Analysis of TopCoder Programming Competition Data
By Shawn Bradford, Kevin Dunn, and Joseph Connor
December 2015
University of Tennessee

Overview

For our final project, we performed a statistical analysis of several different features on the popular programming competition website TopCoder. We gathered a large amount of data from the website using automated web scraping and analyzed a wide variety of measures, including comparisons of programming language usage, programming language success rates, challenge rates, programming times, statistics related to programmers' countries of origin, correlations between successful coding and challenging, and site usage over time. Our principal goal was to investigate and gain insight into the relationship between programming language of choice and programmer ability.

Background

TopCoder is a website where users compete against each other in programming "matches." Competitors are separated into two divisions based on past performance, Div1 and Div2. Users with a higher rating are placed in Div1. At the start of a typical match, each competitor is presented with the same set of three problem statements at three tiers of difficulty with corresponding point values, each of which gives a specification of a program they must write within an allotted time limit. After all competitors have submitted their code, they are given a chance to view others' solutions and "challenge" them if they think the solution is incorrect. Users' final scores are then calculated based on the problems successfully solved, the time it took them to solve each problem, and their challenges of others' code.

A problem statement typically includes some background story to give the problem context, a precise (often mathematical) description of the behavior the program must have, a domain of possible inputs, the expected format for program output, and several example input and output pairs (often with a short explanation for why the output is what it is). Each user can select their own programming language preference from a list of several languages, and receive a problem statement tailored to that language with input and output types given in data structures common for that language (for example, vector<string> for C++ versus String[] for Java). Programs are also given time and memory constraints, typically 2.000 seconds of execution time and 64 megabytes of memory. The difficulty ranges from relatively trivial for the lowest-tier problems (such as writing a Huffman decoder given an input string and encoding), to the highest-tier problems which often require an obscure mathematical insight in order to write a program that can operate within the available time and memory constraints.

After all users have submitted solutions, they are also given a chance to view all the other solutions and challenge any which they believe to be incorrect. Upon challenging a

program, that program is tested against an extensive list of predefined input and outputs. Any output which differs from the expected output or exceeds execution time or memory constraints causes the program to be considered incorrect. If a challenge is successful (meaning at least one output differed from the expected output), the challenger is awarded 50 points and the person who submitted the incorrect program receives no points at all for their submission. Otherwise, the challenger loses 25 points.

Finally, any program which has not already been successfully challenged is tested automatically. Again, any program which differs in at least one output is disqualified and receives no points. Each user's final score is calculated as the sum of their scores for each problem, plus any points or deductions for successful or unsuccessful challenges. Points for each problem are calculated as the problems base score, minus a factor dependent on the amount of time between the moment the user opens the problem statement to the moment their solution is submitted.

Method

TopCoder provides statistics on the outcomes of many of their past matches, as well as statistics on the performance of each member. We primarily used the Python programming language in the form of interactive Python notebooks to perform retrieval and analysis. For storage, we primarily used MongoDB. For analysis and data visualization, we used a combination of Python and R.

One challenging aspect of the data retrieval portion of this project was that TopCoder does not provide an official API for any of their statistics. In addition, statistics of different matches are scattered across different pages, and statistics for individual competitors are given in an entirely different section of the site, and in a different format.

For retrieving statistics related to match outcomes, we had to write scripts to process the HTML of several pages in order to aggregate the statistics we needed. Parsing the HTML was challenging in many cases, because the pages are designed to be formatted for human-viewing, not machine processing. We used the requests library to simplify the retrieval of the raw HTML, and a combination of the BeautifulSoup library and Python's built-in regular expressions module for parsing the HTML. Starting from the list of past matches, we extracted the links to the statistics for each f using BeautifulSoup, then parsed each match's page using regular expressions. By examining the HTML of the pages, we were able to construct regular expressions that would extract the values we needed.

Retrieving user statistics was particularly difficult at first because they are loaded asynchronously onto the user's profile page using Javascript. Therefore, our usual method of downloading and parsing the HTML would not suffice for this case. Instead, we examined the web traffic generated by the page and found the undocumented HTTP API that the Javascript was using to load retrieve the user statistics. We then wrote a script to interface with this API to retrieve the data. Data retrieval actually ended up being somewhat easier this way, because data was returned in a machine-readable JSON rather than the typical human-readable HTML.

Results

We found that by far the most used programming language in Division 1 of TopCoder is C++. There have been 267,363 C++ submissions, which is more than four times as many as the second most popular language, Java, at 59,860. In fact, C++ submissions account for about 78% of all Division 1 submissions. This shows that C++ is clearly the language of choice for most Division 1 competitors. Microsoft's C# programming language was in third place at 13,487 submissions, but still dramatically less popular than either Java or C++. In comparison, the Visual Basic and Python programming languages saw almost no use at all with only 761 and 706 submissions, respectively.

In Division 2, the distribution of language usage was very slightly flatter, but the order of preference was mainly the same as Division 1 except for Visual Basic and Python. C++ was still the clear favorite with 324,222 submissions, followed by Java with 142,258 and C# with 27,782. Python was significantly more popular in Division 2 than Division 1 with 5,132 submissions. Visual Basic was by far the least popular in Division 2 with only 2,049 uses.

In contrast to the wide disparities in language choice, we found only small variations in the percent correct among different languages. However, for DIvision 1, C++ did prove to have the highest proportion of correct submissions (67.5%), which suggests that its overwhelming popularity may be due to its effectiveness. Again, Java was in second place with 65.5% correct, followed by C# with 64.6%, Visual Basic with 61.4%, and Python was again in dead last with only 58.4% of submissions correct.

For Division 2, more submissions in general were correct, and the disparity between languages was even less. But C++ remained the most often correct language with 71.1% correct. In contrast to Division 1, Python was in second place for Division 2 at 67.9% correct. C# was in a very close third place with 67.8%, and Java a close fourth with 67.6%. Visual Basic submission were only correct 63.0% of the time, making it the least effective language for Division 2.

Despite its relatively poor success rate, Visual Basic was by far the least-challenged language in Division 1, with only 31.8% of its submissions being challenged. This, together with Visual Basic's extreme unpopularity among Division 1 coders, perhaps reflects a lack of knowledge of the language among the competitors. C++ was the second-least challenged language with a 37.2% challenge attempt rate, which is probably at least in part due to its high rate of correct submission in general (supposing that competitors are more likely to challenge submissions that are actually incorrect). In third place was Java at 38.0%, followed closely by Python at 38.5%, and lastly by C# at 39.3%.

Challenge attempts were drastically different in Division 2. Here, a mere 21.2% of Python submissions were challenged, making it by far the least-challenged language for Division 2. C++ was in second place again with a 31.0% challenge attempt rate, followed by C# with 32.2%, Java with 33.6%, and Visual Basic with 33.7%.

Surprisingly, Python lead both divisions in successful challenges. Despite being the least-challenged language in Division 2, challenges against submissions written in Python were

most likely to be correct (51.7%). Challenges were least likely to be correct against C++ submissions in both divisions, with only a 39.6% successful challenge rate in Division 2 and a 44.9% successful challenge rate in Division 1.

One particularly surprising result was the incredibly low average submission time for Visual Basic, across both divisions. In both divisions it had the lowest average submission time, but the difference was most dramatic in Division 2 where it had an average of 794 seconds compared to the average of 1630 for the second place language, C#. C++ took competitors the longest to write on average in both divisions, with 1770 seconds in Division 2 and 1555 seconds in Division 1.

The top three countries together accounted for over half of all wins. The United States was in first place with 278 wins, followed by Russia at 234 and China at 202. Canada, Poland, Japan, and Ukraine were the next top three with 79, 76, and 47 wins, respectively. Unfortunately, statistics on total number of rounds played by each country were not available, so we had no way of normalizing these figures to a win rate for each country.

Another interesting result was the difference between number of wins per player among the divisions. In Division 1, there were a couple of dominant players (one with over 100) and several other high-ranking players with 10s and 20s of wins. In Division 2, however, only one player managed to have even 5 wins, and the vast majority of the rest had only 1 or 2 wins. This is almost certainly because players that do well and win more than one game in Division 2 tend to simply move up to Division 1.

Among individual players, we did find a correlation between their ability to write correct code and their ability to find incorrect code. There was a correlation of 0.40 between players' submission correctness rate, and their challenge success rate.

Finally, we found that over time TopCoder has been growing rapidly. None of the first 200 matches had more than 400 players. However, now, at around 600 matches, many have almost 1400 contestants (although some matches still have fewer than 400 players). Around match 380, there was an unusual cluster of matches that all had fewer than 50 competitors, although the cause of this was unclear.

Conclusion

From our data, we conclude that C++ is the most effective language for TopCoder. It had the highest proportion of correct submissions for both divisions, as well as the lower proportion of correct challenges against it in both divisions. It is also clear that Python is generally a poor choice for TopCoder, with a lackluster success rate in both divisions, although it does do much better in Division 2 than in Division 1, probably because Division 2 problems are often much less constrained by the time and memory constraints, a major issue with Python. Not too surprisingly, our results do seem to suggest that those who are better at writing code are also better at reading it (due to the correlation between submission and challenge correctness).