Assignement 1

Iván Piña Arévalo
ivan.pinaarevalo@alum.uca.es

March 24, 2019

Abstract

In this analysis, we will make an study about the temporal complexity of two problems: To check if one binary string is palindrome and check if two binary strings are similar. Both problems will be solved by Turing Machine Model and RAM (Random Access Machine) Model. Then, we will study the temporal complexity of the algorithms with the results, and finally, we will get to the conclusion that both algorithms are similar in the two models. Afther this, we will apply Cobham-Edmons' tesis at these programs.

1 Introduction

First, To remember what is an palindrome is useful. By RAE's definition:

> "Word or expression that is equal if it is read from left to right or right to left"

So, an example of binary string could be:

11011

And an example of binary string no palindrome could be :

10011

Given by reviewed the palindrome concept, we are ready to go to the next exercise.

The next exercise consists on to check if two binary strings are equal. If two binary strings have the same digits in the same position, then they are equal.

2 Methodology

First, we will explain the approach used to solve each problem.

2.1 Turing Machines

2.1.1 Palindrome binary string

Habitually, when we have to face to the decision about if a binary string is palindrome or not, we check if the first position is equal to the last. Next, we look the adjacent position and make the same operation. This action we will repeat to check all positions in the string.

At the Turing machine level we have made something similar. As we can see in the picture, there are two symmetrical paths above and down respectively. They are required so they represent the fact of the position found is 0 or 1, depending on the value of the first position we will take a path or another.

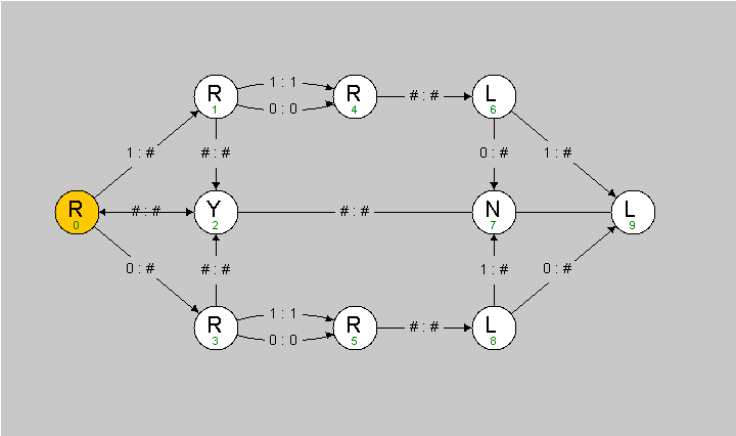

Figure 1: Palindrome Turing Machine

In the middle we have two final states:

- Y: the binary string is palindrome
- N: the binary string is not palindrome.

The performance is the following: The machine looks the first position value, change that value for and then go to the last position. If the value of the last position is the same that the first, then it changes that value for and go to the first position to repeat the process with the present first position (previously that was the second). If the value of the last position is different with the one of the beginning, the machine will finish at the final state N.

2.1.2 Equal binary strings.

In this case, it consists in checking the similitude between two binary string digits. For that, we will add the character & as a both string separator to the Turing machine alphabet. We can watch the turing machine structure in the following picture.

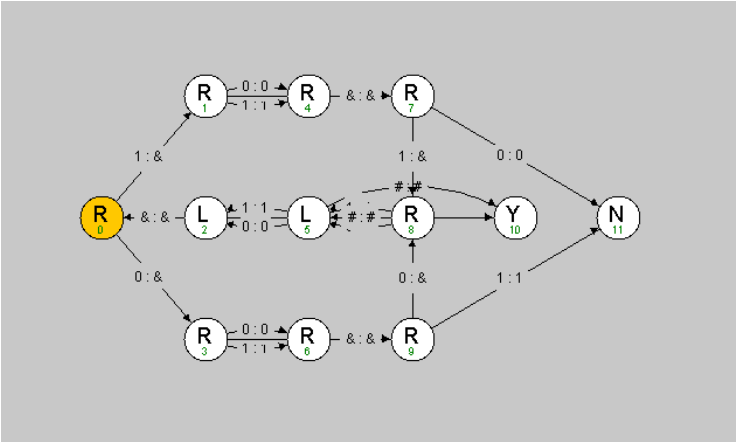

Figure 2: Checker equal binary string

Like the case of the palindrome strings, we can observe two path well differenciated depending on the first position of the first string is 0 or 1.

