MAC0214 - Extracurricular Activities

30/6 Project final report

First Semester of 2019

Professors André Fujita[1], Denis Deratani Mauá[2], Kelly Rosa Braghetto[3]

Undergraduate Vitor Santa Rosa Gomes, 10258862[4]

Supervisor Nathan Benedetto Proença[5]

This is the final report of the project "Algorithm Designing with Competitive Programming" presented to the course "MAC0214 - Extracurricular Activities" held on the Institute of Mathematics and Statistics - University of São Paulo (IME-USP), during the first semester of 2019. The project is part of the routine activities of the MaratonUSP extracurricular group and reviews the underlying theory, strategies, jargon and tricks used on Competitive Programming and their challenge problems, being categorized within the Computer Science field. The following text is divided into four sections, "Problem", "Input", "Output" and "Standings" — roughly inspired by the structure of the field's challenge problems —, respectively (i) introducing the project and defining terminology, (ii) specifying the initial objectives and planning, (iii) describing the methodology and the schedule of the project, and (iv) discussing results and further work.

Keywords: *Competitive Programming*; *Algorithms and Data Structures*; *Extracurricular Activities*.

PROBLEM

Competitive programming is a mind sport in which the participants try to solve programming challenges proposed in a **programming contest**, having grown steadily in popularity during the last decade, as more routinely, public and internet-based contests took place. Competitive programming has its roots on the design and implementation of algorithms, as the core of competitive programming is about inventing efficient algorithms that solve well-defined computational problems. Still, Mathematics plays an important role in competitive programming, so there are no clear boundaries between them. Afterwards, the solutions to problems are evaluated by testing an implemented algorithm using a set of test cases: it is necessary to not online correctly answer the problem, but also to obey some time and memory constraints. Thus, after coming up with an algorithm that solves the problem, the next step is to correctly implement it, which requires good programming skills. In contrast, competitive programming differs from traditional software engineering because programs are usually short, quickly written and have no need for maintenance [AL].

International contests in the field of competitive programming started during the 70's decade and have massively thrived not only in variety, but also in covered countries and number of attendants. Two of the most traditional and prominent international contests are the *International Olympiad in Informatics* (**IOI**, since 1989) — an annual contest for secondary school students — and the *Association for Computing Machinery's International Collegiate Programming Contest* (**ACM-ICPC**, or just **ICPC**, since 1970) — an annual programming contest for university students.

Due to both those international contests covering already more than 80 countries and taking place in person (not through the Internet), each country has to select its team or candidate representative. In Brazil, the national contests *Olimpíada Brasileira de Informática* (**OBI**) and the *Maratona da Programação da Sociedade Brasileira de Computação* (**SBC** or **Maratona**) offer a position to its finalists in the former international contests, respectively. Regarding this, MaratonUSP, originally MaratonIME, came up as an extracurricular group to organize and host the subregional contests of SBC, both for individuals and teams, as well as to gather people interested in competitive program-

[1] fujita@ime.usp.br
[2] ddm@ime.usp.br
[3] kellyrb@ime.usp.br
[4] NUSP 10258862, vitorssrg@usp.br
[5] nathan@ime.usp.br

ming training and sharing knowledge. Now, the group also offers classes that introduces competitive programming to newcomers and publishes list of essential and classical challenge problems.

A competitive programming challenge problem, or just **problem** for short, relates to a standard ICPC problem format. Even allowing some narrow variation, all the problems treated in this report resemble at least a subset of the following structure down below. For better illustration, on the side there is Problem 1: CF_325_B - Stadium and Games, a median problem, an example of an ordinary problem.

1. The **problem title** or name, prefixed with its identification code within the contest.
2. The **problem's tags** and difficulty level, if the contest have already ended, but it is still open for non-rating submissions.
3. Instructions about the judging resource limits for compilation, runtime execution time and allocated memory, maximum solution file size, and accepted programming languages.
4. The **problem statement**, which introduces a usually fictional story or context.
5. Specification of the input format.
6. Specification of the output format.
7. Some examples of expected output given an input, formatted accordingly.
8. Notes or explanation about the examples.

To solve ICPC-like problems, it is required knowledge about algorithms, data structures and general math, which are usually specified on the grater contests syllabus [e.g. IOIS19], also followed by recent, less traditional contests. Although the covered content seems broad, its depth is considerably shallow, mostly addressed by standard introductory Computer Science courses. In practice, it is also expected reasonable literacy in one of the compiled, optimized programing languages like `C`, `C++` or `Java`.

Focusing on the first two aspects, this report distinguishes Algorithms and Data Structures as follows — although highly dependant on each other — similarly to [SW].

An **Algorithm** is a complete implementation, a black box function, procedure or routine that receives an input

B. Stadium and Games

BINARY SEARCH, MATH, *1800

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Daniel is organizing a football tournament. He has come up with the following tournament format:

1. In the first several (possibly zero) stages, while the number of teams is even, they split in pairs and play one game for each pair. At each stage the loser of each pair is eliminated (there are no draws). Such stages are held while the number of teams is even.
2. Eventually there will be an odd number of teams remaining. If there is one team remaining, it will be declared the winner, and the tournament ends. Otherwise each of the remaining teams will play with each other remaining team once in round robin tournament (if there are x teams, there will be $\frac{x \cdot (x-1)}{2}$ games), and the tournament ends.

For example, if there were 20 teams initially, they would begin by playing 10 games. So, 10 teams would be eliminated, and the remaining 10 would play 5 games. Then the remaining 5 teams would play 10 games in a round robin tournament. In total there would be 10+5+10=25 games.

Daniel has already booked the stadium for n games. Help him to determine how many teams he should invite so that the tournament needs **exactly** n games. You should print all possible numbers of teams that will yield exactly n games in ascending order, or -1 if there are no such numbers.

Input

The first line contains a single integer n ($1 \le n \le 10^{18}$), the number of games that should be played.

Please, do not use the %LLD specifier to read or write 64-bit integers in C++. It is preferred to use the CIN, COUT streams or the %I64D specifier.

Output

Print all possible numbers of invited teams in ascending order, one per line. If exactly n games cannot be played, output one number: -1.

Examples

INPUT
3
OUTPUT
3 4

INPUT
25
OUTPUT
25 20

INPUT
2
OUTPUT
-1

Problem 1: CF_325_B - Stadium and Games, a median problem

and may return an output or change the program's state. A **Data Structure** is an Abstract Data Type that encapsulates its initialization, its state and inner variables, and a protocol of available operations, queries and routines that may require some input and may produce an output, change its state of the program's state. Nonetheless, most of the time, an algorithm or a data structure relies on other algorithms and data structures. Finally, **Solutions** to competitive programming challenge problems are single-filed computer programms which traditionally contain refer to, implement, tweak or combine both well-known and sometimes specific, field-related algorithms, algortihms, data structures and important theoretical, mathematical results or theorems.

Once one finishes a possible solution to a problem, they submit the source code to an **online judge** platform, which evaluates the solution on an extensive list of test cases, usually hidden from the competitor. If the solution produces a correct output for all given input tests, obeying some resource constraints, it is labeled as Accepted (**AC**). Otherwise, it responds an error code accordingly. Even though the error code depends on the contests specifications, virtually all contests and platforms follow the ICPC's main error codes: Wrong Answer (**WA**), Time Limit Exceeded (**TLE**), Run-Time Error (**RTE**) and Compile Error (**CE**). Commonly an incorrect or defective solution is labeled with the code the firstly arises during the judging pipeline. The **final standings** of the contest are then defined by how many of the problems each contestant or team successfully solved, as well as accounting for **penalties** from wrong submissions and the time taken for the first successful submission for a problem — see ICPC rules.

The growth of the traditional, international contests have promoted not only regional, university-maintained online platforms problems bases, but also websites with interview-based problems. At the moment, the most active online contest platform is Codeforces, which organizes public contests at most every other week. Other public platforms that host a contest at least once a month are AtCoder, URI, HackerRank, CodeChef and topcoder. Finally, some great online problem databases and online judges aggregators are POJ, URAL, CSES, A^2 Online Judge and CS Academy [AL].

INPUT

Designing and implementing algorithms and data structures for a task, given some constraints and effectively managing available resources is a mandatory competence for any Computer Scientist. Even though standard courses offer a great introduction to those fields os study, it is usually impractical to review both broadly and meticulously the current state of art and the traditional solutions do real-world problems.

