented should be done before writing all of them. The creative task of thinking all the transitions was made using a notebook and a pen, and ideally the states started in 0 and grew up to the number of states needed. To distinguish between the steps of the different steps of the algorithm, build up in different days, a letter before the number was added to Multiplication ("M"), to Comparison ("C") o Restart ("R"). Also, to distinguish between parts of the same steps (for example, if we are copying a "0" or a "1") I introduced a $+n \cdot 10^2$ factor in the numbering of the states, so they share two digits that help to identify them as similar sections but they are not the same states.

As the creation task was done using a notebook, then the instructions were copied into a .txt file to test the machine using the simulator provided. Sometimes, the instructions weren't right, with the most usually issue been the lack of a node to skip an extra blank. To fix those problems I introduced new states, and to do so without changing all the numbers of the states in the file I denoted them with the number of the state right before and adding a "b" after.

The other different label that can be found in the name of the states is a "P". This one was introduced after the algorithm was complete and working, with the aim of reducing the steps needed to finish the task. This optimization of the algorithm will be explained in the Complexity analisys section.

Initialization

During the initialization, the machine writes a string of zeros 5 blank to the left of the input number, the length of the string of zeros will be of $n-1$. After that, other identical string of zeros is created 2 blank to the left of the first one. Finally, a "1" is written in the second position of the strings to denote a number 2 in binary on each one. We are ready to start multiplying.

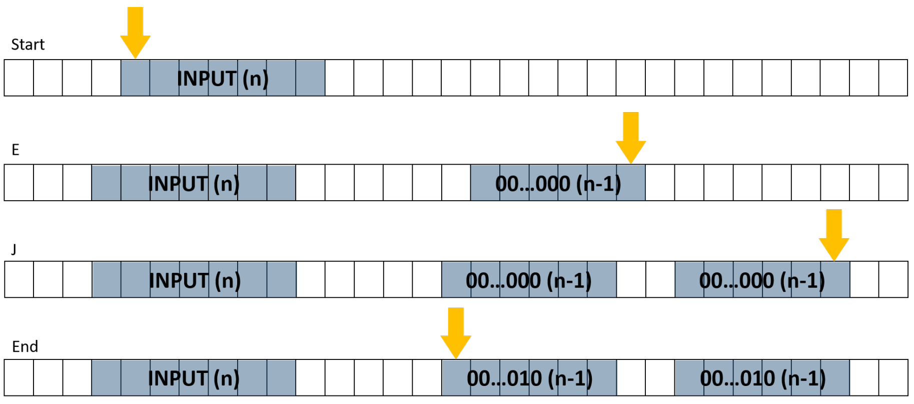

Figure 2: Different tape situations during initialization

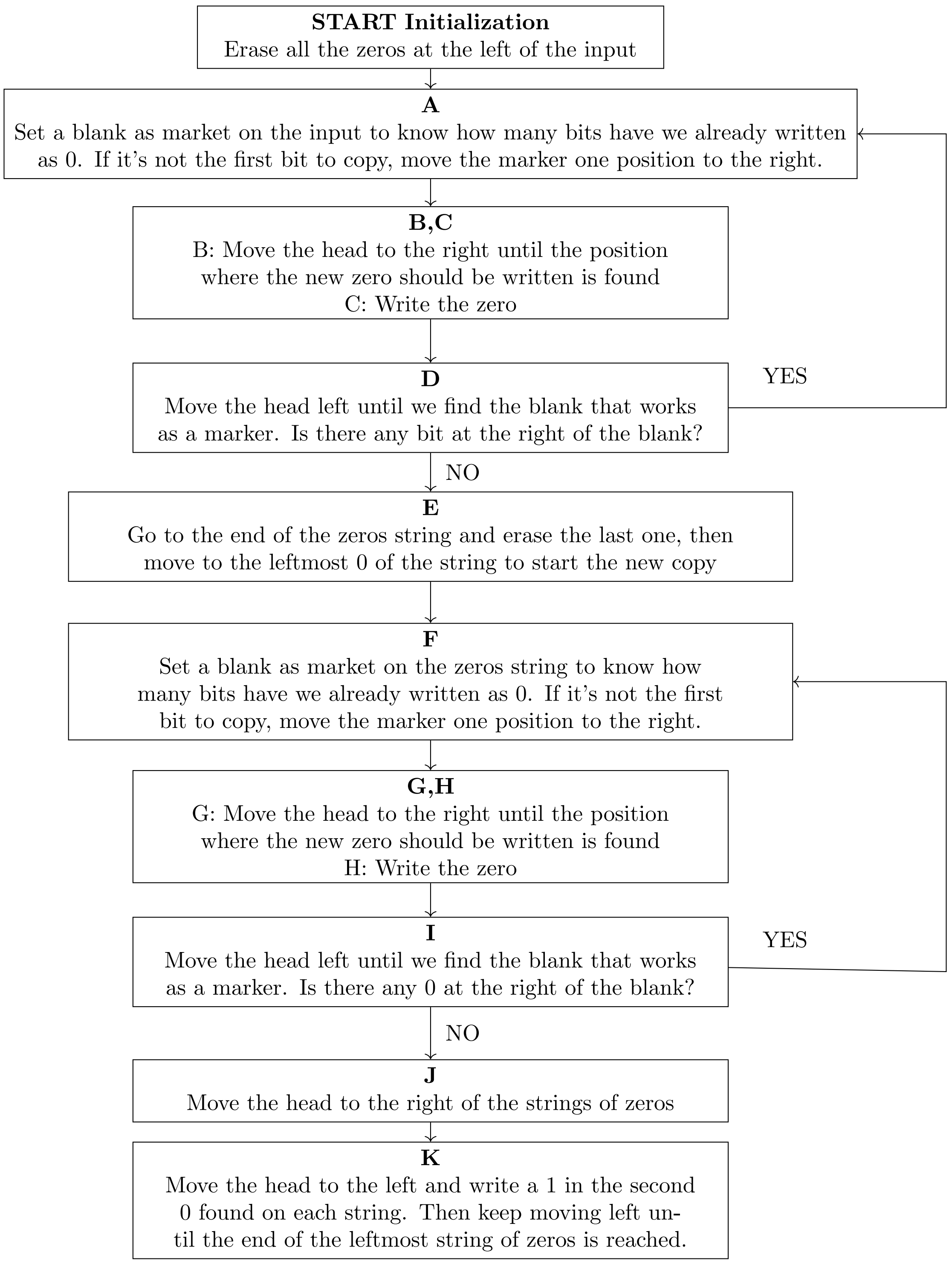
START Initialization
Erase all the zeros at the left of the input
A
Set a blank as market on the input to know how many bits have we already written as 0. If it's not the first bit to copy, move the marker one position to the right.
B,C
B: Move the head to the right until the position where the new zero should be written is found
C: Write the zero
D
Move the head left until we find the blank that works as a marker. Is there any bit at the right of the blank?
YES
NO
E
Go to the end of the zeros string and erase the last one, then move to the leftmost 0 of the string to start the new copy
F
Set a blank as market on the zeros string to know how many bits have we already written as 0. If it's not the first bit to copy, move the marker one position to the right.
G,H
G: Move the head to the right until the position where the new zero should be written is found
H: Write the zero
I
Move the head left until we find the blank that works as a marker. Is there any 0 at the right of the blank?
YES
NO
J
Move the head to the right of the strings of zeros
K
Move the head to the left and write a 1 in the second 0 found on each string. Then keep moving left until the end of the leftmost string of zeros is reached.