In the intermediate path are the neccesaries nodes to return to the initial state once a pair of positions have been checked so pass to the next pair. Likewise we can observe the final nodes:

- Y: The strings are equal
- N: The string are different

2.2 RAM model

2.2.1 Palindrome binary string

Like the program in Turing Machine, we check the first and last position. If they are equal, the program proceed to load in r3 and r5 the values of the adyacent positions and it repeats the same operation.
Checking if two registers have the same value consists in decrementing both and see if the two reach 0 value at the same time. If one is smaller, we can jump to the label *end* with the register 5 with value 1. The register 5 is used as a flag. If the program ends with r5 equal to one, the binary string is palindrome.

```
void program()
{
  //Load registers begin(1) and end(2)
  reset(0);
  reset(1);
  reset(5);
  load(2, 0);
  inc(2); //After label begin register number 2 decrement

 begin:
  inc(1);
  dec(2);
  reset(0);
  cgoto(2, carga);
  cgoto(0, end);

 carga:
  load(3,1);
  load(4,2);

 repeat: //Check if  r3 != r4
  dec(3);
  dec(4);
  cgoto(3, a1);
  cgoto(4, b1);
  cgoto(0,end); //Both numbers are equal

 a1:
  cgoto(4, repeat);
  inc(5);
  cgoto(5, end);

 b1:
  cgoto(3, repeat);
  inc(5);
  cgoto(5, end);

 end: //Store r5 in m[0]
  reset(0);
  cgoto(2,begin);
  store(5,0);

}
```

Figure 3: Palindrome binary string in RAM

2.2.2 Equal binary strings

Here, we observe that binary string has an associated integer value. If the two binary string are equal, they have the same value. Thinking in this way, we can see the problem like checking if given two binary numbers, they are aqual.

```
7
8   void program()
9   {
10    //Load in memory
11    reset(3); //Flag, 0 if equal.
12    reset(0);
13    inc(0);
14    load(1,0);
15    inc(0);
16    load(2,0);
17
18    reset(0);
19  repeat: //Start comprobation
20    dec(1);
21    dec(2);
22    cgoto(1, a1);
23    cgoto(2, b1);
24    cgoto(0, end);
25
26  a1:
27    cgoto(2, repeat);
28    inc(3);
29    cgoto(0, end);
30
31  b1:
32    cgoto(1, repeat);
33    inc(3);
34    cgoto(0, end);
35
36  end:
37    reset(0);
38    store(3,0);
39
40  }
```

Figure 4: Checker equal binary strings RAM

We use the register r1 and r2 to load the string 1 and 2. Finally, we check if r1 is equal to r2. We use the register r3 as flag:

- If r3 is equal to one, the strings aren't equal.
- If r3 equal to zero, they are equals.

3 Results and discussions

3.1 Turing machines

In both exercises, the variable that we have used to calculate the cost has been the step number comes expressed in the development environment.

3.1.1 Palindrome binary string

We have obtained the next results:

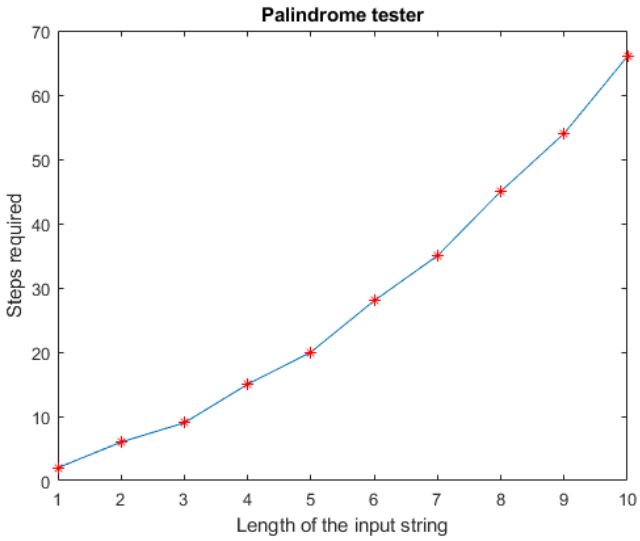

Figure 5: Palindrome Turing Machine execution time

Base on the results obtained, we can make a regression. For that we will use Matlab software. With the adjustment for a polynomial of degree 2 we obtain:

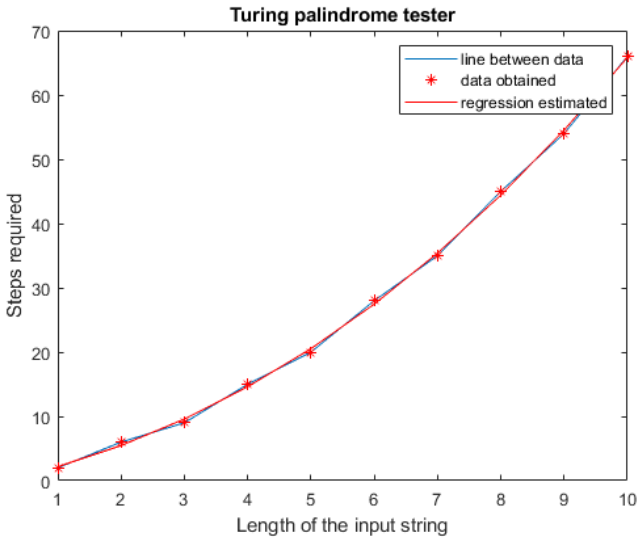

Figure 6: Results of regression

The adjustment is not perfect, but it indicates that the problem has a polynomial order.