With that in mind, the objective of this project is to learn and practice algorithms and data structures not thoroughly covered by traditional algorithm analysis courses, as competitive programing problems offer a neat environment to both development algorithm design and enrich an undergraduate CS student's data structures tool belt. To provide a better view of the progress and diversity of the solutions, every problem solved during the semester would be tagged accordingly with the strategies involved either on its implementation or on its correctness. This compiled problem sheet shall offer to both myself and my colleagues not only a fast look up for algorithms and data structures, but also a list of use cases to understand and practice them.

As the platform Codeforces is currently the most active [AL] online judge and contest holder, offering at least one public contest weekly and accounting with thousandths of problems from past contest available for practice. Most of them follow strictly the ICPC problem format, however, as anyone can publish contests, non-standard variations like quantum programming and april fools contests are also hosted. Those contests are scheduled on Codeforces' calendar around two weeks in advance.

The contests that occur on Codeforces are initially rated as `Div. 3`, `Div. 2` or `Div. 1`, with respectively increase in difficulty. This measurement is previously defined by the **problem setters**, experienced contestants who create the problems statements and deeply understands the available approaches to solve them given the resource constraints. Most of the time, the contests provide from two to three hours to solve six problems, labeled from A to F. After the contest has finished, a **rating** formula calculates a rank for the problems and a new rank for the contestants, founded on the combination of previous contestants' rankings and who of them successfully solved each problem.

Before finishing, some contests may also have **hack** or

system test phases: in the former, every submission is made public and competitors may design an input to degrade a rival's solution; in the latter, the judge perform logger and harder tests on the submitted solutions. Alongside with the final standings, once a contest terminates, the problem setters publish an **Editorial**: a condensed article that comments on approaches to solve the contest's problems, contestant's performance and other announcements. As a result, the problems are accordingly added to the platform's **problemset** database, being available for anyone for practice.

The prior program to fulfill the proposal was to carefully select problems from Codeforces' problemset, ranging lower- to upper-intermediate difficulty and tagged with diverse algorithms and data structures, roughly inspired on MaratonUSP's and MAC0327 - Programming Challenges's lists — see Attachment 1: Project proposal. Nonetheless, it was also intended to actually solve problems during the live contests whenever I'm both available and qualified. Aside form the ones that occasionally happen on Codeforces, during the first semester of 2019, two other major contests took place: Google's Code Jam 2019, Facebook's Hacker Cup 2019. Therefore, it was expected a balance between virtually and lively practice problem solving. Finally, for the sake variety, solving problems from other online judging platforms may also be considered.

OUTPUT

The two main deliverables of this project, namely Table 1: Detailed schedule and Table 2: Problems reference table, respectively concerning the required detailed weekly schedule and aimed lookup table with for problems tags, are attached to the end of this final report: Attachment 2: Project schedule and workload and Attachment 3: Problems reference table.

The initial plan was to **live compete** on every `Div. 3` or `Div. 2` contest scheduled on Codeforces. However, since they normally occur on weekdays during the afternoon on the Pacific Daylight Time (PDT UTC-7:00), the 2 to 3 hours contests mostly conflicted to my regular course schedules — Brasilia Time is BRT UTC-3:00. As an alternative, I **virtually simulated** some of those contests during my spare hours, with the drawback that, in this mode, solving the problems is not accounted to my ranking. This is just a minor concern, given that the aim is to study algorithms and that structures with competitive programming as a medium, not as an end.

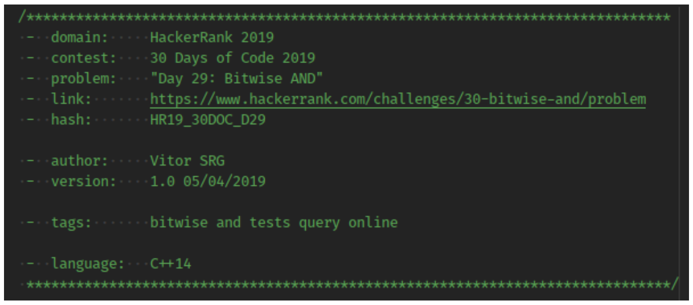
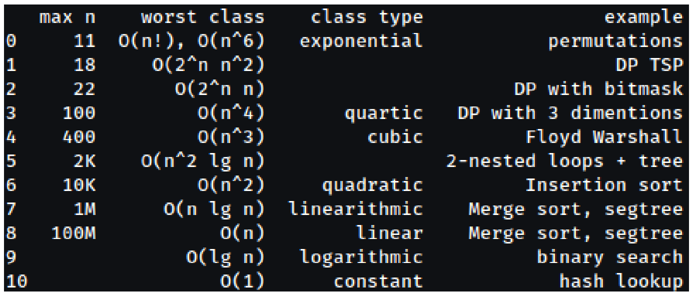

```
/**************************************************************************
  -  domain:    HackerRank 2019
  -  contest:   30 Days of Code 2019
  -  problem:   "Day 29: Bitwise AND"
  -  link:      https://www.hackerrank.com/challenges/30-bitwise-and/problem
  -  hash:      HR19_30DOC_D29

  -  author:    Vitor SRG
  -  version:   1.0 05/04/2019

  -  tags:      bitwise and tests query online

  -  language:  C++14
 **************************************************************************/
```

Figure 1: Example of a problem header

Once lively or virtual participating on a contest, the solutions were designed without assistance; then, generally two days later and after reviewing the published editorial, the problems are revisited and the attempted solutions are revised — this called **upsolving**, a field jargon. All the problems successfully solved or whose final attempt was arguably close to a solution were kept and logged. The reported workload for each contest accounts its duration plus a reasonable time for upsolving, hence experiencing a broad variety of problems is posed as preferable over being stuck on and struggling to solve one.

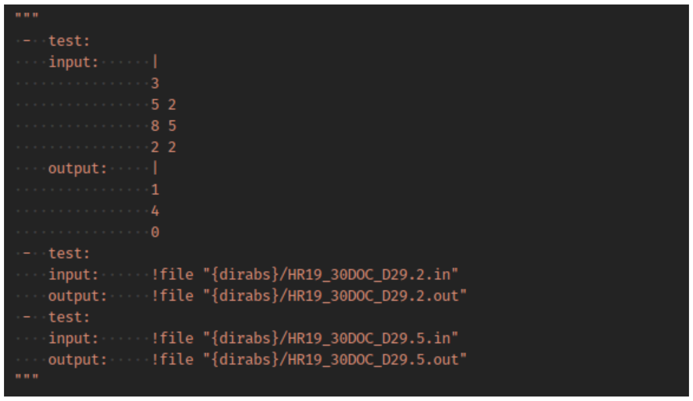
Futhermore, I have live competed on seasonal contests not hosted on Codeforces: Google's Code Jam 2019 Qualification, 1A, 1B and 1C rounds; Google's Kick Start 2019 A, B and C rounds; Facebook's Hacker Cup 2019 Qualification Round; MaratonUSP's Individual Primaries for SBC first round; and VTEX Code Cup first round. Because I was not qualified to further progress on those contests, have also virtually competed on Google's Code Jam 2019 rounds 2 and 3, and on Facebook's Hacker Cup 2019's first round. As well as the Codeforces' contests, the workload for those rounds is the sum of their duration plus a reasonable period for upsolving capped by 5 hours, since some of those rounds were live for more than a day.

Every solution and programs that were very close to solve a problem have been archived and tagged with extensive, meaningful and descriptive keywords that precisely especify what data structures, algorithms, proof techniques, mathematical insights and input/ouput style were used in the program. The early project decision was to append to the solution files a commented header with the problem's metadata and tags — an example of a problem header in a `C++` solution is found in Figure 1: Example



```
    max n    worst class    class type                 example
0      11  O(n!), O(n^6)   exponential            permutations
1      18      O(2^n n^2)                               DP TSP
2      22        O(2^n n)                       DP with bitmask
3     100          O(n^4)      quartic    DP with 3 dimentions
4     400          O(n^3)        cubic          Floyd Warshall
5      2K     O(n^2 lg n)              2-nested loops + tree
6     10K          O(n^2)    quadratic          Insertion sort
7      1M       O(n lg n) linearithmic     Merge sort, segtree
8    100M            O(n)       linear     Merge sort, segtree
9                 O(lg n)  logarithmic           binary search
10                   O(1)     constant             hash lookup
```

Figure 2: Output of timetable



```
"""
 -  test:
    input:      |
                3
                5 2
                8 5
                2 2
    output:     |
                1
                4
                0
 -  test:
    input:      !file "{dirabs}/HR19_30DOC_D29.2.in"
    output:     !file "{dirabs}/HR19_30DOC_D29.2.out"
 -  test:
    input:      !file "{dirabs}/HR19_30DOC_D29.5.in"
    output:     !file "{dirabs}/HR19_30DOC_D29.5.out"
"""
```

Figure 3: Example of a problem footer

of a problem header. Meant for not for better organization and documentation of the solutions files, but also for automatedly parsing their metadata, this approach would allow to programmatically query and interpret them. With that in mind, and for the sake of simplicity, the solutions' metadata are contained in block comments and must be in valid YAML format.

At the begining, a simple command line tool capable of retriving the available solutions that better reseamble a list o query tags have been programmed: it simply recursively searches through the directory tree for solutions, parsing their metadata. For personal convinence and productivity, the tool was slightly incremented throughout the semester, starting with other lookup capapabilities like searching through archived algortihms and data structures implementations or showing an input versus time, input versus memory and datatype ranges tables — e.g. Figure 2: Output of timetable. Finally, a last feature turned the tool into a offline, local judgement system.

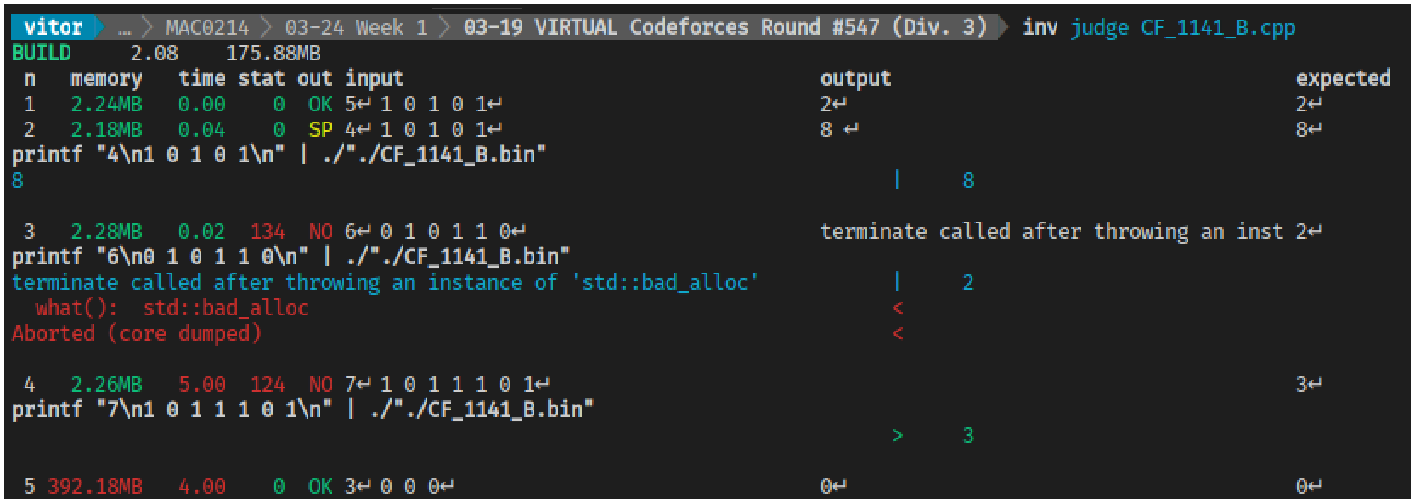
Given a solution file, the **local judge** tool scans through its tags for compilation, exacution and testing metatags, with default fallback options. For convinence, the testing metatags are placed at the end of the file, like in Figure 3: Example of a problem footer. It is possible to clearly and simply specify any number of independent test cases for the solution, optinally setting the maximum time and memory available, the input, the expected output or the required exit status. Because of the flexibility of the YAML format, those values can be literal placed, read from a file or from a program pipe.

Once executing on shell prompt the command `inv judge <solution_file>`, the tool will first try to compile the solution, if applicable, showing whether it was successful and the amount of memory and time used. Then, for every test case registered per line, the amount of memory and time consumed, the exit status, the standard input given, the standard ouput received and the expected output are showed, warning whether any of the resource or termination constraints failed. In particular, when the actual output differs from the expected, their GNU diff is also showed. An ilustration of this process' result can be seen in Figure 4: Example of a problem judgement by local judge:

1. a successful, correct answer,
2. an answer that differs only by space characters,
3. a runtime error instance,
4. an instance that slightly surpasses the time limit,
5. an instance that slightly surpasses the memory limit.

At the half semester mark and concerned about the lack of variety of the solutions, the references "Guide to Competitive Programming: Learning and Improving Algorithms Through Contests" [AL] and "CS 97SI: Introduction to Programming Contests" [CS97SI] were review and added o this project's bibliography. The ICPC-like problems (therefore most of Codeforces' problems) not only have exceedingly elusive problem statements, but also rather focuses dynamic programming, greedy optimal choice and mathematical properties over algorithms and data structures, as the Association for Computing Machinery — ICPC's host — favor theoretical insights over implementation in its contests [AL].

Codeforces' rating ranges from 500 to 3800, whereas CF_325_B (Problem 1: CF_325_B - Stadium and Games, a median problem) is e median problem, rated 1800 and correctly solved by 309 of the 1181 participants of its contests. For comparison, CF_977_A, one of the 500-rated problems and solved by 2581 of 3555 contestants is an



```
vitor  … > MAC0214 > 03-24 Week 1 > 03-19 VIRTUAL Codeforces Round #547 (Div. 3) > inv judge CF_1141_B.cpp
BUILD     2.08    175.88MB
 n   memory   time stat out input                                                 output                                  expected
 1   2.24MB   0.00    0  OK 5↵ 1 0 1 0 1↵                                         2↵                                      2↵
 2   2.18MB   0.04    0  SP 4↵ 1 0 1 0 1↵                                         8 ↵                                     8↵
printf "4\n1 0 1 0 1\n" | ./"./CF_1141_B.bin"
8
                                                                                        |     8

 3   2.28MB   0.02  134  NO 6↵ 0 1 0 1 1 0↵                                       terminate called after throwing an inst 2↵
printf "6\n0 1 0 1 1 0\n" | ./"./CF_1141_B.bin"
terminate called after throwing an instance of 'std::bad_alloc'                         |     2
  what():  std::bad_alloc                                                               <
Aborted (core dumped)                                                                   <

 4   2.26MB   5.00  124  NO 7↵ 1 0 1 1 1 0 1↵                                                                             3↵
printf "7\n1 0 1 1 1 0 1\n" | ./"./CF_1141_B.bin"
                                                                                        >     3

 5 392.18MB   4.00    0  OK 3↵ 0 0 0↵                                             0↵                                      0↵
```

Figure 4: Example of a problem judgement by local judge

"easy" problem; and CF_717_B, arguably on of the hardest, was solved only by 2 of the 819 at the time. As a consequence of Codeforces' non-linear rating formula, the problems that require implementing interesting algorithms or data structures are overshadowed tricky dynamic programming and theoretical results.

Regarding more implementation variety, a last type of problem solving activity was considered: **problem mock practice** of **interview-like problems** from Sphere online judge, HackerRank and LeetCode. In comparison to ICPC-like problems, their statements are more clear, therefore quickening its interpretation and implementation. As an example, both CF_1146_B and SPOJ_CL_GSS1 solutions rely on Segment trees and range queries [CP], however the latter diminishes this data structure with a tricky proof of finding a concatanation invariant. In these activities, a solution is first sketched and compared to the editorial, then it was implemented only if it relies on some algorithm, data structure or technique not tagged before.

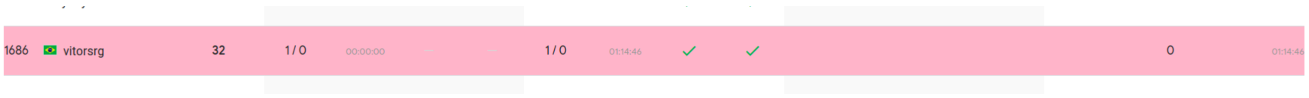
Table 1: Detailed schedule also logs non-problem solving activities like MaratonUSP's meetings and the confection of this project's reports, although the most interesting of them are MaratonUSP's lectures. As I have been preparing myself to lecture on MaratonUSP's classes, I was responsible for filming — alongside Eduardo Freire — and publishing two of them: BixeCamp Lecture 3 - Sorting, binary search and complexity and Prepare equipment, film and publish BixeCamp Lecture 9 - Segment tree.

STANDINGS

The key distinction between the tagging system used on Table 2: Problems reference table and the tags already available on some of the online judge platforms is that the former tags are meant to be extensively and meticulously descriptive about not only the algorithms and data structures used, but also insights and proof techniques related to the particular problem solution. Still, the tags' title follow standard terminology from the references (CRLS, SW, CP and CS97SI) and from the online judge platforms' forums (CF and TC), therefore providing a neat and welcoming environment for newcomers to competitive programing to learn its specificities.

Nonetheless, the tags may *spoil* the approach to solve the problem by one's own: what the online platforms are not interested in; so the table may not suite all practioners. For instance, Codeforces usually only tag a algorithm or data structure when it has to be tweak in the solution — e.g. CF_154_B's tags completely ignores the required methods for its solution. As a consequence of how the tags of this project were chosen, every solution has an expected amount of seven tags, as well as there is a probaility over than 80% that a given combination of `<tag, problem>` listed on Table 2 isn't assigned on the problem's platform. That is, 80% of the links on Problems reference table define a problem-tag relation that isn't assigned in the original platform.

Through the Codeforces' problems, I could practice



1686	vitorsrg	32	1 / 0	00:00:00	—	—	1 / 0	01:14:46	✓	✓	0	01:14:46

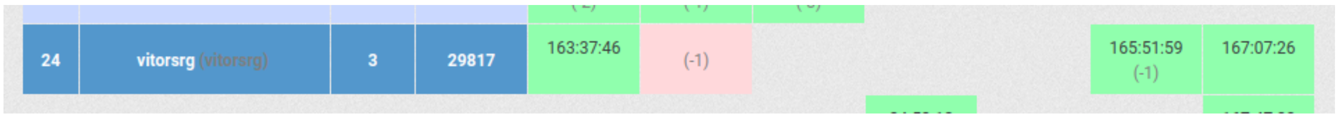
Figure 5: My final standing on GCJ19_1C - Google Code Jam 2019 Round 1C

some algorithms and data structures natural to competitive programming [AL, CS97SI], but unfamiliar to traditional courses and textbooks [CRLS, SW], most notably Segment Tree and Seive of Eratosthenes (and its variations). However, those problems actually improved my mathematical proofing and algorithm analysing skills, as they are always based on finding invariants or optimal substructures, using well-known algorithms as a tool sometimes, — particularly, I found CF_101_C as one of the best representants for a usual Codeforces' problem. In fact, those tree- or table-like algorithms are just recurrent forms of usual **dynamic programming** problems — which, alongside **greedy** arguments from Sieve of Eratosthenes and exchange argument for sorting, are respectively the data structures and algorithms classified as classical in competitive programming —, relying on efficiently querying or updating associative ordered sets and processing factors of the Integers' multiplicative ring.

Though effectively competing was not the purpose of this project, I have managed to achieve two prominent standings. First, I was positioned on the 1686th place on Google Code Jam's Round 1C: from all 27500 contestants qualified for on Round 1, only 4500 would be selectioned to progress to Round 2 (the 1500 best from each rounds 1A, 1B and 1C). As there is just a small time penalty between me and the 1500th place, I consider this a great achievement as this contest is both public and worldwide, being organized by former winners from ACM-ICPC. The Figure 5: My final standing on GCJ19_1C - Google Code Jam 2019 Round 1C contains an screenshot from the publicly available Round 1C standings. Second, I've performed as 24th on MaratonUSP's Seletiva Individual 2019: Primeira prova, correctly solving three problems as all the contestants ranked from 11th to 21th. On both cases, I wasn't available for the beginning of the contests.

From all problems, I found both the ones from Google's contests and the interview-like problems as the most interesting to solve. They are not only more clear and direct, but also based on key algorithm design insights. Because of this, it is straightforward to search for similar solutions (e.g. online), therefore they wouldn't be considered more than median problems in the whole competitive programming, as the theoretical, mathematical component present on more advanced problems allows them to blend evenly: two similar problems may have far distinct problem statements, even after abstracting the fantasy story. In particular, Google's competitions acknowledges this, challenging its contestants on a wider variety of problem solving approaches and languages — for example, "Many problems cannot be approached with random or brute-force solutions, but identifying the problems that can be (in Code Jam or the real world) is a useful skill!", from GCJ19_1A_A - Pylons' editorial. Arguably, a strong evidence of this inclination is that many of my solutions take a more practical, ad-hoc approach when compared to the proposed editorial solutions: I take VJ_299509_G as a great example; my solution with `value-iteration` is far simpler, shorter and faster than my implementation of the editorial's proposed solution based on `dp-graph`.

Overall, more than 300 problems were studied, from which around 230 had a sketched solution, with about 70 working solutions. The problems tags and their contests, the tools and reports produced during this project are publicly available on <github.com/vitorsrg/MAC0214>. Concerning some of the platforms' Terms of Usage, that discourages publishing solutions as it tempers their rating system, some of the solutions files were blanked, consisting only of the problems' header and footer. Then, for better organization, multiple solutions of the same problem or comprised in a single file, with multiple lines of tags.



24	vitorsrg (vitorsrg)	3	29817	163:37:46	(-1)			165:51:59 (-1)	167:07:26

Figure 6: My final standing on VJ_299509 - Seletiva Individual 2019: Primeira prova

To make up for this, I've compiled algorithms implementations, code snippets and ad-hoc tricks that helped me on solving any problem into six categories. Most of the resources come from the reference books CRLS, SW and AL, from the great forums on Codeforces and Topcoder, or from the invaluable competitive programming algorithm base CP-Algorithms. The six groups — inspired by how the CSref:CS97SI lectures are organized — are: Numbers, Trees & Grouping, Ordering, Graphs, Dynamic Programming and Strings. As I couldn't find any succinct and complete implementation of arbitrary-precision integers for `C++`, `Python 3.5+` stands as an alternative for the rare problems which require BigInterger, Decimal or Fractions data types. For the first time, in this semester the algorithms discussed on MaratonUSP's lectures have been published on the Caderno dos Bixes repository, and the six compilations shall be further organized there, as a reference for all newcomers to the extracurricular group and everyone else.

In conclusion, the required hundred hours dedicated to the project were successfully achieved with problem solving and related tasks, conservatively discriminating other activities as an additional workload, as illustrated by Table 1: Detailed schedule. I consider that I have met the initial projects goals: I have studied and practiced a broad variety of algorithms and data structures through a diverse flavor os contests and platforms, as thoroughly organized on the Table 2: Problems reference table. However, those deliverables are no match to great open source, community-driven projects like (i) Awesome Competitive Programming tutorials, (ii) A^2 Online Judge ranking and tagging, and (iii) E-Maxx Algorithms in English's source and explanation of most of the competitive programming algorithms.

Arguably, the most valuable deliverable of the project is the local judge — a simple tool is based on how I have always tested my programming assignments. As it was meant for personal use, its development was not tracked in the workload, as well as its source code isn't fully documented. Nonetheless, it might be a true original project: because in-person contests offer a very limited programming environment, competitive programmers must learn to debug their code by intuition, one of the greatest barriers for newcomers to the field. Both a prior `Ruby` and a later `Python` of the local judge are available in this project's repository. As further work, I plan to restructure the project into a programming assignment tester and grader of IME's courses.

REFERENCES

[AL] LAAKSONEN, Antti (2017). **Guide to Competitive Programming: Learning and Improving Algorithms Through Contests**. Helsinki: Springer.

[SW] SEDGEWICK, Robert; WAYNE, Kevin (2011). **Introduction to Algorithms**. Fourth edition. Boston: Pearson Education.

[CF] **Codeforces**. Available at <codeforces.com>.

[SPOJ] **Sphere online judge**. Available at <www.spoj.com>.

[CS97SI] Park, Jaehyun (2015). **CS 97SI: Introduction to Programming Contests**. Available at <web.stanford.edu/class/cs97si/>.

[CP] **CP-Algorithms**. Available at <cp-algorithms.com>.

[TC] **Topcoder**. Available at <www.topcoder.com/>.

[CRLS] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford (2009). **Algorithms**. Third edition. Cambridge: MIT Press.

[HR] **HackerRank**. Available at <www.hackerrank.com>.

[LC] **LeetCode**. Available at <leetcode.com>.

[GKS] **Google Kick Start**. Available at <codingcompetitions.withgoogle.com/kickstart>.

[GCJ] **Google Code Jam**. Available at <codingcompetitions.withgoogle.com/codejam>.

[FBHC] **Facebook Hacker Cup**. Available at <www.facebook.com/hackercup/contest>.

[IOIS19] **Facebook Hacker Cup**. Available at <people.ksp.sk/ misof/ioi-syllabus/ioi-syllabus-2019.pdf>.

[LT50CIQ] LEETCODE (2014). **LeetCode Clean Code Handbook: 50 Common Interview Questions**. First edition. LeetCode.

ATTACHMENT 1: PROJECT PROPOSAL

MAC0214 - Extracurricular Activities

15/3 Project proposal

Professors André Fujita[1], Denis Deratani Mauá[2], Kelly Rosa Braghetto[3]

First Semester of 2019

Undergraduate Vitor Santa Rosa Gomes[4]

Supervisor Nathan Benedetto Proença[5]

OBJECTIVE

Last semester, I was sincerely amused to discover the Fenwick Tree data structure, which can speed up a $\mathcal{O}(n)$ linear probing to $\mathcal{O}(\log n)$. Although representing only a modest improvement to a greedy or dynamic programming algorithm, this simple and easy to implement detail makes up an impressive practical gain.

With that in mind, the purpose of this project is to learn algorithms and data structures not covered by MAC0121 - Algorithms and Data Structures I, MAC0323 - Algorithms and Data Structures II, MAC0338 - Analysis of Algorithms and other mandatory courses in the Bachelor of Computer Science at IME-USP.

METHODOLOGY

The competitive programing problems offer a neat environment to both development algorithm design and enrich an undergraduate CS student's data structures tool belt. Every problem solved during the semester will be tagged accordingly with its strategies, e.g. in Table 1. This compiled problem sheet will offer to both myself and my colegues not only a fast look up for algorithms and data structures, but also a list of use cases to understand and practice them.

Since the project focuses on algorithmic problem solving, usually from Codeforces, SPOJ, URI, HackerRank, CodeChef and MaratonUSP lists, [CRLS] and [SW] are the base reference books. Nonetheless, more specific texts may be added for more advanced formulations, such as number theory or discrete computacional geometry.

SCHEDULE

As a very conservative approach to fulfill the required hundred hours, from March 15 to June 30, Table 2 contains a detailed schedule of problems carefully chosen, considering their contexts and strategies. Ranging from lower- to upper-intermediate difficulty, it is safe to state that each problem of the former will take at least 20 minutes to solve, whilst each of the latter problems will consume 30 minutes.

No.	Tags	Description	Problems
1	`#binary_search`	Find the position of a value in a sorted array	### ### ### ...
2	`#fenwick_tree`, `#binary_indexed_tree`	Balance element update and prefix sum in arrays	### ### ### ...
2	`#shortest_paths`, `#shortest_paths_from_source`, `#dijkstra`	Find the shortest paths from a source to all vertices in a graph	### ### ### ...

Table 1: Example of the compiled problem sheet

[1] fujita@ime.usp.br
[2] www.ime.usp.br/~ddm/
[3] www.ime.usp.br/~kellyrb/
[4] NUSP 10258862, vitorssrg@usp.br
[5] nathan@ime.usp.br

2

No.	Week	Activities	Workload
-	11/3 – 17/3	Write project proposal	-
1st	18/3 – 24/3	Problems 1009C 1006D 960C 631C 935C 540B 1131F 501C 850A 1000C 507C 616D 682C 255C 1132C 493C 269A 1073D 923A 337C 766C	8h
2nd	25/3 – 31/3	Problems 203D 98A 1044A 1057A 596C 159A 68B 238A 22C 46C 161B 1138B 756B 958F1 117B 409A 630H 319A 314A 34D 813B	8h
3rd	1/4 – 7/4	Problems 364A 411A 200C 202B 158B 527C 407B 585B 207B1 1073C 30B 675D	4h
		Google Code Jam Qualification Round	3h
4th	1/4 – 7/4	Problems 301A 1081D 830B 513G1 928B 436C 845D 1082D 1042D 930B 452B 223A 1102E 888E 822D 232A 769D 792D 1076D 549D 637C	7h
		Google Code Jam Round 1A	?
5th	8/4 – 14/4	Problems 727D 291A 67B 690B1 290C 48C 67A 459C 207A2 234G 37B 130E 926D 56C 180A 76D 85B 216D 162B 79C 295B	7h
6th	15/4 – 21/4	Problems 472D 292E 961E 656A 1054D 566D 95C 999D 19B 294C 196B 490D 770C 847C 1012B 1000D 652D 722D 500D 982D 615D	7h
		Google Code Jam Round 1B	?
7th	22/4 – 28/4	Problems 470E 630I 491B 37C 101B 69C 846C 178B3 43D 457B 269B 120H 309C 291B 176B 126A 257D 82C 29D 72E 661B	7h
		Google Code Jam Round 1C	?
8th	29/4 – 5/5	Problems 582B 117C 1088D 598E 792C 959D 766D 739B 767C 940E 301B 777E 749D 27C 814D 778B 843B 864E 1133E 476D 721D	7h
9th	6/5 – 12/5	Problems 896B 784E 1067B 855C 960F 893E 119C 266C 245F 263C 14E 1080D 180D 975D 1089F 15C 811D 362C 847E 121C 164A	7h
		Google Code Jam Round 2	?
10th	13/5 – 19/5	Problems 191C 132B 100G 345B 235B 343C 337D 840B 527D 837D 590B 720A 963B 20C 125D 514D 1030E 621E 510D 1010D	7h
11th	20/5 – 26/5	Problems 784G 638C 960D 955C 1029F 56D 313D 1083B 630E 645E 1056E 909E 909D 899E 156C 65C 226D 817E	7h
12th	27/5 – 2/6	Problems 386C 464B 1110D 7D 207D3 599E 1002D1 39E 11D 72F 132D 656E 819A 86B 592D 551C	7h
		Google Code Jam Round 3	?
13th	3/6 – 9/6	Problems 500E 620D 538F 793C 217B 908D 1003E	4h
		Facebook Hacker Cup Qualification	3h
14th	10/6 – 16/6	431D 995C 853C 232B 404D 1065D 1001A 356C 821E 615E 723F 809B 1044C 238C	7h
		Facebook Hacker Cup Round 1	?
-	24/6 – 30/6	Write final report	-
	Total		100h

Table 2: Detailed schedule

Those problems might be anticipated or changed by similar ones (from any of the platforms listed earlier), as not only it is desired to the broaden complied problem sheet's variety, but also I'll take part on weekly contests whenever I'm both qualified and available, as well as simulate past ones virtually. In practice, I plan to devote these credits on both more advanced contests and lecturing for MaratonUSP, however only the problems that I will actually be able to solve will are being accounted, as a safe ground for the required hours.

Moreover, two great contests are going to take place soon: Google's Code Jam 2019 and Facebook's Hacker Cup 2019. Only their classifications rounds were taken into account, as it is not guaranteed that I will be competing through the following rounds.

FOLLOW-UP

Every code and report made for this project will be weekly available at the github.com/VitorSRG/MAC0214 git repository. There will also be scanned versions of the weekly schedule signed by the supervisor in up to three weeks.

REFERENCES

[CRLS] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford (2009). **Algorithms**. Third edition. Cambridge: MIT Press.

[SW] SEDGEWICK, Robert; WAYNE, Kevin (2011). **Introduction to Algorithms**. Fourth edition. Boston: Pearson Education.

ATTACHMENT 2: PROJECT SCHEDULE AND WORKLOAD

No.	Week	Date	Activities	Problem solving	Other activities
1st	11/3 – 17/3	17/3	Plan and Write Project Proposal		5h
2nd	18/3 – 24/3	19/3	Virtual simulate and sketch upsolve of CF_1141 - Codeforces Round #547 (Div. 3)	3h	
		20/3	Solve 10 problems from HR_EULER - ProjectEuler+ on HackerRank	4h30	
		22/3	Virtual simulate and sketch upsolve of GKS19_PR - Google Code Kick Start Practice Round	5h	
		23/3	Virtual simulate and sketch upsolve of CF_1140 - Educational Codeforces Round 62 (Rated for Div. 2)	3h	
		24/3	Live compete in and sketch upsolve of GKS19_A - Google Code Kick Start Round A	4h30	
3rd	25/3 – 31/3	29/3	Prepare equipment and film MUSP_191_L3 MaratonUSP's BixeCamp Lecture 3 - Sorting, binary search and complexity		3h
		31/3	Virtual simulate and sketch upsolve of CF_1144 - Codeforces Round #550 (Div. 3)	2h30	
4th	1/4 – 7/4	2/4	Virtual simulate and sketch upsolve of CF_1145 - April Fools Day Contest 2019	3h	
		6/4	Live compete in and sketch upsolve of GCJ19_Q - Google Code Jam Qualification Round	5h	
5th	8/4 – 14/4	13/4	Live compete in and sketch upsolve of GCJ19_1A - Google Code Jam Round 1A	3h30	
		14/4	Quick review of Guide to Competitive Programming: Learning and Improving Algorithms Through Contests [AL]		3h
6th	15/4 – 21/4	16/4	Virtual simulate and sketch upsolve of CF_1154 - Codeforces Round #552 (Div. 3)	3h	
		20/4	Virtual simulate and sketch upsolve of CF_1146 - Forethought Future Cup - Elimination Round	3h	
		21/4	Live compete in and sketch upsolve of GKS19_B - Google Code Kick Start Round B	4h30	
7th	22/4 – 28/4	28/4	Live compete in and sketch upsolve of GCJ19_1B - Google Code Jam Round 1B	3h30	
8th	29/4 – 5/5	1/5	Live compete in and sketch upsolve of CF_1156 - Educational Codeforces Round 64 (Rated for Div. 2)	3h30	
		4/5	Live compete in and sketch upsolve of GCJ19_1C - Google Code Jam Round 1C	3h30	
9th	6/5 – 12/5	10/5	Live compete in and sketch upsolve of MaratonUSP's VJ_299509 - Seletiva Individual 2019: Primeira prova	4h	
		11/5	Live compete in VTEXCC19_1 - VTEX Code Cup Round 1	4h	
10th	13/5 – 19/5	17/5	Review lecture presentations and sketch upsolve of selected problems from CS 97SI: Introduction to Programming Contests [CS97SI]	1h	4h30
		18/5	Virtual simulate and sketch upsolve of GCJ19_2 - Google Code Jam Round 2	3h	
11th	20/5 – 26/5	20/5	Tag and sketch solve classical problems from HR_30DOC19 - HackerRank's 30 Days of Code	6h	
		23/5	Tag and sketch solve classical problems from SPOJ_CL - Sphere online judge's list of classical problems	9h	
		24/5	Tag and sketch solve classical problems from LC_TIQ - LeetCode's Top Interview Questions	7h	
		26/5	Live compete in and sketch upsolve of GKS19_C - Google Code Kick Start Round C	2h30	
12th	27/5 – 2/6	31/5	Prepare equipment, film and publish MUSP_191_L9 MaratonUSP's BixeCamp Lecture 9 - Segment tree		2h30
		2/6	Sketch upsolve of MaratonUSP's VJ_304339 - [Upsolving] Seletiva Individual 2019: Segunda prova	3h	
13th	3/6 – 9/6	9/6	Sketch upsolve of GCJ19_3 - Google Code Jam Round 3	2h	
14th	10/6 – 16/6	15/6	Live compete in and sketch upsolve of FBHC19_Q - Facebook Hacker Cup 2019 Qualification Round	4h	
15th	17/6 – 23/6	20/6	Solve FBCR17_1 - Facebook Careers Sample Interview Questions	1h30	
		22/6	Compile final report		6h
16th	24/6 – 30/6	27/6	Make project poster		4h
		28/6	Present project poster		3h
		29/6	Sketch solve of FBHC19_R1 - Facebook Hacker Cup 2019 Round 1	2h	
		30/6	Review and small amendments to the final report		0h30
	Total			104h	31h30

Table 1: Detailed schedule

ATTACHMENT 3: PROBLEMS REFERENCE TABLE

Tags	Problems
and	HR_30DOC_D29
array	CF_1141_B HR_30DOC_D07 HR_30DOC_D11 HR_30DOC_D21
avg	CF_1154_B SPOJ_CL_02123
backtracking brute-force	GCJ19_Q_C GCJ19_1A_A GCJ19_1A_A
bfs	CF_1146_C SPOJ_CL_01437
biginteger	GCJ19_Q_C SPOJ_CL_00005 SPOJ_CL_00024 SPOJ_CL_00054 SPOJ_CL_00328
binary	CF_1146_C HR_30DOC_D10
binary-tree	CF_1140_A HR_30DOC_D22 HR_30DOC_D23
bipartile	SPOJ_CL_01025 SPOJ_CL_03377
bit bitree fenwick-tree	SPOJ_CL_07742 CF_1146_B GKS19_B_A GKS19_B_C GCJ19_1B_A
bitwise	HR_30DOC_D29
bfs breath-first-sea	VJ_304339_D HR_30DOC_D23
bsearch binary-search	GKS19_PR_A GCJ19_1B_C SPOJ_CL_00297 GKS19_C_B
bsearch-leftmost	SPOJ_CL_00297
bubblesort	HR_30DOC_D20
cases conditionals	CF_1156_A HR_30DOC_D03 HR_30DOC_D26 SPOJ_CL_02157
chinese-remainde	GCJ19_1A_B
coloring	SPOJ_CL_03377
combinatorics	GCJ19_2_A SPOJ_CL_01724 SPOJ_CL_03410
comparator	GKS19_B_B VTEXCC19_1_G SPOJ_CL_02727
connectivity graph-connectivi dynamic-connecti	VJ_299509_B
count counting	CF_1156_B SPOJ_CL_01724 SPOJ_CL_02123 SPOJ_CL_03410 SPOJ_CL_04300 SPOJ_CL_06256 GKS19_B_A CF_1146_B
cripto cryptography	GCJ19_Q_C SPOJ_CL_00400
cycle graph-cycle sequence-cycle	CF_1141_B VTEXCC19_1_C
dfs depth-first-sear	VJ_299509_F VTEXCC19_1_C SPOJ_CL_01436 SPOJ_CL_03377
diameter graph-diameter	CF_1146_C SPOJ_CL_01437
diff	GKS19_B_C
digits	HR_EULER_008 GCJ19_Q_A CF_1146_C
divide-and-conqu	CF_1145_A
division divisor divisors	SPOJ_CL_02148 HR_30DOC_D19 SPOJ_CL_04300
dp dynamic-programm	HR_EULER_002 HR_EULER_009 HR_EULER_010 CF_1145_A GCJ19_1A_C CF_1154_F CF_1154_G GKS19_B_B GCJ19_1B_A VJ_299509_F VJ_299509_G VTEXCC19_1_C SPOJ_CL_00346 SPOJ_CL_00394 SPOJ_CL_00902 SPOJ_CL_01021 SPOJ_CL_03923 SPOJ_CL_04141 SPOJ_CL_06219 SPOJ_CL_14930 FBHC19_Q_A FBHC19_Q_B GKS19_C_B
dp-graph	VTEXCC19_1_C
dp-knapsack boolean-knapsack	CF_1154_F GKS19_B_B
dp-linear	HR_EULER_002 HR_EULER_010 SPOJ_CL_00346 SPOJ_CL_00394 SPOJ_CL_00902 SPOJ_CL_14930
dp-linearithmic	CF_1145_A VJ_299509_G
dp-max	SPOJ_CL_03923 SPOJ_CL_14930
dp-quadratic	HR_EULER_009 SPOJ_CL_01021 SPOJ_CL_06219 FBHC19_Q_A FBHC19_Q_B GKS19_C_B
dp-tree	GCJ19_1A_C VJ_299509_F
edit-distance	FBCR17_1_Q3 SPOJ_CL_06219
euler-totient	VJ_299509_G
events	GKS19_B_C VTEXCC19_1_G
exchange-argumen	SPOJ_CL_03375
expression	SPOJ_CL_00004
factorial	GCJ19_1C_B GCJ19_2_A HR_30DOC_D09 SPOJ_CL_00011 SPOJ_CL_00024
factors	CF_1141_A HR_EULER_003 HR_EULER_005 GCJ19_Q_C SPOJ_CL_00011 SPOJ_CL_04300 SPOJ_CL_09948
float	VJ_299509_F SPOJ_CL_00902 SPOJ_CL_04408 VJ_304339_A
ford-fulkerson maxflow mincut	SPOJ_CL_01025
formula	HR_EULER_006 SPOJ_CL_00302 SPOJ_CL_00328 SPOJ_CL_01030 SPOJ_CL_01724 SPOJ_CL_03410 SPOJ_CL_04408 SPOJ_CL_07406 SPOJ_CL_07424 SPOJ_CL_07974 SPOJ_CL_09948 SPOJ_CL_10509 SPOJ_CL_11063
fraction	GCJ19_2_A GCJ19_2_D
fraction-knapsac	GKS19_B_B
games nim grundy	SPOJ_CL_01419 SPOJ_CL_01419
gcd greatest-common-	CF_1141_A HR_EULER_005 GCJ19_Q_C VJ_299509_G
geometry	CF_1156_A VJ_304339_A
graph	CF_1146_C VJ_299509_B VJ_299509_F VTEXCC19_1_C SPOJ_CL_01025 SPOJ_CL_01436 SPOJ_CL_01437 SPOJ_CL_03377 VJ_304339_D
greedy	CF_1140_B CF_1144_B GCJ19_Q_A GCJ19_Q_B CF_1154_D CF_1154_F CF_1146_A GKS19_B_B CF_1156_B VTEXCC19_1_A VTEXCC19_1_G GCJ19_2_A GCJ19_2_D SPOJ_CL_00297 SPOJ_CL_03375 GKS19_C_A FBHC19_Q_A FBHC19_Q_B FBCR17_1_Q3
grid	GCJ19_Q_B GCJ19_1A_A GCJ19_1B_A VTEXCC19_1_C SPOJ_CL_00302 SPOJ_CL_03410 SPOJ_CL_03923 GKS19_C_A GKS19_C_B FBCR17_1_Q1
hamiltonian graph-hamiltonia	GCJ19_1A_A
hash	GCJ19_1B_A
implementation	CF_1141_A CF_1141_B CF_1144_A CF_1144_B CF_1145_B CF_1145_C CF_1154_A CF_1154_B CF_1154_C CF_1154_D CF_1146_A CF_1146_B CF_1156_A CF_1156_B VJ_299509_A VTEXCC19_1_A HR_30DOC_D15 HR_30DOC_D00 HR_30DOC_D01 HR_30DOC_D02 HR_30DOC_D03 HR_30DOC_D04 HR_30DOC_D05 HR_30DOC_D06 HR_30DOC_D07 HR_30DOC_D08 HR_30DOC_D09 HR_30DOC_D10 HR_30DOC_D11 HR_30DOC_D12 HR_30DOC_D13 HR_30DOC_D14 HR_30DOC_D16 HR_30DOC_D17 HR_30DOC_D18 HR_30DOC_D19 HR_30DOC_D20 HR_30DOC_D26 HR_30DOC_D27 SPOJ_CL_02727 SPOJ_CL_00001 SPOJ_CL_00379 SPOJ_CL_00400 SPOJ_CL_01112 SPOJ_CL_02148 SPOJ_CL_02157 SPOJ_CL_03410 VJ_304339_A FBHC19_Q_A FBHC19_Q_B FBCR17_1_Q1 FBCR17_1_Q2 FBCR17_1_Q3 HR_EULER_001
integer-optimiza	GCJ19_2_A GCJ19_2_D
interactive	GKS19_PR_A CF_1146_C GCJ19_1B_B GCJ19_1C_B
invertion invertions	HR_30DOC_D20 SPOJ_CL_06256

Tags	Problems
ipqueue index-priority-q	GCJ19_1A_C
linear-recurrence	HR_EULER_002
linked-list	HR_30DOC_D15 HR_30DOC_D24
log	SPOJ_CL_09948
longest-path	VJ_304339_D
loop	HR_30DOC_D05 HR_30DOC_D10
math	CF_1141_A CF_1141_C HR_EULER_001 CF_1154_A CF_1154_B CF_1154_C GCJ19_1B_B SPOJ_CL_00302 SPOJ_CL_00328 SPOJ_CL_04408 SPOJ_CL_07406 SPOJ_CL_07424 SPOJ_CL_07733 SPOJ_CL_09948 SPOJ_CL_10509 SPOJ_CL_11063
max	SPOJ_CL_01043 GKS19_C_B
maximum-weighted	SPOJ_CL_01025
memoization	HR_EULER_003 SPOJ_CL_07733
memory-managemen	SPOJ_CL_00002 SPOJ_CL_00346
mergesort	SPOJ_CL_06256
min	CF_1144_B GCJ19_1B_C GKS19_C_B
modpow	SPOJ_CL_03442
modulo	CF_1154_C GCJ19_1B_B SPOJ_CL_01419 SPOJ_CL_02148 SPOJ_CL_10509
monte-carlo	GCJ19_1A_A GCJ19_1A_B
numbers	CF_1141_A HR_EULER_001 GCJ19_Q_A SPOJ_CL_00011 SPOJ_CL_00042 SPOJ_CL_01030 SPOJ_CL_02157 SPOJ_CL_03442 SPOJ_CL_09948 FBCR17_1_Q2
object	HR_30DOC_D15 HR_30DOC_D04 HR_30DOC_D12 HR_30DOC_D13 HR_30DOC_D14 HR_30DOC_D17 HR_30DOC_D18 HR_30DOC_D19
offline	HR_30DOC_D07
online	HR_EULER_008 CF_1140_B CF_1144_A GCJ19_Q_A GCJ19_Q_B CF_1154_B CF_1154_D CF_1146_A CF_1146_A CF_1156_A VJ_299509_A VJ_299509_B HR_30DOC_D08 HR_30DOC_D25 HR_30DOC_D29 SPOJ_CL_00001 SPOJ_CL_03410 SPOJ_CL_07742 SPOJ_CL_07974 SPOJ_CL_08002 GKS19_C_A
operation	VJ_299509_B VJ_299509_F
optimization	GCJ19_2_A GCJ19_2_D
order	CF_1145_A
overflow	SPOJ_CL_02148
palindrome	GKS19_B_A HR_30DOC_D18 SPOJ_CL_00005 SPOJ_CL_01021
parse	HR_30DOC_D16 SPOJ_CL_00004
permutation	SPOJ_CL_00379 GCJ19_2_A
pow	SPOJ_CL_03442 HR_30DOC_D17
pqueue priority-queue	VTEXCC19_1_G
preffix	GCJ19_1A_C
preprocess	HR_EULER_003 GKS19_B_A VJ_299509_G FBHC19_Q_A FBHC19_Q_B
primes	HR_EULER_003 HR_EULER_005 HR_EULER_007 HR_EULER_010 GCJ19_Q_C GCJ19_1A_B HR_30DOC_D25 SPOJ_CL_00002
product	HR_EULER_008
quadratic	GKS19_C_A GKS19_C_B
query	HR_EULER_002 HR_EULER_003 HR_EULER_005 HR_EULER_006 HR_EULER_007 HR_EULER_009 HR_EULER_010 GKS19_B_A GCJ19_1B_B GCJ19_1C_B VJ_299509_B VJ_299509_F VJ_299509_G HR_30DOC_D08 HR_30DOC_D25 HR_30DOC_D29 SPOJ_CL_01043 SPOJ_CL_04141 SPOJ_CL_08002
queue	HR_30DOC_D18
random las-vegas	GCJ19_1A_A GCJ19_1A_B
range	GCJ19_1B_C SPOJ_CL_00002 SPOJ_CL_01043 SPOJ_CL_08002 GKS19_C_B
recurrence	HR_EULER_002 SPOJ_CL_07406 SPOJ_CL_07733
recursion	HR_30DOC_D22 HR_30DOC_D24 SPOJ_CL_00011 SPOJ_CL_00024 FBCR17_1_Q1 GCJ19_1C_B SPOJ_CL_07733 SPOJ_CL_09948
regex	HR_30DOC_D28
reverse	SPOJ_CL_00042
segsieve segmented-sieve	SPOJ_CL_00002 HR_EULER_003
segtree segment-tree	CF_1146_B GKS19_B_A GKS19_B_C GCJ19_1B_A GCJ19_1B_C SPOJ_CL_01043 SPOJ_CL_08002 GKS19_C_B
set tree-set	CF_1140_A
sieve sieve of eratosthenes	HR_EULER_007 HR_EULER_005 HR_EULER_010 CF_1154_G
simulation	GCJ19_1A_C GKS19_C_A CF_1140_A GCJ19_1A_A CF_1154_D VTEXCC19_1_G FBHC19_Q_A FBHC19_Q_B FBCR17_1_Q1 FBCR17_1_Q2
solve equantion fixed-solution	CF_1141_C HR_EULER_001 CF_1154_A CF_1154_B CF_1154_C GCJ19_1B_B SPOJ_CL_07974 SPOJ_CL_11063
sort	CF_1144_B GCJ19_1A_C GKS19_B_B GCJ19_1B_C CF_1156_B VTEXCC19_1_A VTEXCC19_1_G GCJ19_2_A GCJ19_2_D HR_30DOC_D28 SPOJ_CL_02727 SPOJ_CL_00297 SPOJ_CL_02123 SPOJ_CL_03375
stack	HR_30DOC_D18 SPOJ_CL_00004 SPOJ_CL_00095
string	CF_1140_B CF_1144_A GCJ19_Q_C GCJ19_1A_C CF_1146_A CF_1146_B GKS19_B_A CF_1156_B HR_30DOC_D00 HR_30DOC_D01 HR_30DOC_D06 SPOJ_CL_00004 SPOJ_CL_00400 SPOJ_CL_01021 SPOJ_CL_06219 FBCR17_1_Q3
subsequence	SPOJ_CL_00095
sum	HR_EULER_001 HR_EULER_006 CF_1144_B VTEXCC19_1_C SPOJ_CL_00042 SPOJ_CL_01043
tests	HR_EULER_001 HR_EULER_008 GKS19_PR_A CF_1140_B GKS19_PR_A CF_1144_A GCJ19_Q_A GCJ19_Q_B GCJ19_Q_C GCJ19_1A_A GCJ19_1A_B GCJ19_1A_C GKS19_B_A GKS19_B_B GKS19_B_C GCJ19_1B_A GCJ19_1B_B GCJ19_1B_C GCJ19_1C_B VJ_299509_A GCJ19_2_A GCJ19_2_D HR_30DOC_D27 HR_30DOC_D29 SPOJ_CL_02727 SPOJ_CL_00004 SPOJ_CL_00005 SPOJ_CL_00011 SPOJ_CL_00024 SPOJ_CL_00042 SPOJ_CL_00054 SPOJ_CL_00297 SPOJ_CL_00302 SPOJ_CL_00346 SPOJ_CL_00379 SPOJ_CL_00394 SPOJ_CL_00400 SPOJ_CL_00902 SPOJ_CL_01021 SPOJ_CL_01025 SPOJ_CL_01030 SPOJ_CL_01112 SPOJ_CL_01724 SPOJ_CL_02148 SPOJ_CL_02157 SPOJ_CL_03375 SPOJ_CL_03377 SPOJ_CL_03410 SPOJ_CL_03442 SPOJ_CL_03923 SPOJ_CL_04408 SPOJ_CL_06219 SPOJ_CL_06256 SPOJ_CL_07406 SPOJ_CL_07424 SPOJ_CL_07974 SPOJ_CL_08002 SPOJ_CL_10509 SPOJ_CL_11063 SPOJ_CL_14930 GKS19_C_A GKS19_C_B FBHC19_Q_A FBHC19_Q_B
totient	VJ_299509_G SPOJ_CL_04141
transpose transposition-ci	SPOJ_CL_00400
tree graph-tree	CF_1146_C VJ_299509_F SPOJ_CL_01436 SPOJ_CL_01437 VJ_304339_D
union-find	GKS19_C_A
value-iteration	VJ_299509_F
xor	SPOJ_CL_07742

Table 2: Problems reference table

ATTACHMENT 4: PROJECT POSTER

Algorithm Designing with Competitive Programming

MAC0214 - Extracurricular Activities

Undergraduate Vitor Santa Rosa Gomes
vitorssrg@usp.br, 10258862
Supervisor Nathan Benedetto Proença
nathan@ime.usp.br

Professor André Fujita
fujita@ime.usp.br
Professor Denis Deratani Mauá
ddm@ime.usp.br
Professor Kelly Rosa Braghetto
kellyrb@ime.usp.br

INTRODUCTION

Competitive programming has its roots in the scientific study of algorithms, however it focuses on implementing, submitting and debugging them. As a result, it stands out as an excellent tool to learn algorithms, because it encourages to design algorithms that really work, instead of sketching ideas that may work or not [AL].

Through practice of algorithm and data structure designing, this project aims to enrich one's Computer Science formation with a broad variety of carefully chosen real-world problems and their solutions. As a sideproduct, this project also offers to the community convenient and powerful tools to engage and encourage novice forthcomers to this highly specialized field.

METHODOLOGY

The great majority of the problems solved throughout the semester were from classical problem sets on Codeforces [CF], Sphere [SPOJ], HackerRank [HR] and LeetCode [LC] online judging platforms. Those problems were all either upvoted by the community or tailored by company interviewers as essential.

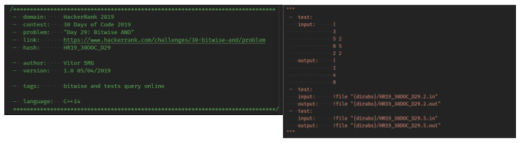
Nonetheless, competing on regular Codeforces contests and on the seasonal Google Kick Start [GKS], Google Code Jam [GCJ] and Facebook Hacker Cup [FBHC] also took place — lively whenever possible, however sometimes virtually.

Overall, more than 300 problems were studied, from which around 230 were sketched and around 70 had a working solution. Information about scheduling, the problems and their contests, some problems' tags and both the tools and compilations produced during this project is available at the <github.com/VitorSRG/MAC0214> github repository.

RESULTS

Initially intended as a personal productivity tool, a practical command line offline local judge application was created. It is not only capable of automatedly compiling, executing and testing any problem, but it also measures and limits the resources available for execution.

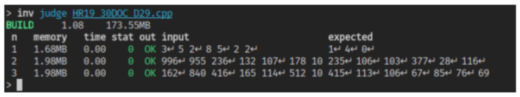
Working similarly to the online juges, it can be configured to run every time the solution is saved. In order to accomplish those, the local judge quickly parses special information contained on the programs' block comments, like in Figure 1. Upon execution, it produces an output similar to the one in Figure 2.



```
/*****************************************************************************
 -  domain:     HackerRank 2019
 -  contest:    30 Days of Code 2019
 -  problem:    "Day 29: Bitwise AND"
 -  link:       https://www.hackerrank.com/challenges/30-bitwise-and/problem
 -  hash:       HR19_30DOC_D29

 -  author:     Vitor SRG
 -  version:    1.0 05/04/2019

 -  tags:       bitwise and tests query online

 -  language:   C++14
 *****************************************************************************/
```

```
"""
 -  test:
    input:      |
                3
                5 2
                8 5
                2 2
    output:     |
                1
                4
                0
 -  test:
    input:      !file "{dirabs}/HR19_30DOC_D29.2.in"
    output:     !file "{dirabs}/HR19_30DOC_D29.2.out"
 -  test:
    input:      !file "{dirabs}/HR19_30DOC_D29.5.in"
    output:     !file "{dirabs}/HR19_30DOC_D29.5.out"
"""
```

Figure 1: Example of problem tagging in C++ and Python



```
> inv judge HR19_30DOC_D29.cpp
BUILD     1.08    173.55MB
 n   memory   time stat out input                          expected
 1   1.68MB   0.00    0  OK 3↵ 5 2↵ 8 5↵ 2 2↵              1↵ 4↵ 0↵
 2   1.98MB   0.00    0  OK 996↵ 955 236↵ 132 107↵ 178 10 235↵ 106↵ 103↵ 377↵ 28↵ 116↵
 3   1.98MB   0.00    0  OK 162↵ 840 416↵ 165 114↵ 512 10 415↵ 113↵ 106↵ 67↵ 85↵ 76↵ 69
> 
```

Figure 2: Example of local judge's output

The application can answer queries for problems, ordering the best matches given certain tags. It is important to point out that those tags are far more descriptive than usual problem tagging on the online judges, as they give away all the required insights.

Finally, every code snippet gathered from the reference books [CRLS], [SW] and [AL], alongside optimization details discussed on [CF], [CP] and [TC] forums that were used in any solution were compile under six groups: Numbers, Grouping, Ordering, Graphs, Dynamic Programming and Strings. Since no succinct, complete and optimized implementation of arbitrary-precision Integer, Decimal and Fraction were found, it is advised to use Python 3.5+ for the rare problems in which they are indeed required.

REFERENCES

[CRLS] CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford (2009). **Algorithms.** Third edition. Cambridge: MIT Press.
[SW] SEDGEWICK, Robert; WAYNE, Kevin (2011). **Introduction to Algorithms.** Fourth edition. Boston: Pearson Education.
[AL] Laaksonen, Antti. **Guide to Competitive Programming: Learning and Improving Algorithms Through Contests.** Helsinki: Springer.
[CF] **Codeforces.** Available at <codeforces.com>.
[SPOJ] **Sphere online judge.** Available at <www.spoj.com>.
[CP] **CP-Algorithms.** Available at <cp-algorithms.com/>.
[TC] **Topcoder.** Available at <www.topcoder.com/>.
[HR] **HackerRank.** Available at <www.hackerrank.com>.
[LC] **LeetCode.** Available at <leetcode.com>.
[GKS] **Google Kick Start.** Available at <codingcompetitions.withgoogle.com/kickstart>.
[GCJ] **Google Code Jam.** Available at <codingcompetitions.withgoogle.com/codejam>.
[FBHC] **Facebook Hacker Cup.** Available at <www.facebook.com/hackercup/contest>.