Figure 3: TM algorithm for the Initialization

The specific decision table for the Initialization step, commented:

```
0 0 _ r 0  #Erase all 0 at the left of the input
0 1 1 * 0i #Find the leftmost 1 of the input
0i 1 _ l 1  #A start: Set a blank space as a marker to count the length
0i 0 _ l 2
```

```
1 _ 1 r 3
2 _ 0 r 3
3 _ _ r 4  #A end
4 0 0 r 4  #B start: Go +5 blank right after the input
4 1 1 r 4
4 _ _ r 5
5 _ _ r 5b
5b _ _ r 6
6 _ _ r 7
7 _ _ r 8  #B end
8 0 0 r 8  #C start: Find the last written 0 and write other afterwards
8 1 1 r 8
8 _ 0 l 9  #C end
9 0 0 l 9  #D start: Move left until the marker on the input is found
9 1 1 l 9
9 _ _ l 10
10 _ _ l 11
11 _ _ l 11b
11b _ _ l 12
12 _ _ l 13
13 0 0 l 13
13 1 1 l 13
13 _ _ r 0i #D end
0i _ _ r 14  #E start: Go to the end of the string of zeros, erase the last one
14 _ _ r 14
14 0 0 r P0
P0 0 0 r P0
P0 _ _ l P1
P1 0 _ l P2  #E end
P2 0 0 l P2  #Moving to the left of the first string to start the second copy
P2 _ _ r P3
P3 0 _ l 15  #F start: Set a blank space as a marker to count the lenght
15 _ 0 r 150  #F end
150 _ _ r 16  #G start: Go +1 blank right after the string of zeros
16 0 0 r 16
16 _ _ r 17  #G end
17 _ 0 l 18  #H start: Find the last written 0 and write other afterwards
17 0 0 r 17  #H end
18 0 0 l 18  #I start: Move left until the marker in the 0s string is found
18 _ _ l 19
19 0 0 l 19
19 _ _ r 20
20 0 0 * P3  #I end
20 _ _ r 21  #J start: Go to the right of the two strings of 0s
21 0 0 r 21
21 _ _ l 22  #J end
22 0 0 l 23  #K start: Write a 1 in the correct position to have 2 in the factors
23 0 1 l 24
24 0 0 l 24
24 _ _ l 25
25 _ _ l 25
25 0 0 l 26
26 0 1 l 27  #K end
27 0 0 l 27  #Move to the leftmost 0 of the left factor to start multiplying
27 _ _ r M0
```

Multiplication

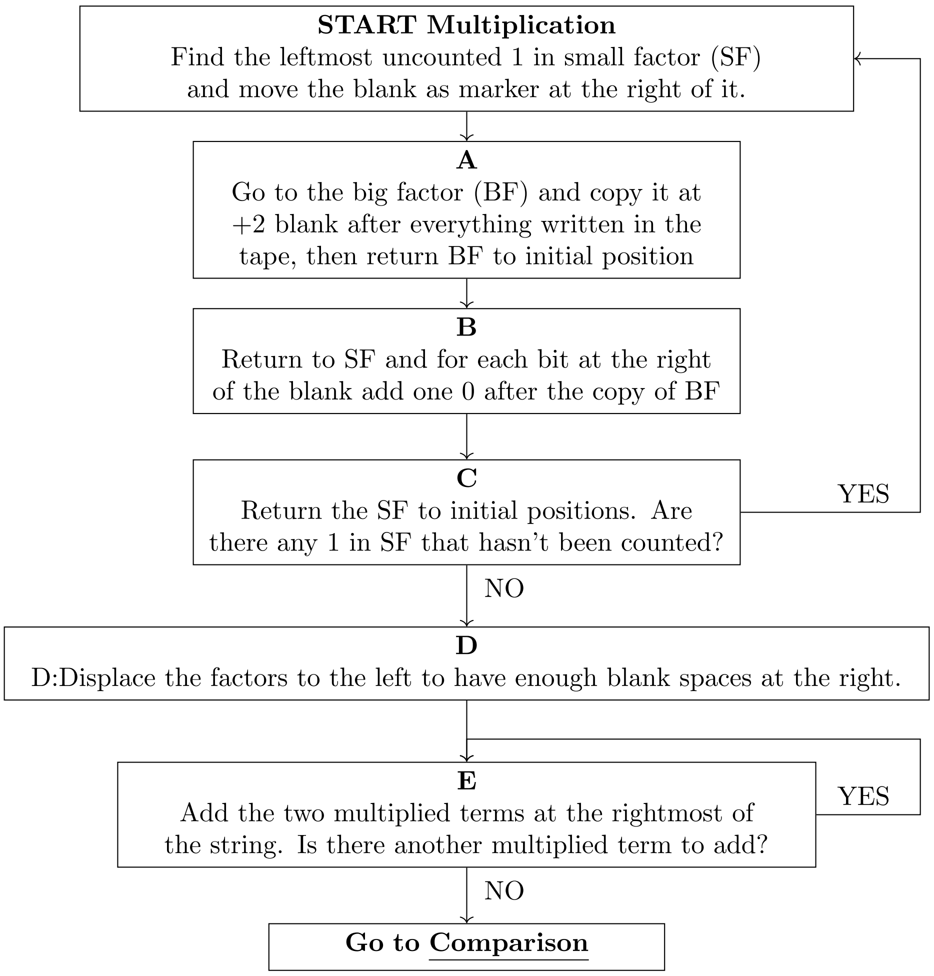
START Multiplication
Find the leftmost uncounted 1 in small factor (SF) and move the blank as marker at the right of it.
A
Go to the big factor (BF) and copy it at +2 blank after everything written in the tape, then return BF to initial position
B
Return to SF and for each bit at the right of the blank add one 0 after the copy of BF
C
Return the SF to initial positions. Are there any 1 in SF that hasn't been counted?
YES
NO
D
D:Displace the factors to the left to have enough blank spaces at the right.
E
Add the two multiplied terms at the rightmost of the string. Is there another multiplied term to add?
YES
NO
Go to Comparison

Figure 4: TM algorithm for multiplying

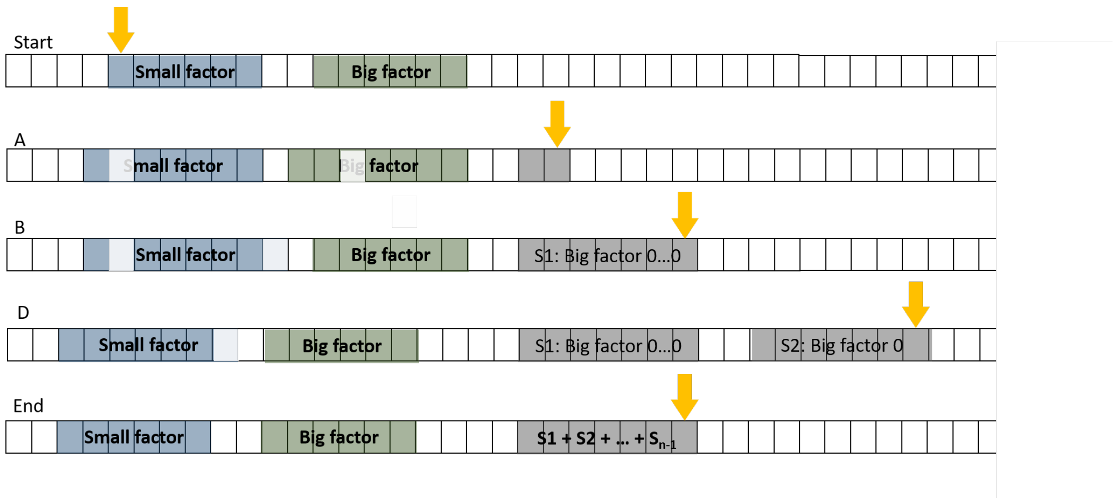

Figure 5: Different tape situations during multiplication

```
M0 0 _ l M201  # START Start: Find the leftmost 1 after blank and move before
M201 _ 0 r M202
M202 _ _ r M0
M0 1 _ l M1
M1 _ 1 r M2
M2 _ _ r M3  # START End
M3 0 0 r M3  # A Start: Go to BF and copy it at +2 blank
M3 1 1 r M3
M3 _ _ r M4
M4 _ _ r M5
M5 1 _ l M6  !! states inside B to the first bit of BF if it is a 1
M6 _ 1 r M7
M7 _ _ r M8
M8 0 0 r M8
M8 1 1 r M8
M8 _ _ r M9
M9 _ _ r M10
M10 0 0 r M8
M10 1 1 r M8
M10 _ 1 l M11  !! end of states to copy 1
M5 0 _ l M106  !! states inside B to the first bit of BF if it is a 0
M106 _ 0 r M107
M107 _ _ r M108
M108 0 0 r M108
M108 1 1 r M108
M108 _ _ r M109
M109 _ _ r M110
M110 0 0 r M108
M110 1 1 r M108
M110 _ 0 l M11  !! end of states to copy 0
M11 _ _ l M12  !! intermediate states to go back to BF
M12 _ _ l M213
M213 0 0 * M13
M213 1 1 * M13
M213 _ _ l M30
M13 0 0 l M13
M13 1 1 l M13
M13 _ _ l M14
M14 0 0 r M16
M14 1 1 r M16
M14 _ _ l M15
M15 0 0 l M13
M15 1 1 l M13
M15 _ _ l M30
M16 _ _ r M17  !! end of movement of traslation to the left
M17 1 _ l M18  !! states inside B to copy 1, not been the first bit copied
M18 _ 1 r M19
M19 _ _ r M20
M20 0 0 r M21
M20 1 1 r M21
M20 _ _ r M24
M21 0 0 r M21
M21 1 1 r M21
M21 _ _ r M22
M22 _ _ r M23
```

```
M23 0 0 r M21
M23 1 1 r M21
M22 0 0 r M21
M22 1 1 r M21
M23 _ _ l M25
M24 _ _ r M22
M25 _ _ l M26
M26 _ 1 l M27  !! end of states to copy 1
M17 0 _ l M118  !! states inside B to copy 0, not been the first bit copied
M118 _ 0 r M119
M119 _ _ r M120
M120 0 0 r M121
M120 1 1 r M121
M120 _ _ r M124
M121 0 0 r M121
M121 1 1 r M121
M121 _ _ r M122
M122 _ _ r M123
M123 0 0 r M121
M123 1 1 r M121
M122 0 0 r M121
M122 1 1 r M121
M123 _ _ l M125
M124 _ _ r M122
M125 _ _ l M126
M126 _ 0 l M27  !! end of states to copy 0
M27 0 0 l M27
M27 1 1 l M27
M27 _ _ l M12
M30 0 _ r M31  !! states to move BF to initial position
M31 _ 0 l M33
M30 1 _ r M32
M32 _ 1 l M33
M33 _ _ l M34
M34 0 _ r M31
M34 1 _ r M32  !! end of the movement of BF
M34 _ _ l M35  # A end
M35 0 0 l M35  # B start: add 0s after the copy of BF
M35 1 1 l M35
M35 _ _ l M36
M36 0 0 l M35
M36 1 1 l M35
M36 _ _ r M40
M40 _ _ r M41
M41 1 _ l M42  !! start making a second blank as marker
M42 _ 1 r M43
M43 _ _ r M41
M41 0 _ l M44
M44 _ 0 r M43  !! end of making a second blank
M41 _ _ r M45
M45 1 _ l M46  !! moving one bit to the left of the marker
M46 _ 1 r M48
M45 0 _ l M47
M47 _ 0 r M48
M48 _ _ r M49  !! end of moving bits to the left of the marker
```

```
M49 0 0 r M49  !! states to go to the very end (right)
M49 1 1 r M49
M49 _ _ r M50
M50 _ _ r M51
M51 0 0 r M49
M51 1 1 r M49
M51 _ _ l M52
M52 _ _ l M53  !! end of going to the right, we are at the end
M53 _ 0 l M54  !! write a 0
M54 0 0 l M54  !! states to go back (left) until find one blank alone
M54 1 1 l M54
M54 _ _ l M55
M55 _ _ l M58
M55 0 0 r M56
M55 1 1 r M56
M56 _ _ r M45
M58 0 0 l M54
M58 1 1 l M54  !! end of states to move left and repeat the loop
M58 _ _ l M59  # B end
M59 0 _ r M60  # C start: return SF to initial position
M60 _ 0 l M62
M59 1 _ r M61
M61 _ 1 l M62
M62 _ _ l M59
M59 _ _ l M63
M63 0 0 * M59
M63 1 1 * M59
M63 _ _ r M64
M64 _ _ r M65
M65 _ _ r M66
M66 0 0 r M66
M66 1 1 r M66
M66 _ _ r M0  # C end
M0 _ _ l M69  ## There are no more bits to multiply
M69 _ _ l M70  # D start: displace factors to the left
M70 0 0 l M70
M70 1 1 l M70  !! reaching the end of SF and moving 1 left only if finished with a 0
M70 _ _ r M71
M71 0 _ l M72
M72 _ 0 r M74
M71 1 _ l M73
M73 _ 1 r M74
M74 _ _ r M71  !! end of moving SF
M45 _ _ r M145  !! patch: to move BF if last digit was a 1
M145 _ _ r M145
M145 0 0 * M75
M145 1 1 * M75 !! end of patch
M71 _ _ r M75  !! going to BF and moving 2 left
M75 _ _ r M75
M75 1 _ l M76
M76 _ _ l M77
M77 _ 1 r M80
M75 0 _ l M78
M78 _ _ l M79
M79 _ 0 r M80
```

```
M81 _ _ r M82
M82 0 0 * M75
M82 1 1 * M75
M82 _ _ r M301  # D end
M301 _ _ r M302 # E Start: add the multiplied terms
M302 0 0 r M302  !! start moving to the right end of the written tape
M302 1 1 r M302
M302 _ _ r M303
M303 _ _ r M304
M304 0 0 r M302
M304 1 1 r M302
M304 _ _ l M305
M305 _ _ l M306
M306 _ _ l M307  !! head is at the righmost written bit
M307 0 0 l M307  !! checking if there are only 1 or 2 bits between rightmost number
M307 1 1 l M307     and the neighbour (we can sum). Otherwise we are done
M307 _ _ l M308
M308 _ _ l M309
M308 0 0 r M311
M308 1 1 r M311
M309 0 0 r M310
M309 1 1 r M310  !! end of checking we can sum
M309 _ _ l M326  !! end of checking we are done with the adding
M310 _ _ r M311  !! moving to the rightmost bit again
M311 _ _ r M312
M312 0 0 r M312
M312 1 1 r M312
M312 _ _ l M313
M313 0 0 l M313  !! subtract 1 to the righmost number
M313 1 0 r M314
M314 0 1 r M314  !! end of subtrac
M314 _ _ l M315  !! moving to the neighbour number
M315 0 0 l M315
M315 1 1 l M315
M315 _ _ l M316
M316 _ _ l M317
M316 0 0 * M317
M316 1 1 * M317  !! end of going to the left neighbour
M317 0 1 r M318  !! add 1 to
M317 1 0 l M317
M317 _ 1 r M318  !! end of add 1
M318 0 0 r M318  !! return to the rightmost bit
M318 1 1 r M318
M318 _ _ r M319
M319 _ _ r M320
M319 0 0 r M320
M319 1 1 r M320
M320 0 0 r M320
M320 1 1 r M320
M320 _ _ l M313  !! head is at the righmost bit start again
M313 _ _ r M324  !! there is no more to add in this number
M324 0 _ r M324  !! deleting the 0s string
M324 _ _ l M325  !! going to the new rightmost bit
M325 _ _ l M325
M325 0 0 l M307  !! start adding again
```

```
M325 1 1 l M307
M326 _ _ r M327
M327 _ _ r M327  !! if there are 4 blank between the result and BF, displace
M327 0 _ l M328     the result one to the left to make easier the comparison process
M328 _ 0 r M330
M327 1 _ l M329
M329 _ 1 r M330
M330 _ _ r M331
M331 0 0 * M327  !! no need to move
M331 1 1 * M327
M326 0 0 r M332
M326 1 1 r M332
M332 _ _ r M332  !! going to the rightmost bit to finish there
M332 0 0 r M333
M332 1 1 r M333
M333 0 0 r M333
M333 1 1 r M333
M331 _ _ l M334
M333 _ _ l C0    # E end
M334 _ _ l C0    # E end
```

Comparison

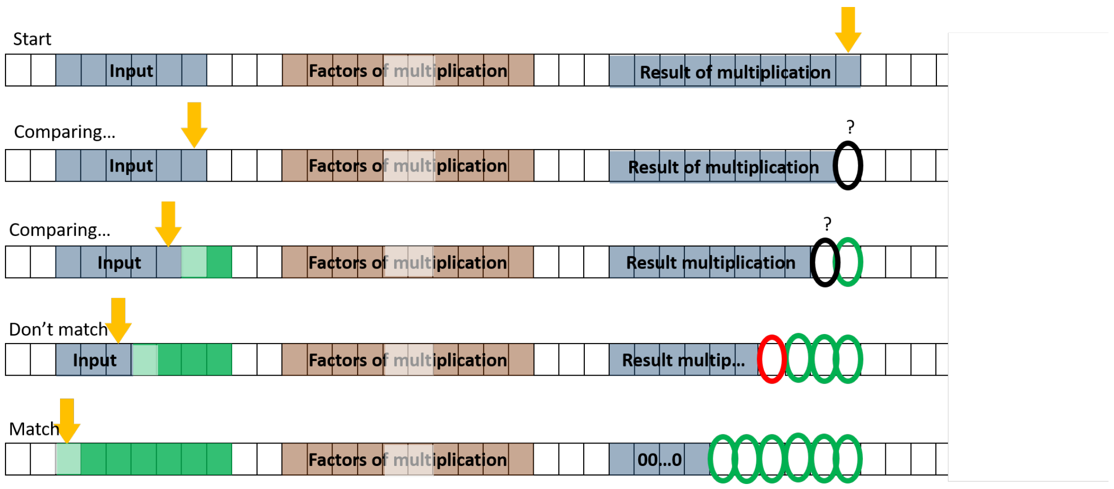

Figure 6: Different tape situations during comparison

The comparison task must be done carefully because there are different possibilities in which the value of the numbers can match in value but the strings differ in the number of bits written

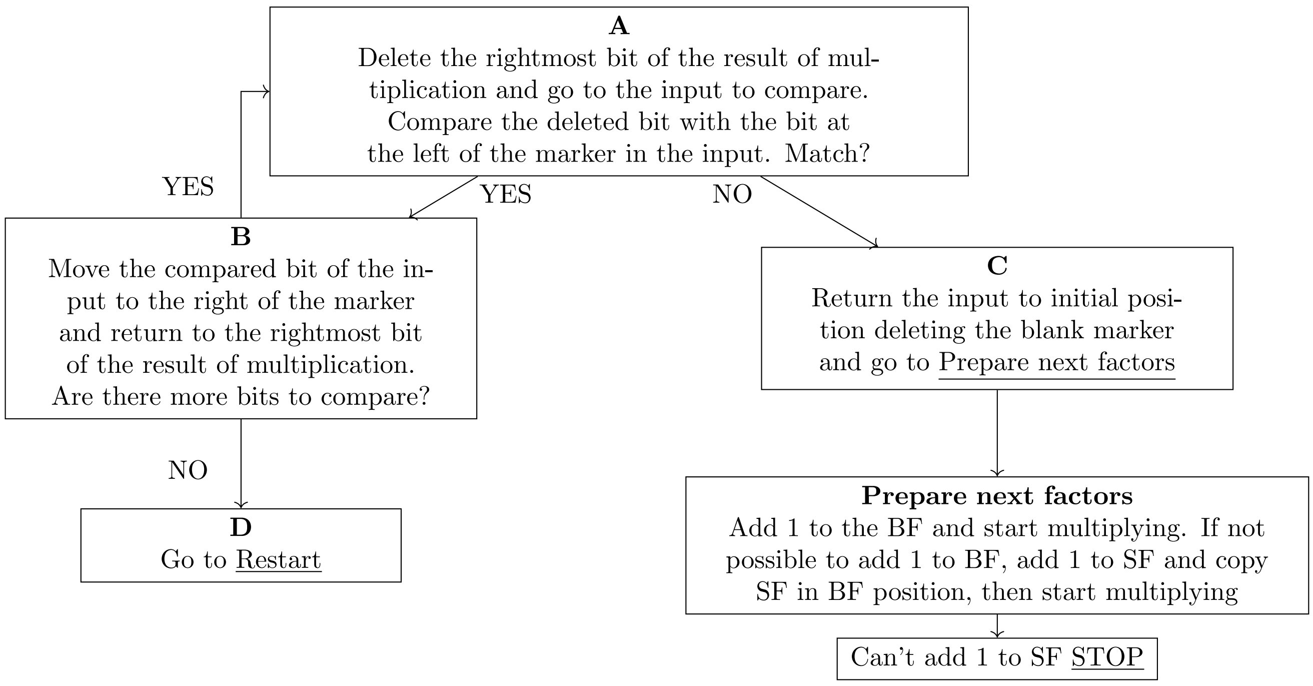
A
Delete the rightmost bit of the result of multiplication and go to the input to compare. Compare the deleted bit with the bit at the left of the marker in the input. Match?
YES
YES
NO
B
Move the compared bit of the input to the right of the marker and return to the rightmost bit of the result of multiplication. Are there more bits to compare?
C
Return the input to initial position deleting the blank marker and go to Prepare next factors
NO
D
Go to Restart
Prepare next factors
Add 1 to the BF and start multiplying. If not possible to add 1 to BF, add 1 to SF and copy SF in BF position, then start multiplying
Can't add 1 to SF STOP

Figure 7: TM algorithm for Comparison + Prepare next factors

```
C0 0 _ l C1  # Start A (0 selected): delete the rightmost bit and go to input
C1 0 0 l C1
C1 1 1 l C1
C1 _ _ l C2
C2 _ _ l C2
C2 0 0 l C3
C2 1 1 l C3
C3 0 0 l C3
C3 1 1 l C3
C3 _ _ l C4
C4 _ _ l C5
C5 0 0 l C5
C5 1 1 l C5
C5 _ _ l C5b
C5b _ _ l C6
C6 _ _ l C7
C6 0 0 l C6
C6 1 1 l C6
C7 0 _ r C8  # end A (0)
C8 _ 0 r C9  # B for (0)
C0 1 _ l C101  # Start A (1 selected)
C101 0 0 l C101
C101 1 1 l C101
C101 _ _ l C102
C102 _ _ l C102
C102 0 0 l C103
C102 1 1 l C103
C103 0 0 l C103
C103 1 1 l C103
C103 _ _ l C104
C104 _ _ l C105
```

```
C105 0 0 l C105
C105 1 1 l C105
C105 _ _ l C105b
C105b _ _ l C106
C106 _ _ l C107
C106 0 0 l C106
C106 1 1 l C106
C107 1 _ r C108  # End A (1)
C108 _ 1 r C9  # B for (1)
C9 0 0 r C9  # Start B: returning to the result of multiplication
C9 1 1 r C9
C9 _ _ r C9b
C9b _ _ r C10
C10 0 0 r C10
C10 1 1 r C10
C10 _ _ r C11
C11 _ _ r C12
C12 0 0 r C13
C12 1 1 r C13
C13 0 0 r C13
C13 1 1 r C13
C13 _ _ r C13b
C13b _ _ r C13c
C13c _ _ r C13d
C13d 0 0 r C13e
C13d 1 1 r C13e
C13e 0 0 r C13e
C13e 1 1 r C13e
C13e _ _ l C0  # End B: going back to A
C13d _ _ l C14  # B (check a): there are no more bits in result to compare
C14 _ _ l C14
C14 0 0 l C15
C14 1 1 l C15
C15 0 0 l C15
C15 1 1 l C15
C15 _ _ l C16
C16 _ _ l C17
C17 0 0 l C17
C17 1 1 l C17
C17 _ _ l C18
C18 _ _ l C19
C19 0 0 l C19
C19 1 1 l C19
C19 _ _ l C20
C20 0 0 l C20
C20 _ _ r R0  # End B (check a): the numbers match
C7 _ _ r C29  # B (check b): there are no more bits in input to compare
C29 _ _ r C30
C30 0 0 r C30
C30 1 1 r C30
C30 _ _ r C31
C31 _ _ r C32
C32 0 0 r C32
C32 1 1 r C32
C32 _ _ r C33
```

```
C33 _ _ r C34
C34 0 0 r C34
C34 1 1 r C34
C34 _ _ r C35
C35 _ _ r C35b
C35b _ _ r C36
C36 0 _ r C36
C36 _ _ l R36  # End B (check b): the numbers match. Go to Restart
C36 1 _ r C36b # End B (check b): the numbers don't match. Start C
C36b 0 _ r C36b  Deleting result of multiplication
C36b 1 _ r C36b
C36b _ _ l C37
C37 _ _ l C37
C37 0 0 * C52
C37 1 1 * C52  !! End of Deleting result
C7 1 1 r C40   !! More options the numbers don't match
C107 0 0 r C40
C107 _ _ r C40
C20 1 1 r C38
C38 0 0 r C38
C39 1 1 r C38
C38 _ _ * C40
C40 _ _ r C41
C41 0 _ l C42  !! Removing the blank marker of the input
C42 _ 0 r C44
C41 1 _ l C43
C43 _ 1 r C44
C44 _ _ r C41  !! Moving to end to delete product and move factors to initial position
C41 _ _ r C45
C45 _ _ r C45
C45 0 0 r C46
C45 1 1 r C46
C46 0 0 r C46
C46 1 1 r C46
C46 _ _ r C47
C47 _ _ r C48
C48 0 0 r C48
C48 1 1 r C48
C48 _ _ r C49
C49 _ _ r C49b
C49b _ _ r C50
C50 0 _ r C50  !! deleting result of multiplication
C50 1 _ r C50  !! deleting result of multiplication
C50 _ _ l C51
C51 _ _ l C51
C51 0 0 * C52
C51 1 1 * C52
C52 0 _ r C53  !! moving BF
C53 _ _ r C54
C54 _ 0 l C57
C52 1 _ r C55
C55 _ _ r C56
C56 _ 1 l C57
C57 _ _ l C58
C58 _ _ l C52
```

```
C52 _ _ l C60
C60 _ _ l C62
C62 0 _ r C63  !! moving SF
C63 _ _ r C64
C64 _ 0 l C67
C62 1 _ r C65
C65 _ _ r C66
C66 _ 1 l C67
C67 _ _ l C68
C68 _ _ l C62
C62 _ _ r C70
C70 _ _ r C70  # End C
C70 0 0 r C71
C70 1 1 r C71
C71 0 0 r C71
C71 1 1 r C71
C71 _ _ r C72
C72 _ _ r C73
C73 0 0 r C73
C73 1 1 r C73
C73 _ _ l C74
C74 0 1 l C75  # Start Prepare next Factors, add 1 to BF
C74 1 0 l C74
C75 0 0 l C75
C75 1 1 l C75
C75 _ _ l C76
C76 _ _ l C77
C77 0 0 l C77
C77 1 1 l C77
C77 _ _ r M0
C74 _ _ r P10  !! can't add 1 to BF, add 1 to SF
P10 0 _ r P10  !! deleting BF
P10 _ _ l P11
P11 _ _ l P11
P11 0 1 l P12  !! add 1 to SF
P11 1 0 l P11b
P11b 0 1 l P12
P11b 1 0 l P11b
P11b _ _ * H  !! can't add 1 to SF, end of program
P12 0 0 l P12
P12 1 1 l P12
P12 _ _ r P13
P13 0 _ l P14  !! marker in SF to create a new BF
P14 _ 0 r P15
P15 _ _ r P16
P16 0 0 r P16
P16 1 1 r P16
P16 _ _ r P17
P17 _ _ r P18
P18 0 0 r P18
P18 1 1 r P18
P18 _ 0 l P19
P19 0 0 l P19
P19 1 1 l P19
P19 _ _ l P20
```

```
P20 _ _ l P21
P21 0 0 l P21
P21 1 1 l P21
P21 _ _ r P13
P13 1 _ l P24
P24 _ 1 r P25
P25 _ _ r P26
P26 0 0 r P26
P26 1 1 r P26
P26 _ _ r P27
P27 _ _ r P28
P28 0 0 r P28
P28 1 1 r P28
P28 _ 1 l P19  !! end of copying SF to BF
P13 _ _ l P30b
P30b _ _ l P30
P30 0 _ r P31  !! moving SF to initial positon
P31 _ 0 l P33
P30 1 _ r P32
P32 _ 1 l P33
P33 _ _ l P30
P30 _ _ r P34
P34 _ _ r M0  # End Prepare next Factors, and start multiplying
```

Restart

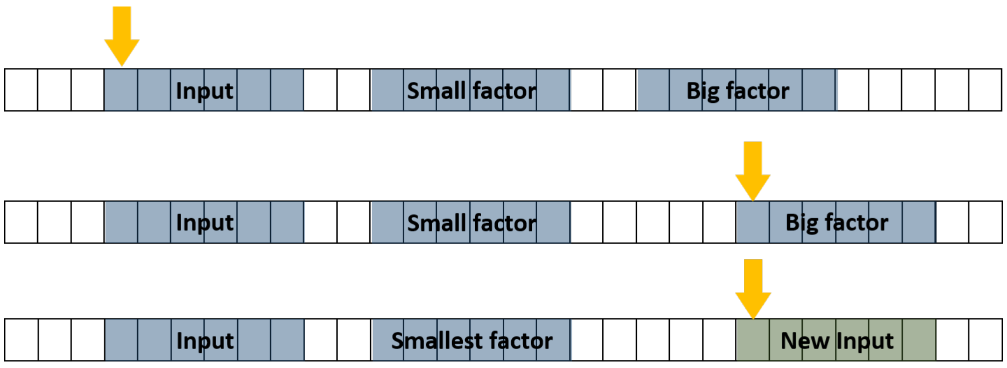

Figure 8: Different tape situations during Restart

The Restart module just moves the big factor to a +5 blank from the small factor (which is the smallest prime factor of the input). Then the factoring algorithm is again implemented using the previous Big factor as input.

```
R0 _ _ r R1 !! Going to the righmost bit of BF
R1 0 0 r R1
R1 1 1 r R1
R1 _ _ r R2
R2 _ _ r R3
R3 0 0 r R3
R3 1 1 r R3
R3 _ _ r R4
R4 _ _ r R5
R5 0 0 r R5
```

```
R5 1 1 r R5
R5 _ _ l R6
C36 _ _ l R36
R36 _ _ l R36
R36 0 0 * R6
R36 1 1 * R6
R6 0 _ r R7  !! Start copy of bits, displacing +3 blanks
R7 _ _ r R8
R8 _ _ r R9
R9 _ 0 l R10
R6 1 _ r R11
R11 _ _ r R12
R12 _ _ r R13
R13 _ 1 l R10
R10 _ _ l R14
R14 _ _ l R15
R15 _ _ l R6
R6 _ _ r R16
R16 _ _ r R16
R16 0 0 * 0  !! end of displacing, start factoring again Initialization
R16 1 1 * 0i
```

Final output

Due to the method applied to find the factors of the input and the way of saving the smallest one and restarting the task with the biggest one, the final state of the tape, where the factors are written, has a very special scheme.

- The input is a prime number of n bits:

  Output will be:

  InputNumber(n)$\_\_\_\_\_\underbrace{00000}_{n-1}$

- The input of n bits is not prime, InputNumber $= A^aB^bC^c$ with A, B, C prime numbers and $A < B < C$:

  Output will be:

$$\text{InputNumber } A^aB^bC^c\_\_\underbrace{A\_\_\_\_\_A^{a-1}B^bC^c\_\_A\_\_\_\_\_A^{a-2}B^bC^c\_\_A\ldots\_\_A}_{a \text{ times}}\_\_$$
$$\_\_\_B^bC^c\_\_\underbrace{B\_\_\_\_\_B^{b-1}C^c\_\_B\_\_\ldots\_\_B}_{b \text{ times}}\_\_\_\_\_C^c\_\_C\_\_\ldots\_\_C\_\_\_\_\_\underbrace{00000}_{(\text{lenth of } C)-1}$$

  Where the machine what is really doing is giving pairs of numbers with 2 blank between them, where the one on the left is the input and the one on the right is the smallest prime factor of the other. As it is run recursively the input for each run it's the quotient between the previous entry and the smallest factor found. To obtain the factors for the input, one just need to pick the strings that are after 2 blank spaces and ignore the ones that are after 5.

  I decided not to clean the output, giving only the factors, because with this output in pairs one can use the multiplying machine with a pair of numbers to obtain a bigger factor and reconstruct the much more easy task of checking if the factorization is well done.

Complexity analysis

As this machine is too complex to make a complexity analysis of the whole machine, the analysis will focused in different subroutines of the machine and in the efficiency of the factoring algorithm.

- Initialization routine:

  The initialization routine just creates two strings of zeros according to the length of the input n. To do the task, the machine copies one bit of the input and moves to a empty space in the tape, making a total displacement of $n+6$ positions, then writes the 0 and returns to the input to pick another number, doing making another displacement of $n+6$. As this task is done n times, to create the first string of zeros the machine needs $\sim 2n(n+6)$ steps. To create the second string, reducing both strings in one unit of the length, the machine needs $\sim 2(n-1)(n+5)$ steps. After that, the routine just moves the head to the rightmost bit of the first string, $\sim 2(n-1)$ steps

  So, combining all the steps needed $\sim 2n(n+6)+2(n-1)(n+5)+2(n-1) = O(n^2)$

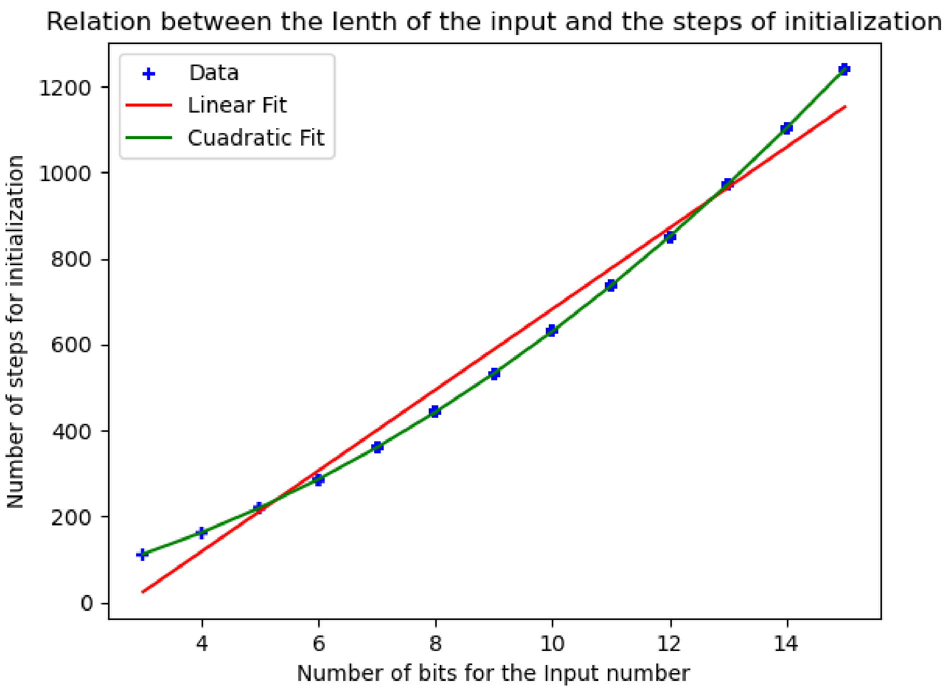

Figure 9: Steps needed to run Initialization as a function of the input

  In figure 9 are plotted the steps needed to run initialization for all possible inputs under 16 bits. It's easy to see that the evolution of the cost is not linear but can be perfectly fitted to a quadratic function as expected.

  NOTE 1 OPTIMIZATION PATCH: the optimization patch introduced in the initialization process was to reduce the length of the string on the factors form n to $n-1$, reducing with that a factor of 4 the number of possible factors checked before saying one number is prime.

- Multiplication

  The multiplication involves two factors of length $n-1$, and the worst case multiplication is that in which both factors are strings of ones. In that case, for each bit of the first factor the second factors is copied at the left of the previous copied factors and then the associated number of zeros is added.

To copy a factor of $n-1$ bits, the head covers $2(n-1)$ times the distance between the initial factor and the position where the number should be written. This distance grows with the numbers of factors and, for the factor number m is $(n-1)+2+\sum_{l=1}^{m-1}(n-1+2+(n-2-m))$. Also, this distance is covered another $2(n-2-m)$ times to add the zeros at the end of the copies.

With that, even before we start adding all the factors, the steps are:

$$\begin{aligned}&\sim \sum_{m=1}^{n-1}\left(2(n-1)+2(n-2-m)\right)\left((n-1)+2+\sum_{l=1}^{m-1}(n-1+2+(n-2-m))\right)\\&=\frac{1}{6}\left(11n^4-48n^3+103n^2-138n+72\right)=O(n^4)\end{aligned}$$

Now, adding all the terms is what makes this type of multiplication waste a lot of time. The smallest factor has a decimal value of $2^{n-1}-1$, which makes the head of the TM cover the distance between the two smallest factors (distance $= n-1+2$) an amount of $2\times 2^{n-1}$ times. As the adding keep going, the second smallest factor has a decimal value of $2\times(2^{n-1}-1)$, and the distance to cover is $n+2$. Following the adding process until the biggest factor, the steps needed (even without taking in account that when adding sometimes the head moves further that we are assuming) are:

$$\begin{aligned}&\sim \sum_{m=1}^{n-2}(n-1+2+m-1)\left(2^{n-1}-1+\sum_{l=1}^{m-1}2^l(2^{n-1}-1)\right)\\&=\frac{1}{4}\left(2^n-2\right)\left(-3n^2+3n+2^n(2n-3)+2\right)=O(n4^n)\end{aligned}$$

This multiplication method is very inefficient, but all the methods find using the same restrictions (one tape and alphabet of 0 and 1) are of the same order of complexity.

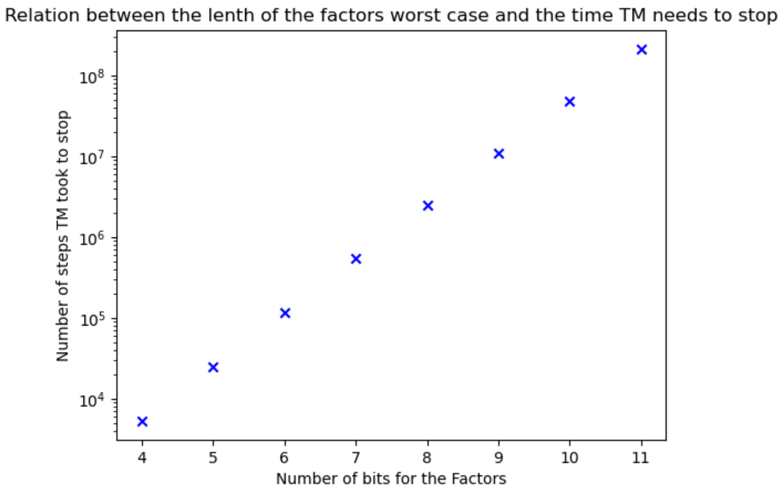

Figure 10: Relation between the length of the factors and worsk case steps of the TM

NOTE 2 OPTIMIZATION PATCH 2: In the Prepare next factors decisions the modification was done to change the initial value of the big factor when one unit is added to the second. Before, the big factor was not always bigger than the other because it was starting again from 4, but with the patch it starts in the current value of the small factor and only can grow. This modification reduced the number of multiplications needed a factor of 2.

- Comparison

  To compare the result of the multiplication, the TM needs to compare bit by bit, and the distance between bits in the input and the result of the multiplication a total amount of $8 + 2(n-1)$. Also the result of the operation can have at most a length of $(n-1) + (n-2)$ bits, so for the whole operation in the worst case where all n bits are compared, the steps needed are:

  $$\sim \sum_{i=1}^{n} 2\left(8 + 2(n-1) + n - 1 + n - 2\right) = 2n(4n+3) = O(n^2)$$

- The TM complete, complexity of the algorithm.

  The factoring problem is, up to date, an NP problem as nobody has found an algorithm to solve the problem in polynomial time. That means that, independently of how efficient the algorithm we have implemented is, the cost will be exponential.

  In our algorithm, the worst case scenario is to have a prime number as input, for which all possible factors will be tried to fit and will fail; the machine won't stop until it reaches the maximum value for the Small Factor. In this case, assuming that the multiplication cost a number of steps N, for an input of length n the number of Small Factors we will have is 2^n, and for each SF m we have $2^n - m$ Big Factors. So, to complete the search for a prime number we need a total number of multiplication+comparison of:

  $$\sim N \sum_{i=4}^{2^n} (2^n - i) = N\left((2^n - 4)2^n - \frac{2^n - 4}{2}\right) = N\left(2^{2n} - 2^{n+2} - 2^{n-1} + 2\right) = NO(2^{2n})$$

  As our multiplication machine is not efficient, the TM implemented will spend a lot of time to run for big values of n. One method to reduce the exponential cost of the multiplication is: every time 2 new factors are set, replace the multiplications with sums. Having as factors f_1 and f_2, which give a product p; when we add 1 unit to f_2 to try again, instead of multiplying again it's possible to find the result of the product just by doing $p + f_1$. This method can work if during the comparison step the result of the multiplication is not deleted, and we will only need the multiplications when we need to increment f_1. The *adding* process will have a cost of $O(2^n)$ because we are only adding a string of length $(n-1)$, which is a reduction of 2^n compared with the multiplication worst case.

  Assuming that we can preform the multiplication in the optimal way, with an the algorithm based on multiplication the base of the exponential is still bigger than for other algorithms. For example, one algorithm that will preform much better that ours is based on division, where a new factor is found whenever the division of the input with the trial number gives a rest of 0. In this case, assuming that the division cost a number N' of steps and bounding the value of the possible factors with the square root of the smallest power of two bigger than the input (of length n), the cost of the algorithm will be:

  $$\sim N' \sum_{i=4}^{\sqrt{2^{n+1}}} = N' \frac{\sqrt{2^{n+1}} - 4}{2} = N'\left(2^{(n-1)/2} - 2\right) = N'O(2^{n/2})$$

By running the TM for all possible inputs with length between 3 and 8 bits, we can plot the relation between the length of the input and the time that TM needs to stop after trying to factorize the input. In figure 11, where the result of this calculations is showed, the exponential growth of the time needed is very clear: the worst case (prime numbers) needs an exponential time to stop. But also we can observe that the time for the best case grows slower and without an exponential rate.

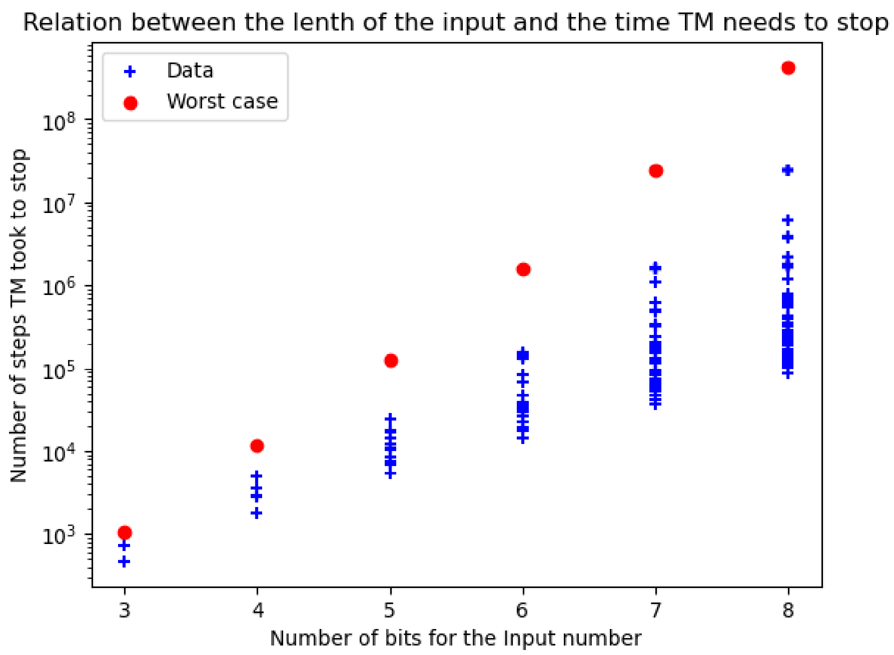

Figure 11: Steps needed to run the TM as a function of the input

1 Files

The most relevant files related with this work can be found in the github repository:
https://github.com/Hectorbm29/MQST-QComputing