3.1.2 Equal binary strings

We have obtained the following results:

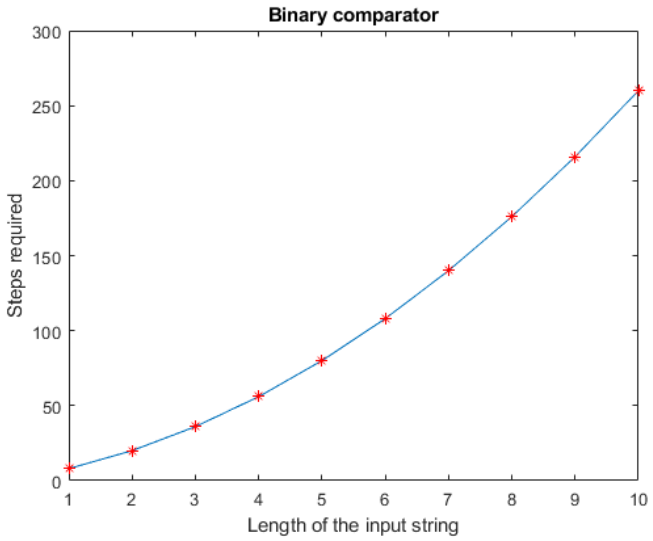

Figure 7: Checker equal string Turing Machine time

observing the results obtained, we have make a regression with Matlab software based on the results to get some idea of the problem complexity.

And now we show the obtained values through calculated coefficients in red:

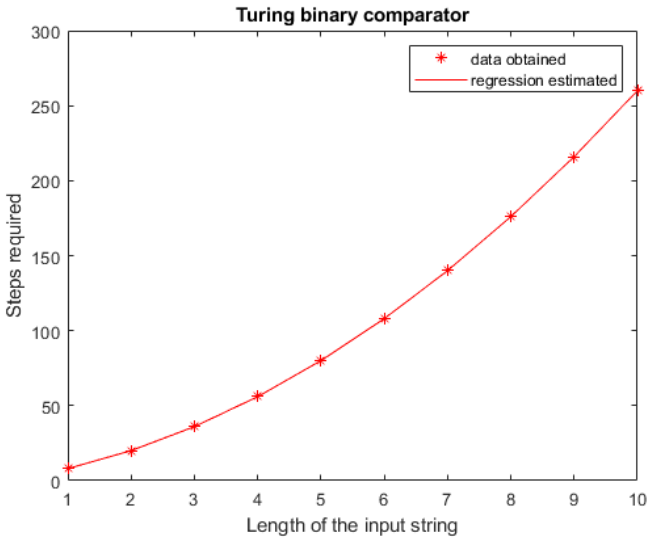

Figure 8: Results of regression

As we can see, the adjustment is almost perfect (it overwrites completely the blue line), then we can conclude that this problem has complexity $O(n^2)$.

3.2 RAM model

In the RAM model the variable which we have used has been the time. For that we have use the clock function() from the library <ctime>. Clock functions returns us the consumed processor cycles, that is why they have to be translated into seconds using $CLOCKS\_PER\_SEC$ macro.

Also if we execute once the program, it gives time measures very different. This is due to the influence of factors like work load at that moment and the run time is very short. To reduce that factor we have executed each test 1000 times and we have calculated the average time that it takes to execute himself. Here we can see an example of the time measure:

```
58    double t0, t1 = 0;
59    for(int i = 0; i < 1000; ++i)
60    {
61      t0 = clock();
62      program();
63      t1 += clock() - t0;
64    }
65    t1 = t1/1000;
66
67    double time = double(t1)/CLOCKS_PER_SEC;
68    cout << "Time: " << time << "s" << endl;
69
```

Figure 9: Measurement of execution time

As well, the computer used to make the tests has the following features:

- Processor:: Intel Core i7 7700K
- RAM memory: 8GB DDR4
- Operating system: Manjaro (distribution based on Arch)

3.2.1 Palindrome binary string

In spite of the measures taken against factors like load in a certain time, there remains an important variability in the results. The proofs to size 3,4 and 9 are anomalus and they can be attribute to processor load or the SO in that certain time. However, as this can sometimes occur it makes sense to leave it reflected.

