UpSolve: Stuck on a problem?

Pranjal Walia

" **I will always choose a lazy person to do a difficult job...because he will find the easiest way to do it..**"

~ **Bill Gates**

1 Introduction

The best way for beginners to get better at programming and problem solving is through competitive programming. It's quite easy in the beginning to do trivial things like finding the average value of an array, but what happens when the problems start to get more and more difficult? People say to try your best and solve the problems on your own, but what happens when you want a faster implementation of the problem, or you just can't get it? Then starts the journey of scrolling through all the submitted codes and finding the most efficient solution.

"**After solving it, look at others' submissions, try to implement it yourself, that's where the real learning happens..**"

~ **A Senior**

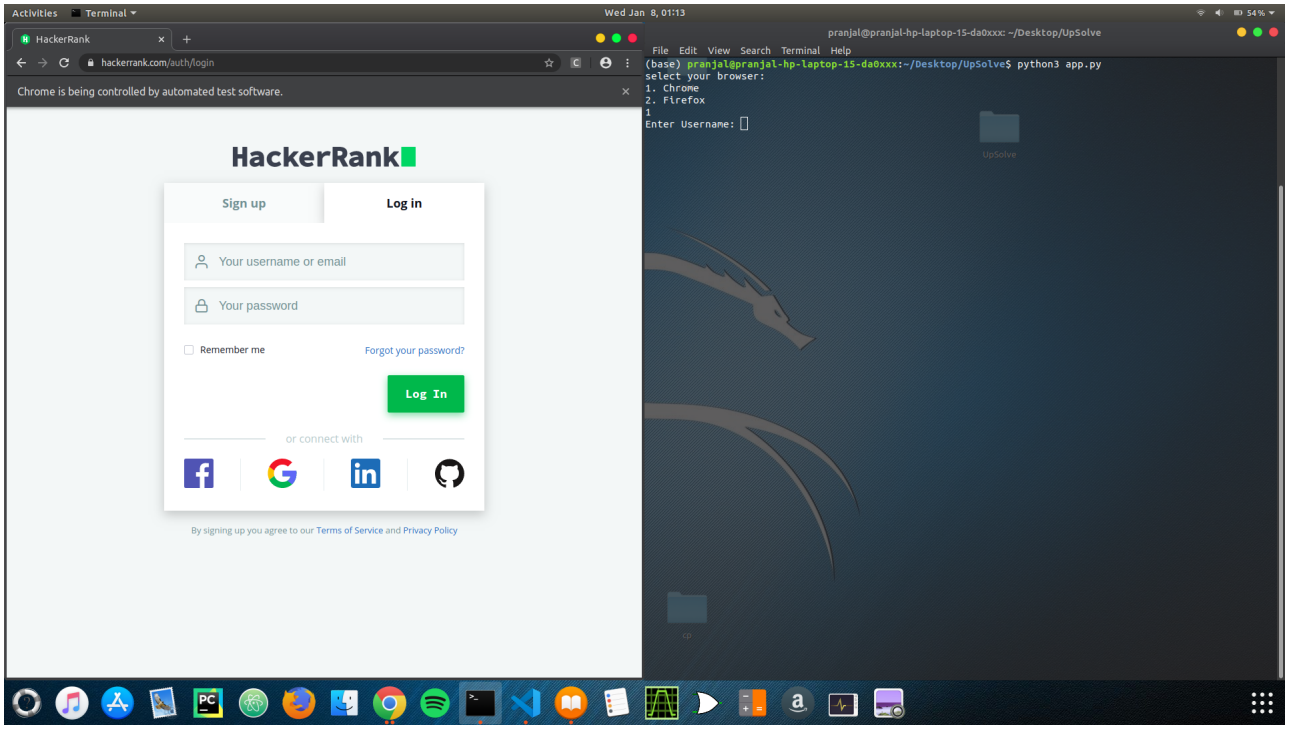
Similar was my situation when I started programming not that long ago, so I decided to make something that would do all this for me, all I would have to do is tell it the name of the problem, and I get a copy of the highest rated submission for it on my system.

2 Implementation details:

I have used the selenium framework to automate the process of searching for a particular solution to a problem on Hackerrank. The entire codebase is written in python3.x as it uses some relatively new features like "f strings". I have tried my best to give the user freedom of choice including deciding the browser he runs it on, currently the script is supported only on chrome and firefox as they are the most popular choices. The user is also free to set his default language choice in the script, else the default is assumed to be C++.

The working of the script involves the following processes:

2.1 Opening



The browser opens up with the login page of hackerrank.

2.1.1 Logging In

I have to enter the username and password for my account in the terminal.

