BABY STEP GIANT STEP

Discrete logarithm problem

Marc Cervera Rosell

47980320C

Computational tools for problem solving

Josep Maria Miret

Jordi Pujolàs

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

Index

Instructions

The first exercise is to write a code to solve the problem of the baby step giant step and the second one is to compute a few of examples given by the professor.

Exercise i:

To solve the first exercise, we've used the python programming language with Python 3.10 interpreter.

First of all, I've to clarify that the pylint errors disabled are only aesthetic errors, for this reason are disabled.

If we open the *exercise_i.py* file, we'll see a total of 5 methods, each of one compute a different task.

Build_list_1 ():

This method generates the firs list in the way that describes the instructions. So, this list will contain the tuple: $(y * g^j \ mod \ n, j)$.

To do it, the first step is to create an empty list that, at the end, will be returned.

Once we've the list, the algorithm performs a *for* loop that is going to compute every value of the tuple and will append it to the list.

Finally, the list is returned.

Build_list_2 ():

This method, runs exactly equal to the method above, the only think that change is the tuple format.

In this case the tuple will be: $(g^{s*i} \ mod \ n, \ s * i)$.

Search_match ():

This method receives the two lists as parameter and it search two tuples that have the same first component.

To see clearly what does this method, in the following pictures, is going to be showed an example of execution (of this part):

```
List 1 =  [(19, 0), (34, 1), (16, 2), (66, 3), (6, 4), (7, 5), (20, 6), (47, 7), (43, 8), (62, 9)]
List 2 =  [(1, 0), (7, 9), (49, 18), (59, 27), (58, 36), (51, 45), (2, 54), (14, 63), (27, 72), (47, 81)]
```

```
The match is between the tuples (7, 5) of the first list and (7, 9) of the second list
```

To implement this algorithm, we've coded two nested *for* loops. Both of these loops will go from 0 to the length of the first list (because both lists have the same length). The first loop, will control the tuple that we're reading in the first list and the second loop will control the tuple that we're

reading in the second list. The behaviour of this algorithm is very easy. For every tuple of the first list, we'll search a match (the first component of the two tuples is equal) with a tuple of the second list, and when we've found it, the algorithm will stop returning the tuples that make match.

Bs_gs ():

This method, is very easy to understand. Only calls the other methods to make the calculations and prints information about it.

This method, only calculates the value of 'x' and, of course, prints the value once is calculated.

If __name__ == "__main__":

This is the main method but it doesn't solve the second part of the practical case.

This method is only used to test the different functions in the file.

The behaviour are only 4 code lines and 3 of them are keyboard inputs. The last of them is a call to the *bs_gs()* method with the keyboard parameters.

Exercise ii:

This exercise is performed in another python file and only has the main method. Inside of it, we only call the *bs_gs()* method with the parameters given in the instructions.