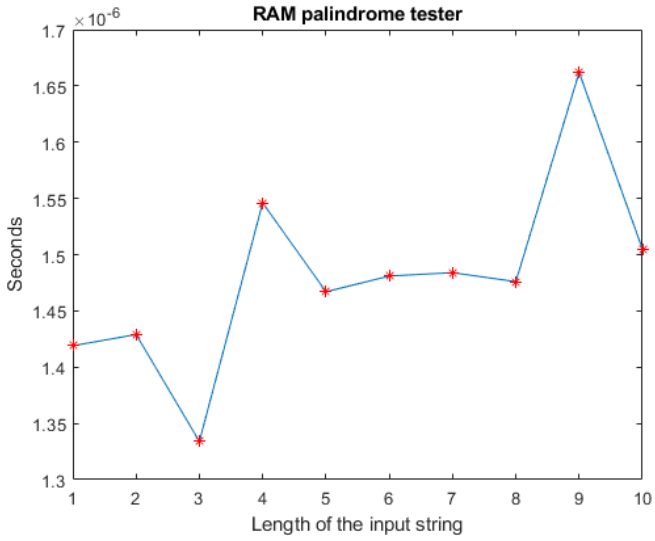

Figure 10: Palindrome RAM execution time

Making a regression, we can see that the problem has a complexity order $O(n^2)$. However, regression coefficient are not appropiate due to anomalous values

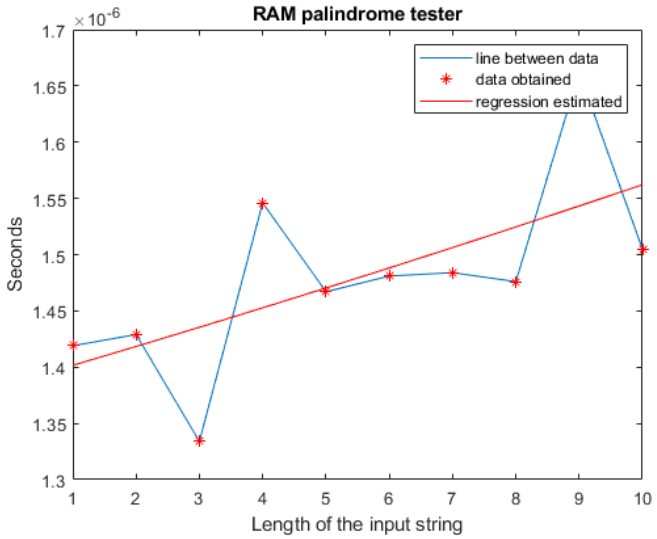

Figure 11: Estimated regression

3.2.2 Equal binary strings

We have obtained the following results:

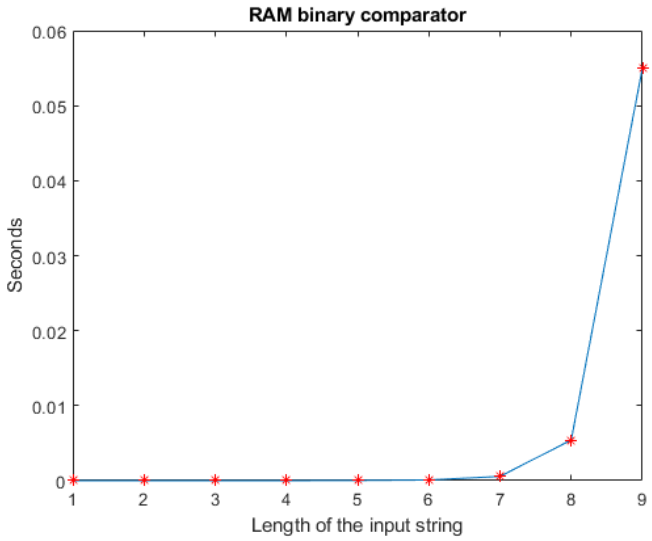

Figure 12: Equal binary string RAM execution time

At first sight we can notice that the graph is totally different with the others, as much in URM as Turing machines. I associate this basic difference to the approach to the problem statement. Instead of dumping the binary string to digit by memory position, I dump each string into one memory position. Due to the binary codification is biyective regarding to their values, if two binary strings are equal they will have associated the same numeric value. On the basis of this, it is enough to decrement simultaneously both strings. In one of them arrives to zero before the other, it means that they are different.

Due to this whole different approach to the one followed in the rest of exercises, it makes sense to think the computational cost can be different to the others in this work.

4 Conclusions

After analysing each exercise, it is coherent to conclude that both problems have polynomial complexity. Accordingly, Cobham-Edmons thesis is satisfied, due to the problems in both computational models have an equivalent temporary costs.

5 References

- Computational Complexity. A Conceptual Perspective. Odded Goldreich, Cambridge
- Introduction to the Theory of Computation, Second Edition. Michael Sipser, Thomson.
- https://av03-18-19.uca.es/moodle/pluginfile.php/103085/mod_resource/content/7/ram.html
- https://introcs.cs.princeton.edu/java/52turing/