* **for a single user, this can be made faster by hard-coding the credentials in the script very easily.**

2.1.2 Finding the Target Element

After the login, the browser opens up in the dashboard page, the browser then automatically clicks on the 'problem solving' tab. This opens up a page that leads to the set of available problems. At this time the terminal prompts to enter the problem name, I search for the problem using the `find_element_by_partial_link_text()` function of selenium.

This will still locate the problem link even if you're unsure of the complete name of the problem, all you have to do is **enter the part of the problem name you remember correctly**.

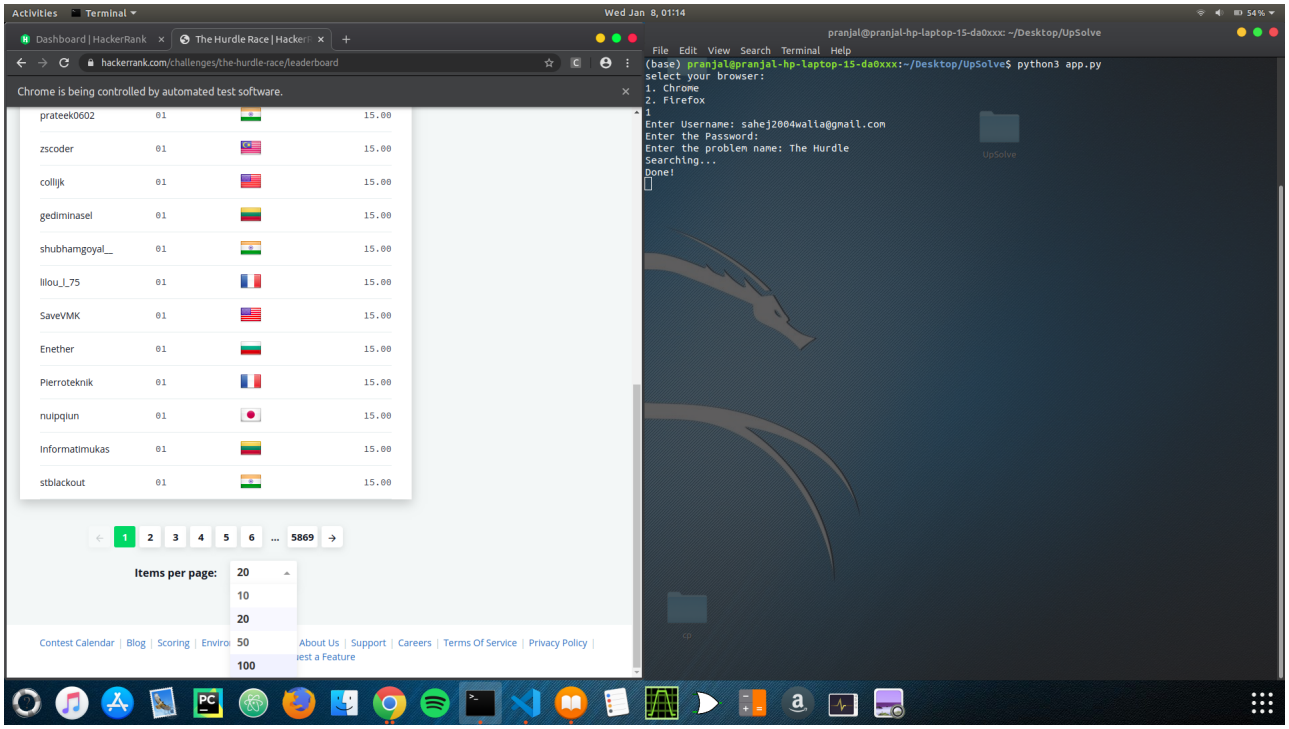
If the problem name involves **some complex characters**, it is best to **just leave out that part and enter the rest of the part you remember.**

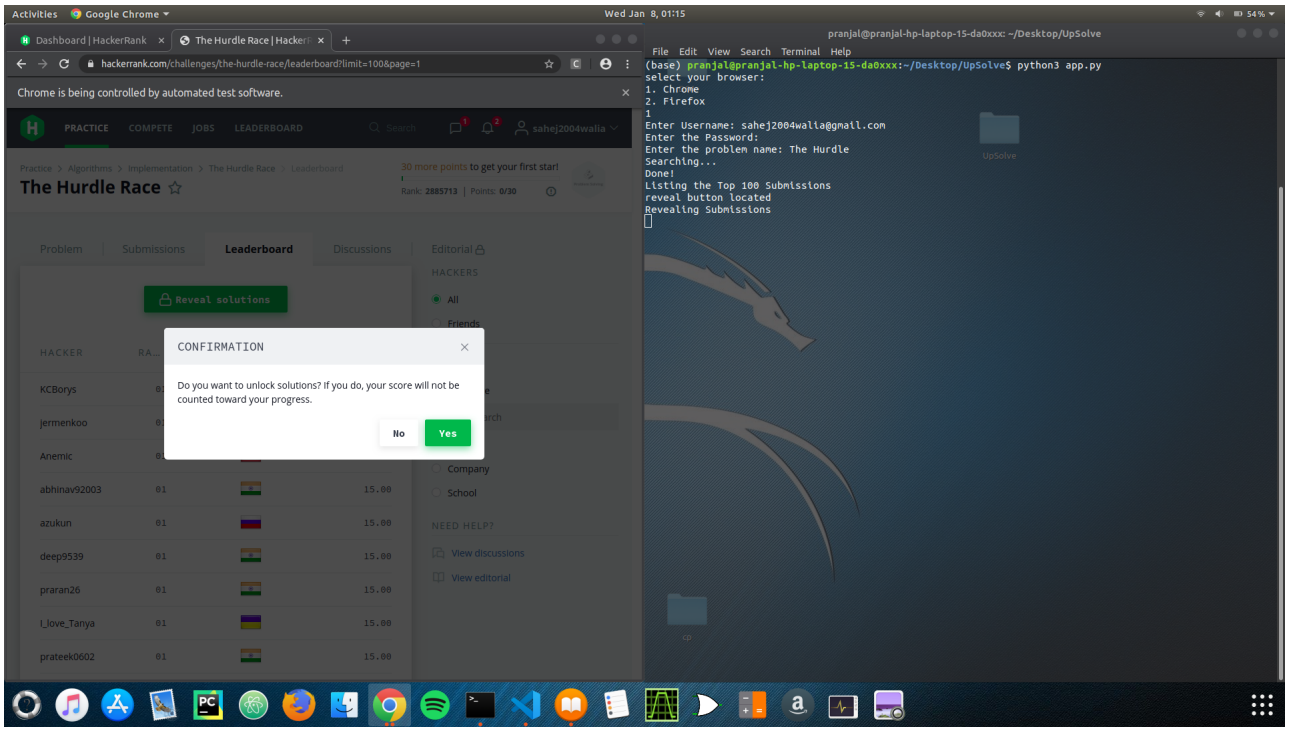
For example, a problem called "Bon Appétit", to search for this just enter "Bon" when prompted in the terminal and this will search for the problem link.

2.1.3 Opening the problem statement

The browser will click on the problem link after it has been located, this opens up the problem statement.

2.2 Opening the leaderboard for the problem

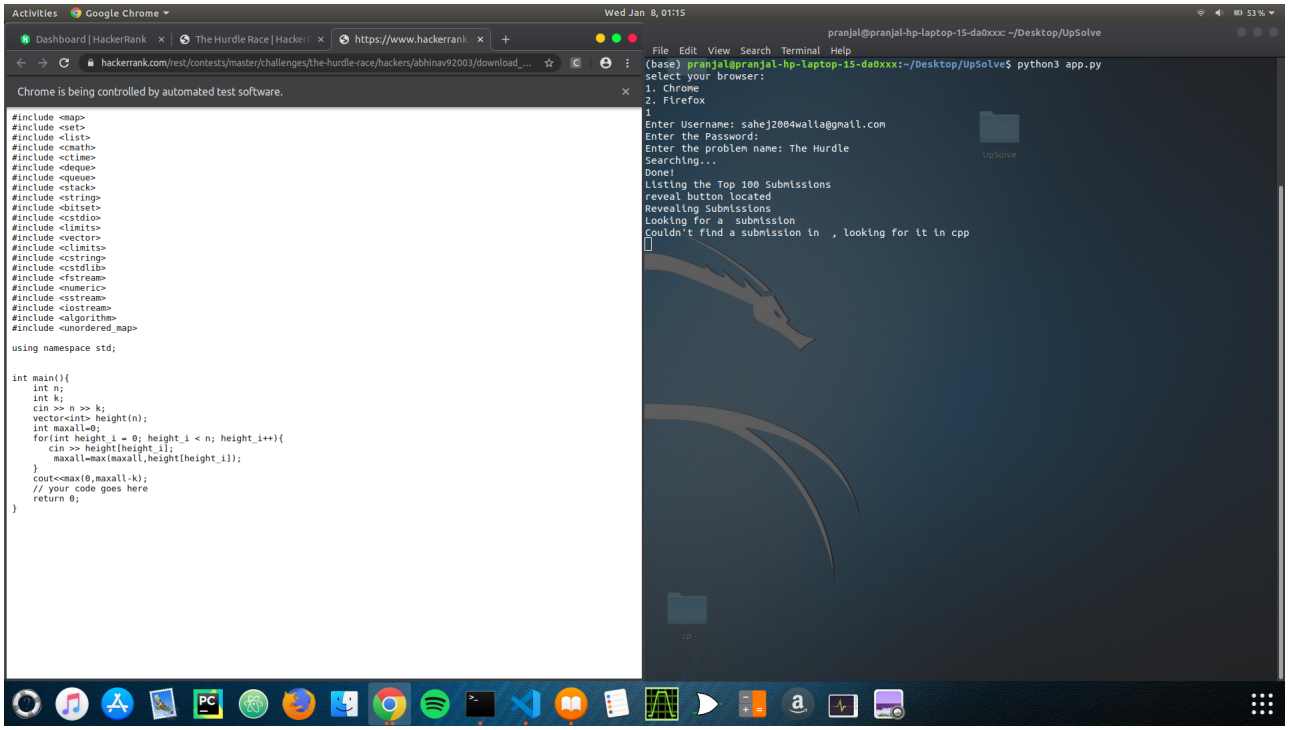




In the problem statement, I need to locate and click on the leaderboard link for the given problem. For this, I search for the link to the leaderboard using the XPATH in the source code. I also learned how to locate a link that has a certain distinct attribute using XPATH and how I could define my own custom XPATH to find this. This made me familiar with the usage of XPATH with selenium and I think perhaps XPATH is the most powerfull way to locate elements on a webpage.

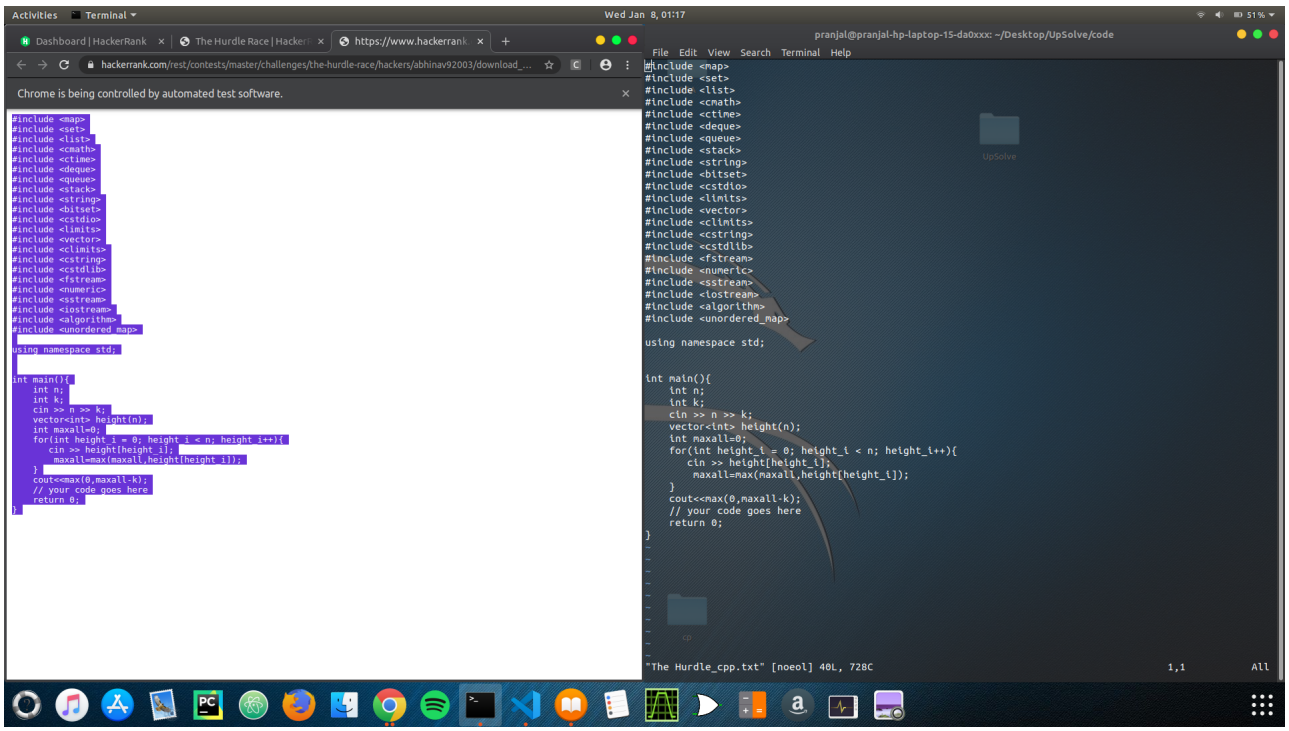
2.3 Opening the solution from the leaderboard:

At the leaderboard, the script searches for the solution of the problem submitted in **user defined** language. If a solution is not available in the top 100 for that specific language, then the script switches to it's default and searches for a solution in **C++**. All this is done on the first page of the leaderboard for the problem as it has the highest rated solutions. For opening the submitted code, I perform a search on the leaderboard to locate a submission in **User defined** language and later on in C++. For this, I use XPATH of the links, the indivigual links have an attribute of the data tag to hold the language of submission. Based on this, I define an XPATH to search for the link of submission, for example, **" //a[@data-attr2='cpp'] "** for C++ and **" //a[@data-attr2='python3']"** for python3.



2.4 Saving the code for the submission

After the submission link is located, it's link is clicked and this opens the submitted code in a new tab. I locate the tag using XPATH in which the code is stored in the page source, then all I have to do is find the text(submitted code) stored in the defined tag and store it in a text file. A folder named **code** is created in the project directory upon sucessfull execution of the script that contains the submission code in the form of a text file by the name **`entered_name_lang.txt`**.



2.5 Database.py

The script has one major **drawback**, it's mechanism for searching amongst the problems is prone to fail sometimes because of **human error**. As a result of this problem, I have created another script **"database.py"** that writes a list of all problems in a file **" problems.txt "**, present in the code directory that will be created automatically after running database.py. The problem name can be directly pasted from here when prompted in the terminal. This script uses **requests** for getting data and is not automated using selenium, therefore takes very less time, finally it helps the user to make an efficient search.

3 Problem Searching Guidelines

A little something to optimise your searching methods to find the required submissions...,

All problems names on Hackerrank follow a simple convention, all words written in the problem name start with capital letters the only exception for this is the case of simple connecting words like **and** , **the** , **on**, **a** , **is** , **in**.

Some Example problem names are like **ACM ICPC Team** , **Bigger is Greater** , **Designer PDF Viewer** , **Cut the Sticks**.

However, If at any point you're unsure about a problem name, just execute the database.py file, it will generate a list of all the problem names and write this in " problems.txt " file in the code folder that will be generated upon execution of database.py , you can directly paste the name from there.

Refer to **Section 2.1.2** Finding the Target Element for more.

4 Additional features

1. I have used the **ExplicitWait** function in selenium using **WebDriverWait()** to wait for links to load completely so that they are visible when searching for them as internet speeds can vary.

2. I have used the **time.sleep()** function in python to pause the execution of the script at certain points of time in order make the search process more accurate and also to let the user have a look at the links that are being clicked automatically.

3. The user is allowed to choose the browser according to his preference.

4. The user is free to give the language in which he wants a solution.

5. I have implemented a check-login feature in the script that verifies if the user login is sucessfull or not. In case of invalid login, the script terminates and prompts the user to re-launch it.

6. A small thing about the hackerrank UI, if you've not solved a problem before, then to reveal it's solutions, hackerrank does not reveal the submission codes, until you agree to forgo points for that problem. The script is programmed to click on the 'reveal' button to overcome this, so as to open all the submissions.

7. I have used the **getpass** module of python to input the password from the user, so that it is not visible on the terminal while entering it.

8. The leaderboard, by default is set to open only the top 20 submissions, however to help the user find a submission in his desired language most of the times, the script will display always the Top 100 submissions after clicking on the **submissions per page** button.

5 Future scope

1. I plan to extend the script to it's original purpose, to scrape contest problems and their respective editorials.
2. I plan to make the search process more resistant to user error, more efficient and faster. This think that I can streamline this process using machine learning algorithms in the future.
3. I plan to make more modules, so that I can extend it to other platforms like CodeChef and Codeforces.

6 Project Experience

1. I learned how complex webpages are structured and how different elements of webpage can be identified using many ways, the most powerfull being XPATH.
2. I learned how to use to the "Inspect element" feature of browsers according to my prefrences.
3. Most importantly, I learned the art of self-help as there was nobody to help me with the project. I learned how to use selenium for automation in a relatively short period of time and the entire process was filled with a lot of experimenting, trial and error.
4. I learned how to use resources like google and stackoverflow more efficiently.
5. I also had many first experiences, the most promenient one was asking questions on stackoverflow, then later on realising that the stackoverflow community is brutal.
6. Working under a deadline made me realise the importance of proper planning before starting anything.

Github Link: `https://github.com/masterchief01/UpSolve`
Demo Video: