Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Alexandru Rudi March 29, 2023

Analyzing Competitive Programming Competitions to Develop Effective Training Methods for Improving Problem-Solving Skills in Students

By

Alexandru Rudi

Jinho D. Choi, Ph.D.
Advisor

Computer Science

Jinho D. Choi, Ph.D.
Advisor

Michelangelo Grigni, Ph.D.
Committee Member

Steven La Fleur, Ph.D.
Committee Member

2023

Analyzing Competitive Programming Competitions to Develop Effective Training Methods for Improving Problem-Solving Skills in Students

By

Alexandru Rudi

Jinho D. Choi, Ph.D.
Advisor

An abstract of
A thesis submitted to the Faculty of the
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements for the degree of
Bachelor of Science with Honors
Computer Science
2023

Abstract

Analyzing Competitive Programming Competitions to Develop Effective Training Methods for Improving Problem-Solving Skills in Students
By Alexandru Rudi

Competitive programming is a mind sport that has become increasingly popular in recent years, with thousands of programmers participating in online competitions every day. Despite its popularity, there is not much research on the competition or effective training methods for students. This study aimed to fill this gap by analyzing previous competitive programming competitions to tag problems and determine which theoretical knowledge is most valuable in improving students' problem-solving skills. We also examined how competitive programmers approach problem-solving, resulting in a flowchart algorithm for solving problems, which we tested on problems from previous competitions to show its usefulness. Based on this research, we created a series of competitive programming practice meetings for students at Emory University, tracking their progress over time. Our approach was found to be successful, culminating in the university's participation in the ICPC Southeast Regional and qualification for the North America Championship.

Analyzing Competitive Programming Competitions to Develop Effective Training Methods for Improving Problem-Solving Skills in Students

By

Alexandru Rudi

Jinho D. Choi, Ph.D.
Advisor

A thesis submitted to the Faculty of the
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements for the degree of
Bachelor of Science with Honors
Computer Science
2023

Acknowledgments

I extend my heartfelt gratitude to Jinho Choi, who served as my thesis advisor and ICPC coach. His unwavering help and support were indispensable in bringing this project to fruition. I am also thankful to my committee members, Michelangelo Grigni and Steven La Fleur, for generously sharing their valuable time and feedback, which greatly contributed to the quality of this work. I wish to acknowledge the students who participated in our meetings as well, as their engagement and contributions were crucial in realizing this project. Finally, I am deeply grateful to my parents and Sabrina for their unwavering love and support throughout this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to competitive programming

In competitive programming competitions, participants strive to develop programs that can solve a given set of programming problems within a predetermined time-frame. These problems usually contain multiple inputs, commonly known as tests, for which the participant's program must produce the expected output within a short time frame of around 1 to 5 seconds. As a result, participants must solve the problem theoretically and write highly efficient and accurate code as quickly as possible to excel in such competitions. There are many competitive programming competitions, including weekly online contests on platforms such as Codeforces [1], as well as larger international events such as the International Olympiad in Informatics (IOI) and the International Collegiate Programming Contest (ICPC).

1.2 Competitive programming training

With such a highly competitive environment and a prestigious competition, it is important to find the best way to train students for it. To succeed in competitive programming, a participant needs to have a good understanding of algorithms and

data structures, as well as experience in solving problems to quickly identify and implement solutions using ingrained pattern recognition skills. This requires a combination of both theoretical and practical training. In this thesis, we aim to create a method that enhances student improvement in these areas, measured by their ability to solve competitive programming problems.

1.3 Emory University and the ICPC

The primary objective of this study is to enhance the performance of Emory University's teams in competitive programming competitions, in particular the ICPC. For more than a decade, Emory University has been competing in the regional ICPC competition and has qualified for the North American Championship for the past three years. However, our previous practice sessions have been largely unstructured and suboptimal, with the teams practicing for only 1-2 hours per week. Our practice usually consisted of randomly selecting a contest and solving problems individually, with occasional consultation when necessary.

To address the inefficiency of our past practice sessions, we attempted to design a better methodology for competitive programming practice. Our approach involved analyzing prior ICPC competitions to identify the structure of problem solutions, creating a flowchart algorithm to aid students in problem-solving, and identifying the subset of algorithmic theoretical knowledge required for success in competitions. We created a comprehensive set of theoretical topics that are likely to appear in competitive programming problems and ranked them based on their frequency in past ICPC competitions, prioritizing easier topics by weighing each problem's difficulty. We then implemented our methodology over the course of a semester and a half through weekly practice meetings for Emory students.

1.4 Thesis statement

Despite the growing popularity of competitive programming in computer science circles, there has been limited research focused on optimal practice methods for competitive programming. Therefore, the objective of this thesis is to explore this area by designing a competitive programming practice methodology for groups of university students preparing for competitive programming competitions, with a focus on the ICPC. We aim to measure the effectiveness of our methodology by training Emory's competitive programming teams for the Southern Conference ICPC Regional. Our hypothesis is that by analyzing previous ICPC competitions, we can create an optimal curriculum for our meetings that will enable us to better focus on improving the most relevant skills necessary to solve problems, thereby maximizing students' performance in competitions. We will evaluate our methodology based on students' performance in a series of practice contests, as well as their self-reported confidence in the topics studied. At the end of the practice meetings, three teams will participate in the Southern Conference ICPC Regional, and their results will reflect their competitive programming abilities as a result of our methodology.

Chapter 2

Background

The sport of competitive programming emerged towards the end of the 20th century, with formats like the International Collegiate Programming Contest and the International Olympiad in Informatics setting the standard. The widespread availability of the internet facilitated the emergence of online competition sites like Topcoder and Codeforces, which helped popularize the sport worldwide. Today, online competitions routinely attract more than 30,000 participants [2]. As competitive programming gained popularity in programming circles, it also attracted some research attention.

2.1 Competitive programming research

Competitive programming has been the subject of several areas of research, including the automatic solving of problems through AI. One such model is AlphaCode, developed by the Google Deepmind team, which generates code solutions to competitive programming problems [10]. Other interesting areas of research include automatic problem tagging from statements or code solutions [12], detecting vulnerabilities in competitive programming code [4], identifying code plagiarism [7], and applying competitive programming tools in other contexts such as university courses [3].

In contrast, our study focused on a more traditional area of research - developing

methodologies for optimizing practice and effectively coaching students. There are several approaches to this task, such as compiling a syllabus of the theory that students should study or finding ways to optimally assign practice problems to students. For example, one paper proposed an algorithm that uses a student's history of solved problems to predict the most useful practice problem to solve next. This system was partially implemented for training high school students participating in the Italian Olympiad in Informatics [5]. However, these methods are not yet fully developed and practical enough to be applied in our case.

2.2 The format of the ICPC

The International Collegiate Programming Contest is a prestigious annual competition that challenges teams of 3 students from the same university to solve algorithmic problems within a specified time frame. The competition consists of multiple stages, starting from the regional level and progressing to the World Finals, where the winning team is crowned as the World Champion. In each competition, teams are presented with around 12 algorithmic problems of varying difficulty, and they have a time limit of 5 hours to solve as many as possible. Each participating university fields a team composed of three students, who have access to only one computer and keyboard throughout the contest. Consequently, strategizing is essential to optimize computer usage among team members. The teams are ranked based on the number of problems solved, with penalties applied for incorrect submissions and time taken to solve each problem. Each problem is described as an algorithmic challenge, usually accompanied by a story. For example, a typical ICPC problem statement might look like this:

ICPC North America Qualifier 2022 Problem M [15]

> Your state has a number of cities, and the cities are connected by roads. Unfortunately, all of the roads are toll roads!

You now run the local chapter of AAA (American Automobile Association), and people are constantly asking you about the tolls. In particular, they've been asking about individual tolls on any single road on a path between two cities. Odd, but that's what they've been asking!

Given a description of the cities in your state and the roads that connect them, and a series of queries consisting of two separate cities, for each query determine two things:

- First, the smallest value such that there is a route between the two cities where no road has a toll greater than that value.
- Second, the number of cities reachable from your starting city using no road with a toll greater than that first value.

Input

The first line of input contains three integers n ($2 \leq n \leq 2 \times 10^5$), m ($1 \leq m \leq 2 \times 10^5$) and q ($1 \leq q \leq 2 \times 10^5$), where n is the number of cities, m is the number of roads, and q is the number of queries. The cities are each identified by a number 1 through n.

Each of the next m lines contains three integers u, v ($1 \leq u, v \leq n, u \neq v$)) and t ($0 \leq t \leq 2 \times 10^5$), which represents a road between cities u and v with toll t. The roads are two-way, and the toll is the same in either direction. It is guaranteed that there is a path between any two cities, and that there is at most one road between any two cities.

Each of the next q lines contains two integers a and b ($1 \leq a, b \leq n, a \neq b$). This represents a query about a path from a to b.

Output

Output q lines. Each line is an answer to a query, in the order that they appear. Output two space-separated integers, w and k, on each line, where w is the smallest amount such that there is a route from a to b with no toll greater than w, and k is the number of cities reachable from a using no road with a toll greater than w.

Typically, each statement in the ICPC presents a few sample test cases to assist students in comprehending the problem and testing their solutions. The team is responsible for writing code that reads the input, generates the correct output, and prints it using standard I/O. When the team submits their solution file, it is assessed on the competition's server using a sequence of hidden test cases. It is the job of the problem author to ensure that these test cases are thorough enough to reject solutions that do not handle all edge cases or are inefficient. Each problem is constrained by a time and memory limit, and the team's program must not exceed these limits

during any test case. If the program submitted by the team produces the correct output within the time and memory limits for each test case, the problem is considered solved. Typically, the time limit is between 1 and 5 seconds, and the intended complexity of the problem is specified. Given the importance of runtime efficiency in many problems, the most widely used programming language among competitive programmers is C++, owing to its speed and strong standard library. The ICPC additionally allows students to submit solutions written in C, Python, Java, and Kotlin. Thus, a competitive programmer must quickly read the problem, interpret it mathematically, develop an efficient algorithm, and implement it correctly while under time pressure.

Our team's performance in the ICPC competition is an important measure of success. This year, we participated in the Southern Conference Regional, which brings together top universities from the southern United States. Teams can choose to compete in division 1 or 2, with division 1 being more challenging but offering a chance to advance to the next stages. The top 15 teams from the Southern Conference Regional Division 1 qualify for the North America Championship, where approximately 60 teams from the United States and Canada will compete. The North American Championship's top 20 teams will then advance to the World Finals, where about 150 universities worldwide will compete for the title of World Champions. In 2022, the ICPC World Finals took place in Dhaka, Bangladesh, with the top 140 universities from around the world competing for 12 medals. Emory University's team has successfully qualified for the North America Championship for the past three years but has yet to make it to the World Finals. Our closest attempt was ranking 22nd in the 2021 North America Championship.

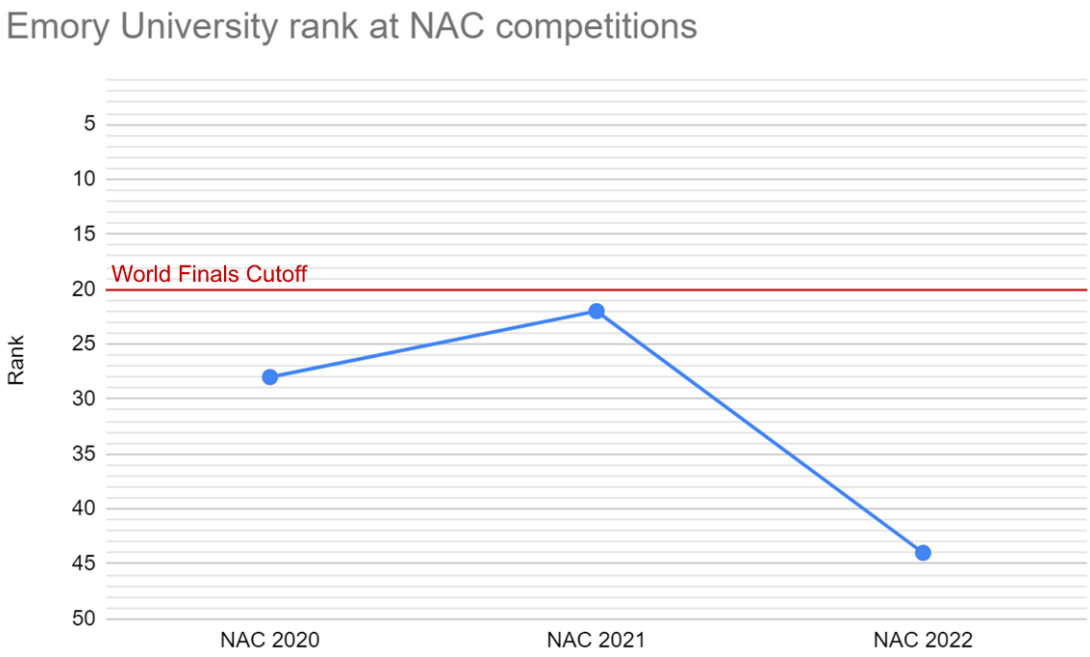


Figure 2.1: Emory University's ranking in the NAC across time

2.3 A competitive programming syllabus

Defining a clear syllabus of topics is crucial for any course on competitive programming. Some competitions such as the International Olympiad in Informatics have a set syllabus of algorithms and data structures students should be familiar with [14], in order to exclude problems that require topics like calculus or complex data structures that high school students shouldn't be expected to be familiar with. The ICPC however doesn't have rigid rules in regard to the topics that can appear, so the task of finding the most important topics is more challenging. According to Laaksonen [9], while there are similarities between the curriculum of a typical undergraduate algorithms course and that of competitive programming, there are also differences. Competitive programmers need to write short and efficient code, so they rely heavily on standard library functions and handle techniques like range queries, hashing, and binary search in unique ways. As a result, creating a syllabus based on a university curriculum may not be ideal. Instead, numerous resources exist in the competitive

programming community, such as the "Competitive Programmer's Handbook" [8], "Algorithms for Competitive Programming" website [6], and "USACO Guide" [13], which provide explanations of the most commonly used techniques, algorithms, and data structures for solving competitive programming problems, compiled by experienced competitive programmers. To develop a syllabus for our course, we used these resources to compile a list of topics.

Chapter 3

Ranking Topics for Competitive Programming

3.1 A problem-solving flowchart algorithm

The initial step in developing an effective methodology for practicing competitive programming involved transforming the problem-solving thought process of a competitive programmer into a flowchart algorithm. Creating such a flowchart can be approached in various ways, but no matter how intricate it is, it cannot cover every possible scenario since problem-solving is inherently complex. Our flowchart, depicted in Figure 3.1, strikes a suitable balance between simplicity and complexity. In the flowchart, a green arrow indicates a "yes" answer, while a red arrow denotes a "no" response.

Step 0 of the flowchart involves analyzing whether a problem can be broken down into one or more subproblems. For instance, finding the minimum spanning tree in a weighted graph can be reduced to three subproblems: sorting the edges, implementing a disjoint-union structure, and iterating through the edges in sorted order, using the disjoint-union structure to determine which edges to add to the minimum spanning

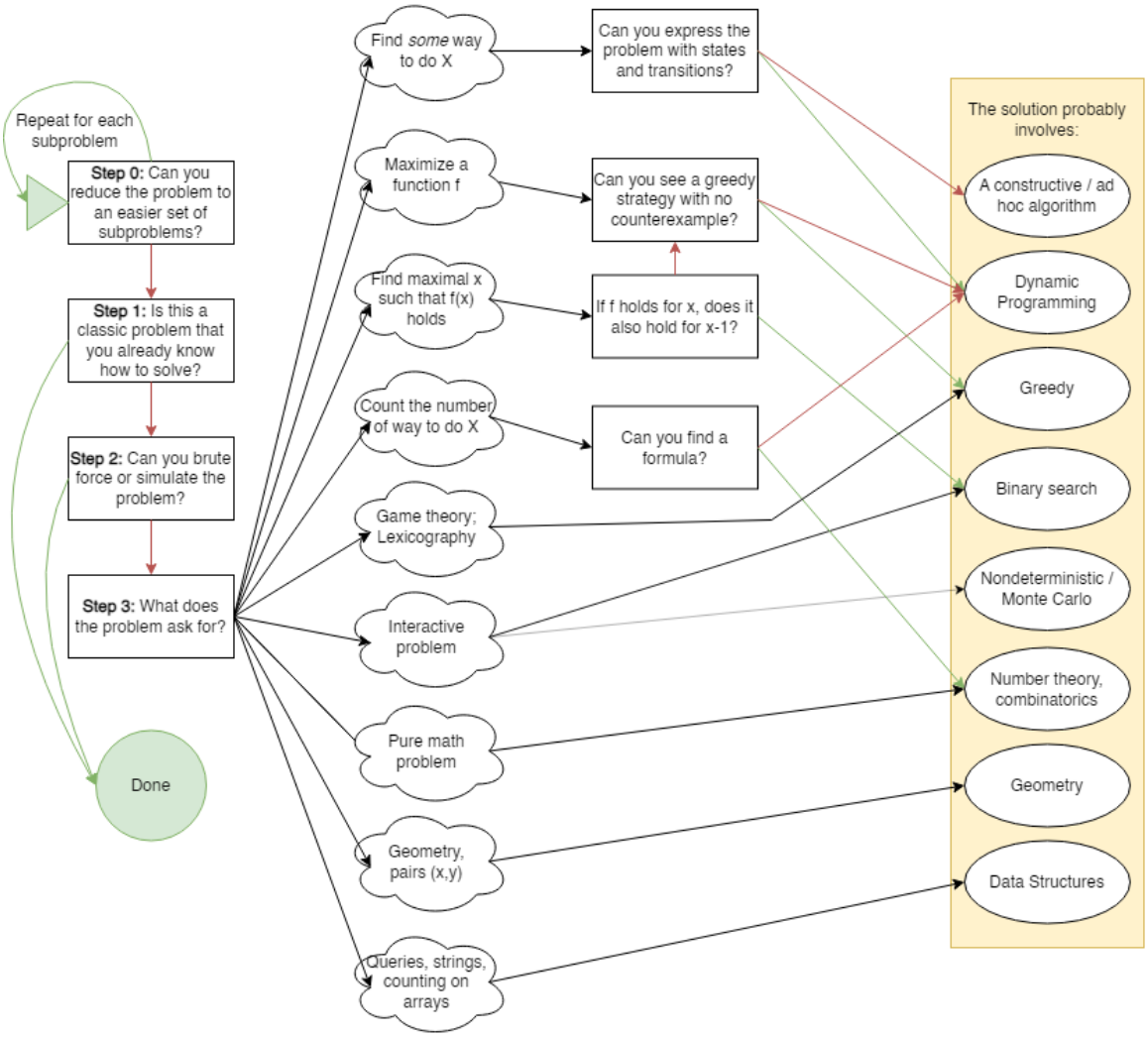


Figure 3.1: The problem-solving flowchart

tree. It is also possible for a reduction to only create one subproblem, thus essentially just reducing the complexity of the problem, such as re-expressing a problem that deals with pairs of elements into a graph problem. Step 0 is often the hardest one, and rarely actually occurs at the start of the problem-solving process, instead often being a result of the latter stages in the flowchart.

Step 1 of the methodology is critical as it prompts the question of whether the problem at hand is one that the student has encountered before. For instance, in the previous example, the subproblems of sorting the edges and implementing the disjoint-union structure are topics that a student should already know how to solve

due to their frequent appearance as subproblems. In this paper, we will refer to such subproblems as "topics". The more familiar the student is with these topics, the quicker they can break down a given problem into solvable subproblems and solve it. Therefore, it is essential to identify which topics are common and should be mastered by every student and which are obscure and not worth studying.

Step 2 involves checking whether the problem can be solved using a simple simulation or brute force approach. For instance, a problem asking for the number of vowels in a string can be solved by examining each letter. While these types of problems are common at the beginner level, they can become more challenging when the solution space is exponential, as in cases where all permutations of an array must be checked. Recursive algorithms are often required in such cases, which can be difficult to understand at an early stage.

If the problem cannot be solved through well-known methods or brute force, the student must find a clever algorithm to tackle it. Nevertheless, there are still discernible patterns at this stage that can be exploited. The vast majority of algorithms can be categorized into five types: dynamic programming, greedy, divide and conquer (which is often implicitly embedded within a data structure or a binary search, as noted by [9]), non-deterministic (such as Rabin-Karp hashing), and constructive/ad-hoc algorithms. The last category is difficult to define, but generally refers to algorithms that are specific to a given problem and are challenging to abstract, such as constructing a string with a particular property. Furthermore, certain problems involve concepts from number theory, combinatorics, and geometry, requiring more specialized algorithms, such as the Fast-Fourier Transform algorithm or using a radial sweep to find the convex hull of a set of points. Often, the problem statement alone can provide a good idea of the category of the solution, and step 3, along with the rest of the flowchart in Figure 3.1, exemplifies one such algorithm for determining the category of a solution from the statement. However, the algorithm remains deliberately

imprecise to maintain its broad applicability.

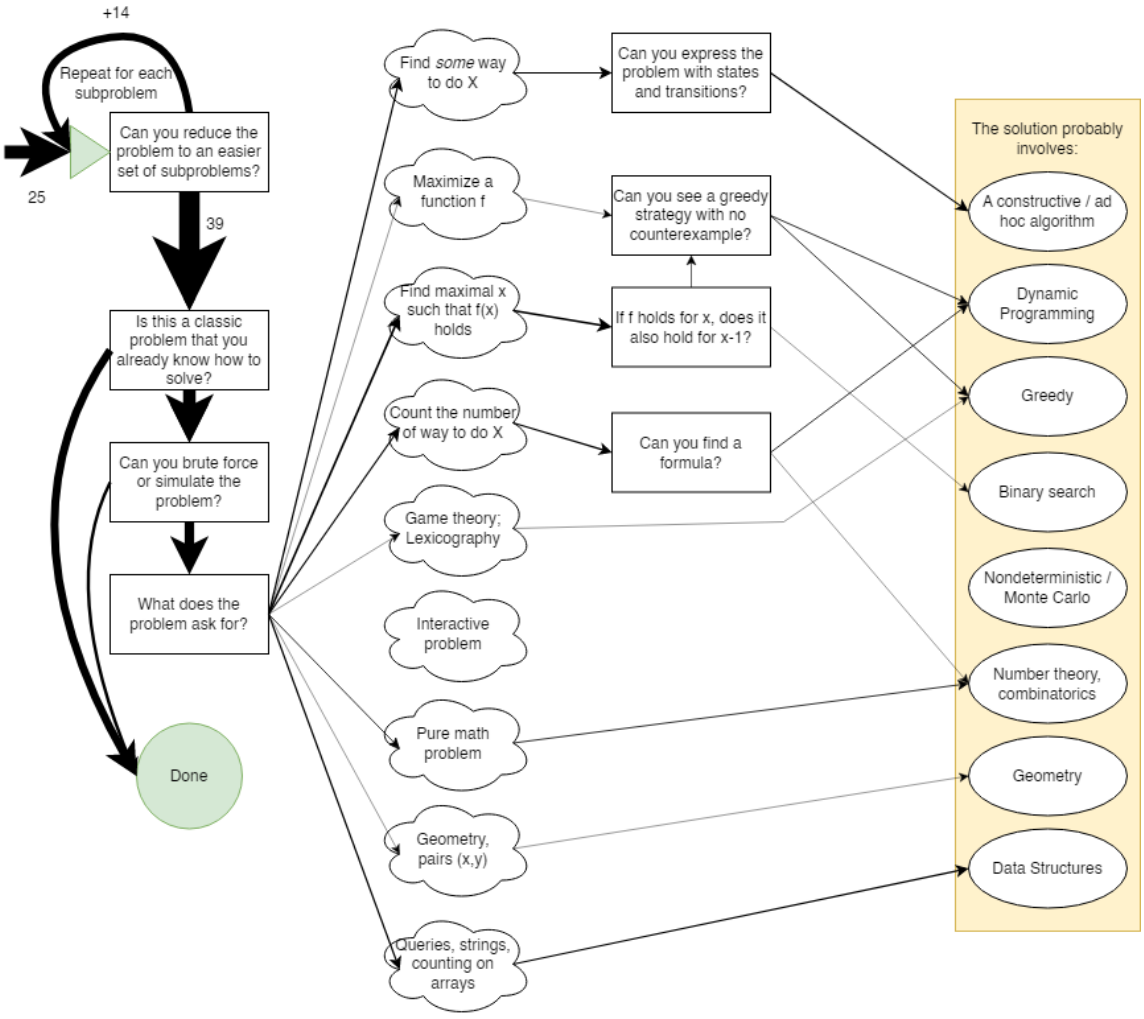


Figure 3.2: The problem-solving flowchart weighted by frequency in our data

To evaluate the effectiveness of our flowchart in real-world scenarios, we conducted tests using 25 problems from two competitions - the North America Qualifier 2022 and the South Conference Regional 2022. We converted the solution for each problem into our flowchart's format, resulting in 39 subproblems to solve after the reduction process. Each subproblem represents a path from the starting point to either the "done" node, in case the problem is well-known or can be brute-forced, or to one of the technique nodes, indicating the technique the solution employs and the hints from the statement suggesting this technique For each edge e in our flowchart, we

kept track of the number n_e of subproblems that included that edge in their path, and the results can be seen in 3.2, where the thickness of each arrow is proportional to n_e. Additionally, Appendix B displays the path of each subproblem in detail. Our flowchart successfully represented each of the 39 subproblems to a satisfactory degree, demonstrating its general applicability to a majority of ICPC problems. We also see that almost all edges in our flowchart appeared in at least one subproblem, with the notable exception of interactive problems, which appear very rarely if at all at ICPC-type competitions but are nonetheless pretty common in online competitions and other international competitions such as the IOI, and thus were included in the flowchart. In summary, our flowchart is general enough to apply to most ICPC problems while being specific enough to provide useful information about a problem's solution.

3.2 Measuring topic importance

Although experienced competitive programmers have an intuition regarding the relative frequency of different topics in competitive programming problems, we sought to support this intuition with concrete data. To achieve this, we generated a comprehensive list of topics that encompasses almost all the topics needed at the beginner and intermediate levels. We compiled this list of 58 algorithms, data structures, and techniques that span from beginner to advanced levels, utilizing resources such as the "Competitive Programmer's Handbook" [8] and "Algorithms for Competitive Programmers" [6]. These resources are maintained by the competitive programming community, frequently updated, and expanded, and as such, the list of topics we compiled from them is sufficient to solve nearly all intermediate competitive programming problems.

We selected eight contests, namely the North American Championship 2020, 2021,

and 2022, the World Finals 2017, 2018, and 2019, the North America Qualifier 2022, and the South Conference Regional 2022, to construct our dataset of 97 intermediate to advanced level problems. For each problem, we analyzed the solution and identified which topics appear as subproblems in the solution, and tagged those topics to the problem. Let $a_{i,j} = 1$ if the j-th topic is used in the solution of the i-th problem, and $a_{i,j} = 0$ otherwise. Additionally, we gave each problem a difficulty score d_i between 0 and 1, where

$$d_i = \frac{\text{number of teams that solved problem} i}{\text{number of teams in the contest problem } i \text{ belongs to}}$$

The final importance score of the topic j is

$$w_j = \sum_i d_i \cdot a_{i,j}.$$

The topics that appear more often or appear in easier problems have a higher score, meaning a student should prioritize learning them more, meaning that w_j is the metric we were looking for to measure topic importance. You can see the results in Appendix A, which we later used to create our curriculum.

3.3 Practical and theoretical knowledge

As mentioned before, it is important to make the distinction between practical knowledge and theoretical knowledge in competitive programming. For example, most computer science students have a very good theoretical understanding of binary search, being able to describe and implement a basic application of the algorithm. However, more often than not, when binary search is used in a competitive programming problem, it is well hidden and requires problem-reduction skills to observe that the problem can be reduced to binary search. Our flowchart in Figure 3.2 shows that

out of the 39 subproblems we analyzed, 15 (38%) were well-known problems that fall under theoretical knowledge, while the remaining 24 (62%) required practical knowledge to solve. Practical knowledge skills are difficult to teach and require practice on specific problems to master. Therefore, competitive programming places a greater emphasis on practical knowledge than a university curriculum for example. Oftentimes the most successful competitive programmers have relatively limited algorithmic knowledge, specializing in only what is strictly required to solve problems. As such, as a rough approximation, a good competitive programmer only spends about 10% of their practice time on studying theory, instead focusing the majority of their time on solving problems and gaining practical knowledge, which is something that we will take into account in our methodology.

Chapter 4

Structuring a Semester of Practice Sessions

After researching the most important topics, we developed a curriculum that we taught to the students over the course of the fall semester. Since most of the students were computer science majors, we assumed they had a basic level of familiarity with these topics to be able to fit all these topics into one semester. As such, each practice session would introduce 2-4 topics to the students. We focused on practical knowledge, which we believe is essential for improving skills, particularly at an early stage. To this end, we designed each lecture to maximize practice time rather than overloading the students with information. Each lecture lasted 90 minutes and was split into three 30-minute segments. During the first segment, we presented three problems related to the current topic, designed to be of relatively easy difficulty. This allowed students who were already familiar with the topic to practice, while those who were unfamiliar would be presented with a concrete problem that required knowledge of the topic to solve, but they would hypothetically try to solve with their current knowledge and fail, thus allowing them to see why the new topic is required for solving the problems. In the second segment, we delivered a 30-minute lecture on the theory behind the

Meeting #	Topics
1	Time Complexity, C++ STL, Greedy, Complete Search 1
2	Complete Search 2, Bitmasks, Binary Search, Query Problems, Prefix Sums
3	Number Theory & Combinatorics
4	Two Pointers, Sliding Window, Dynamic Programming 1
5	Trees, Graphs, DFS, BFS, Shortest Path Algorithms
6	Dynamic Programming 2
7	Graph Algorithms - Greedy and Dynamic Programming
8	Sparse Tables, Segment Trees
9	Rabin-Karp Hashing

Table 4.1: List of topics for each meeting

topic, accompanied by implementations to reinforce their understanding. In the final segment, we presented the same three problems as before along with three more difficult problems, intended for practice at home. This provided students with the opportunity to put their new knowledge into practice and interact with the provided implementations or implement it themselves.

We structured our 9 meetings (excluding the introductory meeting and a 4-hour practice contest mid-way through) according to the plan shown in Figure 4. We introduced topics sequentially ordered roughly by difficulty and grouped similar topics in the same day. However, we did not strictly adhere to the topic importance scores due to our limited dataset, which resulted in some topics being deemed more important than they actually are in practice or not suitable for beginner to intermediate students, such as "suffix arrays" or "max flow."

Based on our topic rankings, the sum of w_j of all topics is 22.4, while the sum of w_j of the topics we selected is 15.1, which represents 67.4% of the total. This suggests that mastering our selected topics should provide students with the theoretical knowledge needed to solve most beginner and intermediate problems.

Selecting practice problems for each meeting was also a crucial task. We obtained the problems from Codeforces [1], the most widely used website for competitive pro-

gramming worldwide. To identify appropriate problems, we searched their database of problems from previous contests, filtering by tags related to the topic, such as greedy, and hand-picking problems. Given our limited practice time, we prioritized problems with short, simple statements that had solutions that clearly used the given topic.

Our team's main webpage was hosted on Github [11], providing comprehensive information about our meetings and team, a schedule of events for the year, as well as a list of practice resources and tips, including references to valuable resources such as the Competitive Programmer's Handbook [8] Our schedule provided links to the slides for each meeting and the practice contests. Most importantly, the Github repository provided original code implementations for every algorithm and data structure in our schedule. This allowed students to reference the code while studying the topics and simplified the implementation process for applying the new knowledge to concrete problems. The Github page even garnered attention outside of Emory, appearing in 11 unique Google searches in just one month and receiving 4 stars from non-Emory users.

4.1 Meeting 0

Introduction

Attendance: 23

In the initial meeting, I introduced competitive programming to the participants and presented them with a 1-hour contest consisting of 6 problems, designed to be simple and familiarize them with the format. A total of 22 students took part in the contest, with 2 students solving 2 problems and 3 students solving 1 problem. After the contest, a survey was conducted with 7 respondents, revealing that the primary difficulties encountered were debugging (4), inability to find a solution (2), and inabil-

ity to implement a solution (2). This, along with the students' personal experiences, indicates that becoming accustomed to the contest format and input-output operations are the primary initial hurdles. The students who were more familiar with the format were able to solve 1-2 easy problems, which serves as the starting point for our efforts to enhance their skills.

4.2 Meeting 1

Time Complexity, C++ STL, Greedy, Complete Search 1

Attendance: 15

As Codeforces was undergoing maintenance during the meeting, a practice contest was not held. Instead, I conducted a 1.5-hour lecture on the relevant topics and provided code demonstrations. The discussion centered around time complexity and its relevance to competitive programming, including how the CPU can perform approximately 10^9 simple operations per second, and the constraints on N that allow complexities such as $O(N), O(N^2)$, or $O(2^N)$. Additionally, we explored the use of programming language libraries, specifically the C++ STL, and how to employ them efficiently. The topic then shifted to greedy strategies, with examples of how they can be applied to scheduling problems and the essential principles for proving their optimality. Finally, we delved into complete search and backtracking, showcasing how to utilize recursion to generate all subsets and permutations of a given set.

4.3 Meeting 2

Complete Search 2, Bitmasks, Binary Search, Query Problems, Prefix Sums

Attendance: 14

In this meeting, we concluded our exploration of complete search by examining further applications, such as the queen's problem. Additionally, I presented a clever

approach to generate subsets without recursion using a bitmask, encoding each subset of a set comprising N elements into the bits of a number ranging from 0 to $2^N - 1$. We also delved into binary search and its utility in answering queries, such as finding the k-th element in an ordered set, quickly. Lastly, we discussed the prefix sums technique, which stores the sum of each prefix of an array, enabling us to answer interval sum queries in constant time by representing them as the difference between two prefixes.

No participants were able to solve any problems during the contests, potentially because the first problem was quite challenging. Upon analyzing the attempted submissions, we noticed that none of the students had correct solution ideas before the lecture. However, after the lecture, some students had the correct solution ideas but failed to implement them in time. The survey conducted after the meeting indicated that most individuals struggled with translating their solutions into code, confirming our observations and showcasing the need to place greater emphasis on implementing the algorithms during the lectures.

4.4 Meeting 3

Number Theory and Combinatorics

Attendance: 15

In this meeting, we explored various concepts from number theory and combinatorics, including the modulo operator, prime factorization, the sieve of Eratosthenes, Euclid's algorithm, logarithmic exponentiation, modular inverse, and binomial coefficients. We also covered the implementation details for all of these algorithms.

During the practice contest, the students performed better than the previous week, with three individuals successfully solving at least one problem. However, as in the previous meeting, the survey conducted after the session revealed that most

participants faced challenges in translating their solutions into code. Notably, all respondents acknowledged having a solution in mind that, after the lecture, they realized was incorrect, indicating that the lecture had a positive impact on deepening their understanding of these topics.

4.5 Meeting 4

Two Pointers, Sliding Window, Dynamic Programming 1

Attendance: 10

In this meeting, we delved into various techniques for problem-solving on arrays, including two-pointers and sliding window methods. These techniques are useful for solving problems such as determining whether there exist two elements in an array of length N that add up to a given sum X, with a time complexity of $O(N \log N)$. Additionally, we introduced dynamic programming, which we used to solve the longest increasing subsequence problem in $O(N^2)$ and the maximum path in a grid from the top-left to bottom-right problem in $O(N \cdot M)$.

Unfortunately, Codeforces experienced downtime again, causing disruption during the second practice contest. However, we were still able to achieve a few correct solutions for the first problem.

4.6 Practice Contest 1

Attendance: 3

Following the fourth meeting, we organized an offline practice contest which had three student participants. The contest problems were similar in difficulty to those presented in the first meeting, and this time, one student managed to solve three problems while two others solved two problems each. This is a notable improvement of one additional solved problem compared to the first contest, indicating that practice

had a positive impact on their skills.

4.7 Meeting 5

Trees, Graphs, DFS, BFS, Shortest Path Algorithms

Attendance: 4

During this lesson, we covered fundamental graph algorithms, including DFS, BFS, Dijkstra, Bellman-Ford, and Floyd-Warshall, providing an opportunity to practice data structure and dynamic programming skills (priority queues in Dijkstra, dynamic programming approaches in the latter three algorithms).

Although only four students attended the session due to a building fire alarm in the building, they performed well, with one student even solving an intermediate difficulty problem according to the Codeforces' rating system. Following this point, the students grew accustomed to the problem format and competitions, becoming more comfortable with their programming languages and implementing their solutions. Issues such as input/output ceased to arise, and the students began to concentrate on more abstract concepts.

4.8 Meeting 6

Dynamic Programming

Due to scheduling conflicts, this meeting was held offline. The students were presented with a dynamic programming contest featuring 25 classical problems of varying difficulty levels. Their task was to solve as many problems as possible, which were later reviewed and discussed in a subsequent meeting during the second semester.

4.9 Meeting 7

Graph Algorithms - Greedy and Dynamic Programming

Attendance: 4

In this session, we delved deeper into graph algorithms, revisiting both greedy and dynamic programming techniques. Prim's algorithm for finding the minimum spanning tree of a weighted graph was covered under greedy, while we discussed directed acyclic graphs and topological sorting as part of dynamic programming. We learned how to perform dynamic programming on directed acyclic graphs using the topological sorting, for example, finding the longest path in a directed acyclic graph.

Attendance remained low with only 3 to 4 students showing up for this and subsequent meetings. In general, students solved more problems after the lecture than before, which could be attributed to either the lecture or to them having more time to think about the problems.

4.10 Meeting 8

Sparse Tables, Segment Trees

Attendance: 3

At this meeting, we introduced two advanced data structures - sparse tables and segment trees - which are commonly used in competitive programming. Segment trees, in particular, are highly versatile and have a short implementation, making them a popular choice for data structure problems. However, it could be argued that Fenwick trees, despite being more complicated in theory and less versatile than segment trees, should also be included due to their extremely short implementation.

4.11 Meeting 9

Rabin-Karp Hashing

Attendance: 3

In this meeting, we covered Rabin-Karp hashing, a highly flexible approach for solving string-matching problems that does not require the use of challenging-to-implement data structures like suffix arrays or intricate algorithms such as Knuth-Morris-Pratt. Nonetheless, it has its own intricacies, such as the requirement to be familiar with concepts like hash collisions, the birthday paradox, and number theory concepts like modular arithmetic and inverses. Nevertheless, of all intermediate string algorithms, we opted for Rabin-Karp due to its high versatility and ease of implementation once comprehended thoroughly.

4.12 Measuring the semester

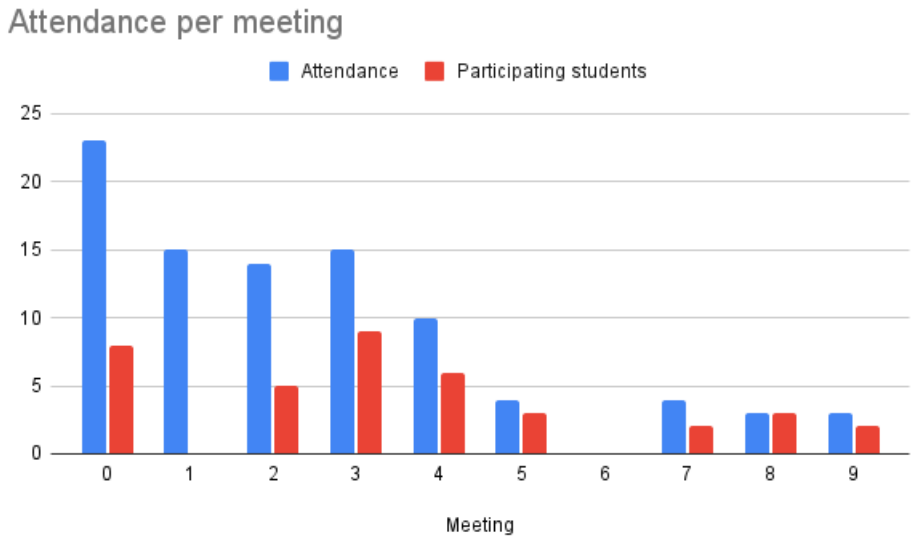


Figure 4.1: Attendance per meeting

The attendance trend over the course of the meetings can be observed in Figure 4.1. While the attendance started off strong, only a small number of students, typi-

cally 3-4, were present in the final meetings. This drop in attendance may have been caused by various factors, such as waning interest, the difficulty level of the practice problems, and the pace of the lectures. The number of participating students in each meeting, defined as those who attempted to submit at least one solution during the practice contests, is also depicted in the graph. The graph also reveals disruptions in attendance, such as the absence of practice contests during meeting 1 and the virtual format of meeting 6. These meetings have been excluded from subsequent analyses.

On average, 60% of the students submitted code solutions during practice contests in our meetings, with participation gradually increasing as the number of participants decreased and their skills improved. To evaluate the students' performance, we used the difficulty ratings of the practice problems provided by Codeforces, where each problem is rated at least 800, with lower ratings indicating easier problems. We measured a student's performance using two metrics: summing the ratings or taking the maximum rating of the problems they solved during each practice contest. Although neither metric is perfect, as nuances such as the difficulty of a single high-rated problem versus two lower-rated problems are not fully captured, both metrics are adequate for our purposes. Figure 4.2 shows the average performance of the students for each meeting using both metrics, with an upward trend, albeit with unreliable data due to the small sample size and short practice contests. To further measure the success of our meetings, Figure 4.3 depicts the performance of the top two students throughout the meetings, as they attended almost all meetings and clearly showed improvement during the second half of the semester.

After the regional competition, a survey was distributed to the students, which was completed by 7 of them. However, only 5 of the 7 students had attended during the first semester. Consequently, only the responses of these 5 students were taken into account for the subsequent analysis. The survey aimed to evaluate the students' proficiency in a range of broad competitive programming concepts before and after

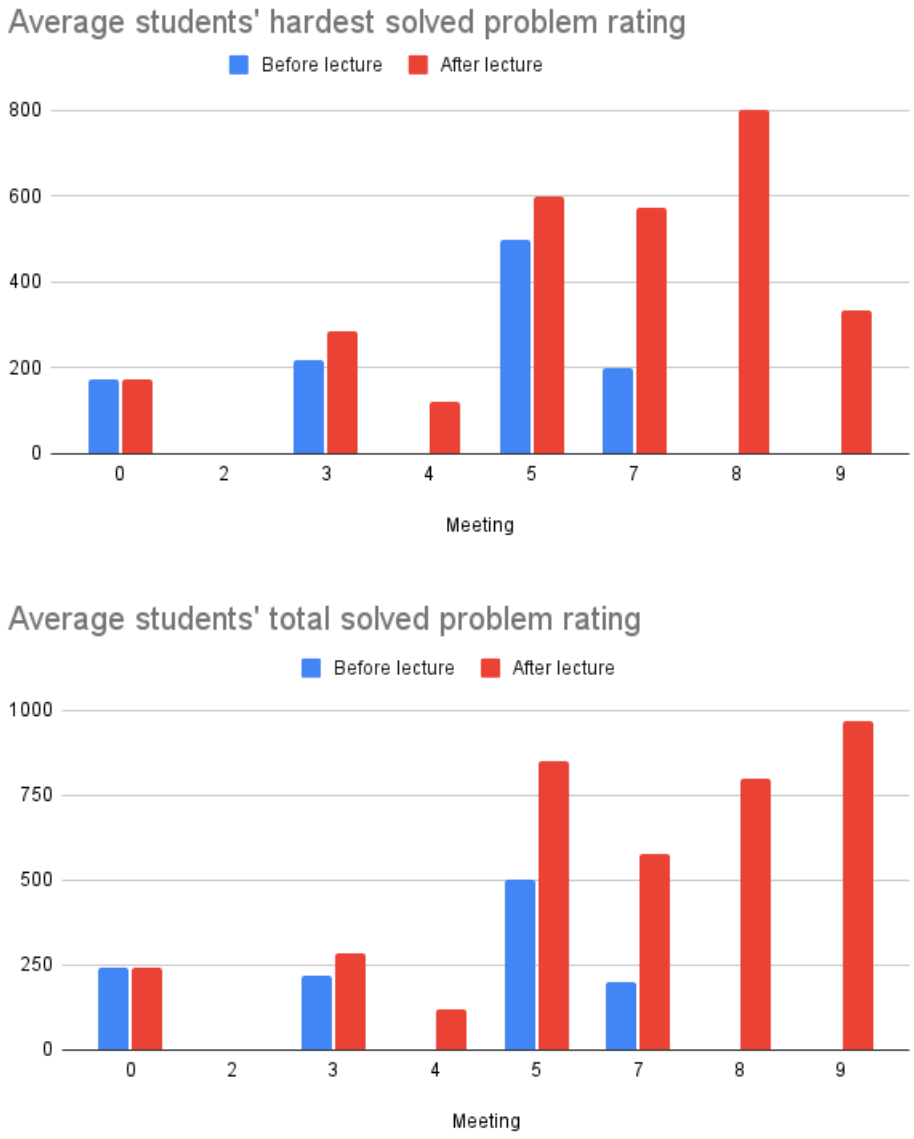


Figure 4.2: Average students' performance as measured by hardest solved problem rating and total solved problem rating

attending any meeting throughout the year. Each concept was rated on a scale of 0 to 5 based on proficiency, with 0 indicating no prior knowledge, 1 indicating prior knowledge but no study, 2 indicating attempted study but poor understanding, 3 indicating theoretical understanding but no practical experience, 4 indicating basic problem-solving ability, and 5 indicating a deep understanding. The results, presented in Figure 4.4, demonstrated that students felt an improvement in every concept. On

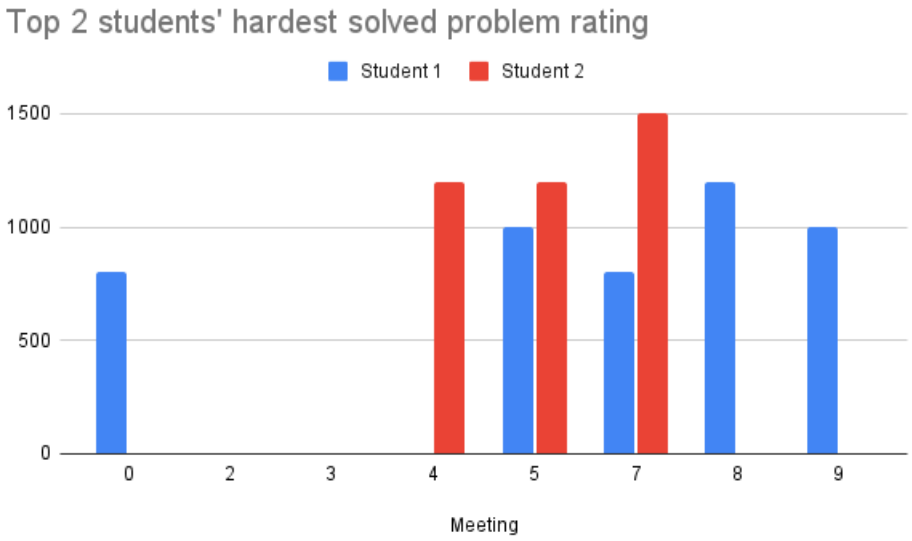


Figure 4.3: Top 2 students' hardest solved problem rating

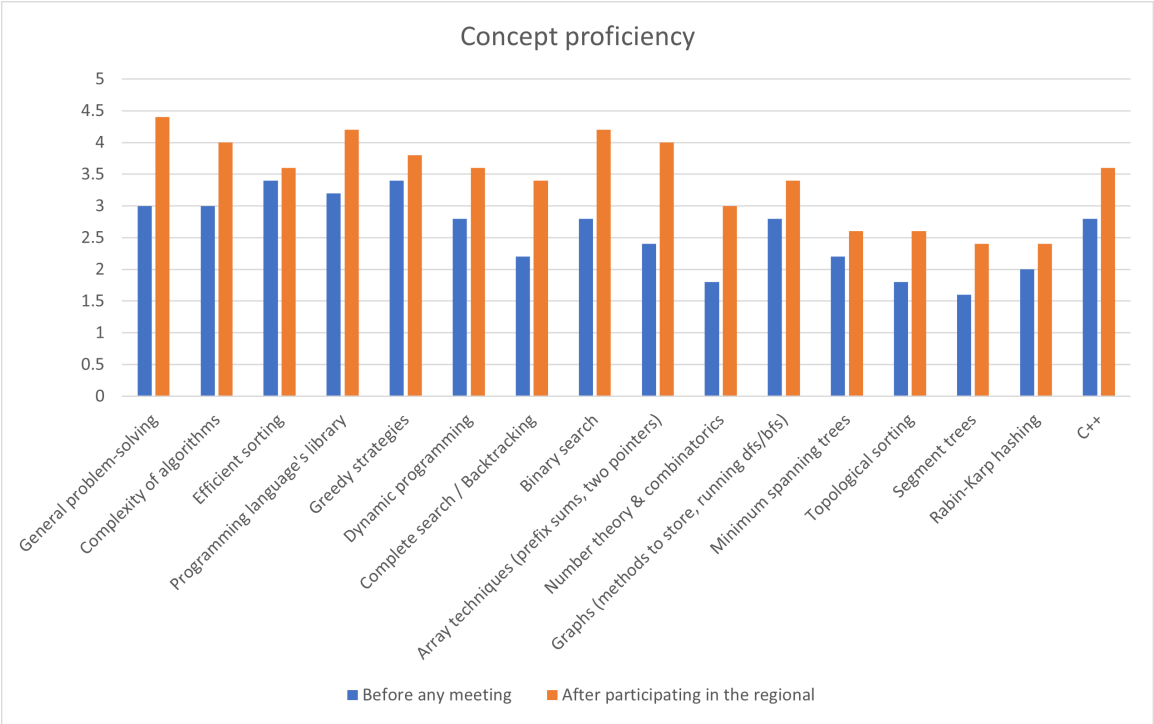


Figure 4.4: Concept proficiency

average, students reported a 36% increase in proficiency across all concepts, with number theory, combinatorics, and array techniques being the most well-performing areas.

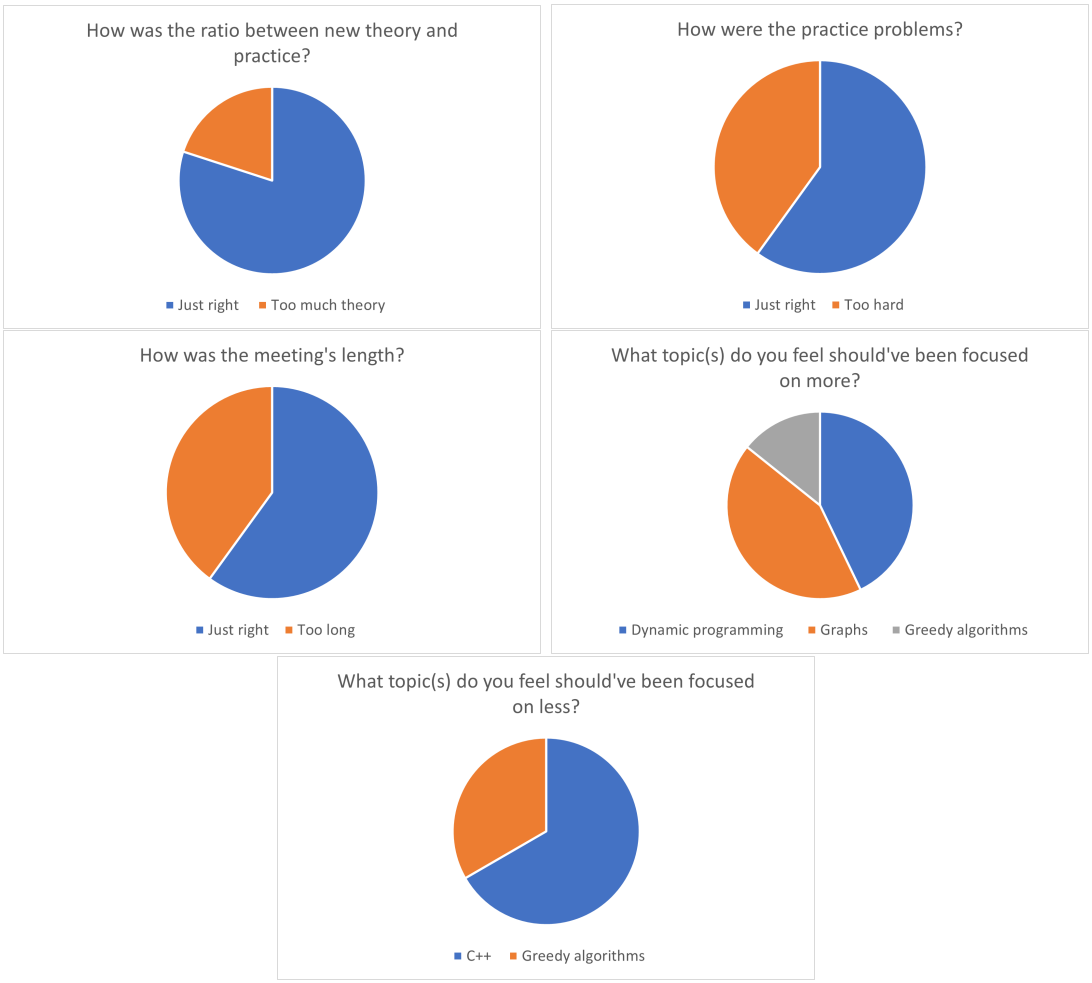


Figure 4.5: Results of student opinion survey

Furthermore, the students were surveyed with five questions to gather their opinions on the meeting's structure, and the results are presented in Figure 4.5. Overall, the students appreciated the format, with 80% of them feeling that the balance between theory and practice was appropriate. However, on average, the practice problems were deemed slightly too difficult, and the meetings were slightly too long, which may have contributed to a decrease in interest and lower attendance rates in the latter half of the semester. In addition, the students were asked to identify topics that should receive more or less focus, and the frequency of each topic's mention in the responses revealed that the curriculum was lacking in dynamic programming and graph theory while covering C++-specific knowledge too extensively.

Chapter 5

Developing Practical Skills

During the second semester, the students had acquired sufficient knowledge in the topics covered in the previous semester. Therefore, the emphasis was shifted to developing practical skills through problem-solving. To achieve this, the duration of the lectures was extended by an hour, and the entire time was dedicated to individually solving problems from a provided ICPC contest. As each student had varying levels of proficiency, individual problem-solving was considered optimal since it enabled them to skip the easy problems and devote more time to those that they found challenging and productive. However, they were encouraged to seek help when they were unable to generate new ideas for a problem within 10-15 minutes. If a student was stuck but still had some new ideas, they were to continue working on the problem, and if genuinely stuck, a hint would be provided to guide them towards the correct approach. If a hint proved inadequate, the entire solution was discussed, and students were requested to implement it.

5.1 Meeting 1

ICPC Southeast Regional 2021 Division 2

Attendance: 5

The students' progress was evident in the contest as all of them were able to solve a minimum of three problems. Two students particularly demonstrated significant improvement, solving up to six problems and even discussing the seventh problem with me. It was apparent that the students were adept at utilizing programming techniques like greedy and dynamic programming to their advantage. None of the students encountered difficulties related to input/output or a lack of knowledge about the programming language or contest format.

5.2 Team Selection Test

Attendance: 4

The team selection test for the main Emory team was successful, with impressive performances from the students. The top two participants, selected as members of our main team this year, solved 6 and 7 problems, respectively. These results indicate that they would have scored highly in the contest that was simulated, demonstrating that the students were now equipped to compete with some of the best competitive programmers in the country after completing the lectures.

5.3 Meeting 2

ICPC Southeast Regional 2020

Attendance: 8

Although the students continued to perform satisfactorily, it was evident that they required additional practice to solidify their understanding of the material covered in the previous semester. In particular, they encountered challenges when attempting to reduce a problem to concepts such as topological sorting and expressed concerns about their ability to effectively implement it.

5.4 Meeting 3

ICPC Southeast Regional 2020

Attendance: 7

To reinforce the students' proficiency in implementing complex algorithms, we continued practicing with the same contest since we had not made significant progress previously. In addition, we practiced using Team Reference Documents, a feature of the ICPC that enables the team to refer to pre-implemented algorithms during the contest. This approach aimed to enhance the students' confidence and familiarity with utilizing challenging algorithms in a competitive setting.

5.5 Meeting 4

ICPC North America Qualifier 2022

Attendance: 7

Our teams participated in the ICPC North America Qualifier 2022, held last Saturday, which was a preparatory contest for all teams before the regional competitions. Both teams performed well, with the primary team solving 9 out of 12 problems and the secondary team solving 4. Following the competition, we conducted a comprehensive review of all the problems, and some students demonstrated an understanding of even the most challenging solutions. This demonstrated that their skills have significantly improved, enabling them to tackle most intermediate and even some advanced problems.

5.6 Meeting 5

Dynamic Programming

Attendance: 6

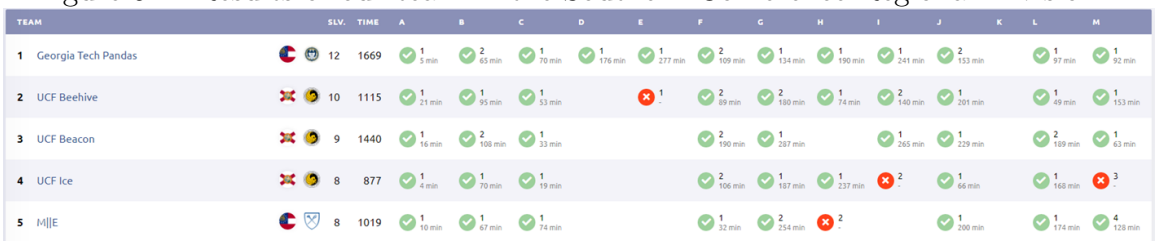
As the students were uncertain about their grasp of dynamic programming, we dedicated the entire meeting to reviewing dynamic programming problems from the previous semester's contest. Using concrete problem examples proved helpful in strengthening the students' understanding of the dynamic programming technique, and they became more adept at formulating dynamic programming states and transitions.

5.7 The South Conference Regional

On Feb. 25, the South Conference Regional competition was held, featuring 271 teams from the top universities in the southern United States, including the Georgia Institute of Technology, the University of Central Florida, the University of Maryland, the University of Texas at Austin, Texas A&M University, and the University of Texas at Dallas. Emory University was represented by three teams: team M||E in division 1, and teams M|E and MorE in division 2. At the competition, team M||E placed 12th out of 152 in the South Conference Regional Division 1, qualifying for the North America Championship in May. Moreover, team M||E received the Bronze medal for being the 5th team out of 35 in the Southeast Region and the Gold medal for placing first at the Augusta site. Our division 2 teams also had a strong showing, with team M|E placing 19th and team MorE placing 30th out of 119 teams in the South Conference Regional Division 2. They also placed 2nd and 4th out of 15 at the Augusta site, which would have resulted in a silver medal, but our division 2 teams were not eligible for medals due to our university having a team participating in division 1. These results reflect the great effort put in by all three of our teams this year.

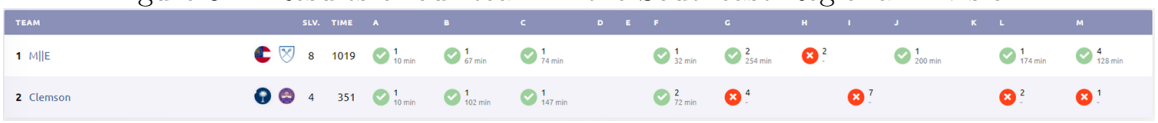
	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Georgia Tech Pandas	12	1669	1 5 min	2 65 min	1 70 min	1 176 min	1 277 min	2 109 min	1 134 min	1 190 min	1 241 min	2 153 min		1 97 min	1 92 min
2	UCF "Pickup" Cactus	12	1681	1 9 min	1 40 min	1 73 min	1 284 min	1 238 min	1 170 min	2 85 min	1 201 min	2 172 min	2 156 min	5 -	1 114 min	2 59 min
3	UMD RED	10	998	1 4 min	1 24 min	1 35 min		1 63 min	2 54 min	1 126 min	1 231 min	1 -	1 137 min		1 82 min	2 202 min
4	UCF Beehive	10	1115	1 21 min	1 95 min	1 53 min		1 -	2 89 min	2 180 min	1 74 min	2 140 min	1 201 min		1 49 min	1 153 min
5	UT Cerulean	10	1810	4 22 min	6 97 min	2 34 min	3 253 min	2 131 min	1 144 min			4 292 min	5 208 min		1 99 min	4 90 min
6	Lithium Flower	9	942	1 11 min	1 46 min	1 58 min		1 142 min	1 27 min		1 194 min	1 -	1 230 min		2 91 min	2 103 min
7	UCF Beacon	9	1440	1 16 min	2 108 min	1 33 min			2 190 min	1 287 min		1 265 min	1 229 min		2 189 min	1 63 min
8	tableflipTLE	9	1459	1 19 min	2 92 min	1 101 min		2 258 min	4 143 min		1 284 min		1 241 min		2 140 min	1 61 min
9	UT Orange	9	1557	1 19 min	2 178 min	3 78 min			1 110 min	1 -	1 88 min	3 253 min	2 227 min		2 165 min	2 279 min
10	whoosh	9	1611	1 4 min	6 107 min	1 47 min			3 141 min	4 299 min	1 278 min	4 -	2 154 min		1 175 min	5 106 min
11	UCF Ice	8	877	1 4 min	1 70 min	1 19 min			2 106 min	1 187 min	1 237 min	2 -	1 66 min		1 168 min	3 -
12	M||E	8	1019	1 10 min	1 67 min	1 74 min			1 32 min	2 254 min	2 -		1 200 min		1 174 min	4 128 min

Figure 5.1: Results of our team in the Southern Conference Regional Division 1



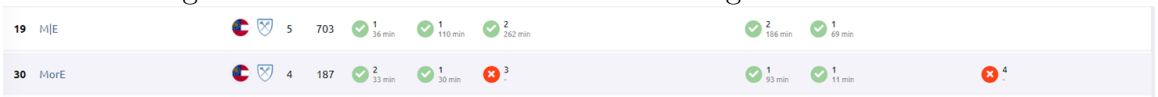
	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Georgia Tech Pandas	12	1669	1 5 min	2 65 min	1 70 min	1 176 min	1 277 min	2 109 min	1 134 min	1 190 min	1 241 min	2 153 min		1 97 min	1 92 min
2	UCF Beehive	10	1115	1 21 min	1 95 min	1 53 min		1 -	2 89 min	2 180 min	1 74 min	2 140 min	1 201 min		1 49 min	1 153 min
3	UCF Beacon	9	1440	1 16 min	2 108 min	1 33 min			2 190 min	1 287 min		1 265 min	1 229 min		2 189 min	1 63 min
4	UCF Ice	8	877	1 4 min	1 70 min	1 19 min			2 106 min	1 187 min	1 237 min	2 -	1 66 min		1 168 min	3 -
5	M||E	8	1019	1 10 min	1 67 min	1 74 min			1 32 min	2 254 min	2 -		1 200 min		1 174 min	4 128 min

Figure 5.2: Results of our team in the Southeast Regional Division 1



	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L	M
1	M||E	8	1019	1 10 min	1 67 min	1 74 min			1 32 min	2 254 min	2 -		1 200 min		1 174 min	4 128 min
2	Clemson	4	351	1 10 min	1 102 min	1 147 min			2 72 min	4 -		7 -		2 -		1 -

Figure 5.3: Results of our team at the Augusta site Division 1



19	M|E	5	703	1 36 min	1 110 min	2 262 min				2 186 min	1 69 min					
30	MorE	4	187	2 33 min	1 30 min	3 -				1 93 min	1 11 min			4 -		

Figure 5.4: Results of our teams in the Southern Conference Regional Division 2

Chapter 6

Discussion

Overall, the results from the practice meetings, opinion survey, and ICPC regional performance indicate that our methodology was effective in preparing students for ICPC competitions. While the practice meetings showed a general improvement in the students' problem-solving abilities, as shown in Figure 4.2, the data was not very conclusive due to technical difficulties, low attendance, and the difficulty of the practice problems leading to low problem-solving rates. Despite these challenges, the students showed noticeable improvements in their problem-solving skills, as seen in the number and difficulty of problems solved during practice sessions during the second semester and in their survey results. In the end, the students had a great performance in the ICPC regional, where the main team qualified for the North America Championship and the other two teams performed excellently in division 2, placing within the top quarter of participants despite only practicing for one and a half semesters. While further studies are needed to provide more conclusive evidence, the data we have suggests that our methodology succeeded in improving the students' competitive programming abilities.

Our curriculum was generally successful in improving students' confidence in the concepts discussed during our meetings. However, some areas like greedy strategies,

dynamic programming, and graphs didn't show significant improvement despite being crucial concepts in competitive programming. In our survey, students also expressed the need for more focus on these topics. Although we had dedicated meetings for these concepts, they are challenging to master, and thus more attention is needed. Additionally, the topics discussed in the last two meetings, sparse tables, segment trees, and Rabin-Karp hashing, seemed too advanced for the level of the students. Therefore, we suggest replacing these topics with more practice on dynamic programming, greedy algorithms, and graph algorithms. We also recommend moving the meeting on number theory and combinatorics later in the schedule as the material was too mathematically complex for the early stages in the curriculum.

The format of the program had a weakness in that it did not allocate time for discussing specific problem solutions, instead only focusing on theory during the sessions and leaving students to tackle problems on their own during practice contests. To address this shortcoming, an improvement would be to add a 30-minute segment at the end of the sessions to discuss the first few practice problems, providing concrete examples of how to apply the newly learned concepts and assisting students who may be struggling. In the second semester, we addressed this issue by focusing exclusively on solving ICPC problems, which likely contributed to our success in the regional competition.

Chapter 7

Conclusion

To summarize, our practice methodology was successful in designing a curriculum and conducting a series of competitive programming meetings for Emory students. These meetings resulted in self-reported proficiency increases for the required topics, and Emory's teams performed well in the ICPC regional competition. However, further studies are needed to determine the objective improvement in skill level due to insufficient data during the practice sessions. The topic selection was successful, but slightly imbalanced towards intermediate topics, to the detriment of more fundamental topics like dynamic programming. The lesson format was effective and positively received by the students, although some implementation flaws were noted, such as the practice problems being too difficult. We hope to refine our methodology in the future and test it on a larger sample size of students.

Appendix A

Topic Importance Scores

Importance score	Number of problems	Topic
4.11	9	DFS and BFS
3.74	7	Binary search
3.66	13	Data structures
1.93	3	Sets and maps
1.59	2	Shortest path (Dijkstra / Bellman-Ford / Floyd-Warshall
1.42	9	Geometry
1.34	6	Pruning
1.10	5	Complete search / backtracking
0.89	3	Biconnectivity
0.88	2	DAGs (Topological sort / DP)
0.86	6	Linear sweeps
0.83	1	Maximum matching
0.62	2	Advanced data structures

Importance score	Number of problems	Topic
0.52	2	Suffix array
0.46	3	Segment trees
0.46	3	Binary indexed trees
0.44	1	Prefix sums
0.40	1	Aho-Corasick
0.33	1	Two pointers
0.24	1	Divide and conquer
0.23	2	Advanced number theory
0.15	2	Radial sweeps
0.12	2	LCA
0.12	1	RMQ
0.12	1	LCP
0.12	1	KMP / Z-algorithm
0.12	1	Hashing
0.10	2	LP
0.09	1	MST
0.07	3	Max flow / Min cut
0.06	1	Polygon clipping
0.05		Voronoi diagram
0.04	1	Trie
0.04	1	Tree DP
0.04	1	Parallel BS
0.04	1	Linear algebra
0.04	1	Grundy theorem

Importance score	Number of problems	Topic
0.04	1	DP optimizations (CHT, D&C, Knuth, Alien's Trick)
0.03	1	Meet in the middle
0.02	1	Exchange Arguments
0.02	1	Matroids
0.02	1	Bitmasks
0.00	0	Suffix automaton
0.00	0	Square root decomposition, Mo's algorithm
0.00	0	Sliding window / Nearest largest
0.00	0	SCC
0.00	0	Nondeterministic
0.00	0	Interactive
0.00	0	Inclusion-exclusion
0.00	0	Gray codes
0.00	0	Generating functions
0.00	0	Eulerian paths
0.00	1	DSU
0.00	0	Cayley formula / Prufer codes
0.00	0	Burnside's lemma
0.00	0	Bitset optimizations
0.00	0	Bitmask DP
0.00	0	Binary lifting
0.00	0	Advanced segment trees
0.00	0	2SAT

Appendix B

Problems flowchart paths

The subproblems are indicated by numbers. For example, if problem L reduces to 3 subproblems, they will be indicated by L1, L2, and L3. If it only reduces to one subproblem, that is indicated in the path by the step "Reduce".

B.1 Southeast Regional 2022

A: Not classic → Brute force

B1: Classic (prefix sums)

B2: Not classic → Brute force

C: Reduce → Classic (BFS)

D: Not classic → Not brute force → Find some way to do X → Can't express with states → Constructive

E: Reduce → Not classic → Not brute force → Maximize x such that $f(x)$ holds → Not monotonic → No greedy strategy → Dynamic programming

F: Reduce → Not classic → Not brute force → Maximize x such that $f(x)$ holds → Not monotonic → Greedy or dynamic programming

G1: Not classic → Not brute force → Maximize x such that $f(x)$ holds → Monotonic → Binary search

G2: Classic (articulation points)

H1: Classic (radial sweep)

H2: Not classic → Not brute force → Counting on arrays → Data structures

H3: Classic (Fenwick tree)

I: Not classic → Not brute force → Find some way to do X → Can't express with states → Constructive

J1: Not classic → Not brute force → Pure math → Combinatorics

J2: Not classic → Brute force

K1: Not classic → Not brute force → Find some way to do X → Can't express with states → Constructive

K2: Classic (DSU)

L1: Not classic → Not brute force → Queries → Data structures

L2: Classic (DFS Tree)

L3: Classic (Small to large)

M: Reduce → Not classic → Not brute force → Maximize x such that $f(x)$ holds → Not monotonic → No greedy strategy → Dynamic programming

B.2 North America Qualifier 2022

A1: Classic (sorting)

A2: Not classic → Not brute force → Game theory → Greedy

B1: Not classic → Not brute force → Count the number of ways to do X → No formula → Dynamic programming

B2: Not classic → Not brute force → Pure math → Number theory

C: Classic (sorting)

D: Not classic → Simulate

E: Reduce → Classic (DFS)

F: Not classic → Brute force

G1: Classic (DFS)

G2: Not classic → Not brute force → Count the number of ways to do X → No formula → Dynamic programming

G3: Not classic → Not brute force → Count the number of ways to do X → Formula → Combinatorics

H: Reduce → Not classic → Not brute force → Maximize a function f → No greedy strategy → Dynamic programming

J: Reduce → Classic (Max flow)

K: Not classic → Brute force

L: Reduce → Not classic → No brute force → Geometry

M1: Not classic → Not brute force → Queries → Data structures

M2: Classic (Kruskal)

M3: Classic (Least common ancestor)

Bibliography

[1] Codeforces. Codeforces, 2023. URL `https://codeforces.com/`. Accessed: 2023-01-28.

[2] Codeforces. Codeforces contests, 2023. URL `https://codeforces.com/contests`. Accessed: 2023-01-28.

[3] Daniel Coore and Daniel Fokum. Facilitating course assessment with a competitive programming platform. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 449–455, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287511. URL `https://doi.org/10.1145/3287324.3287511`.

[4] Debeshee Das, Noble Saji Mathews, and Sridhar Chimalakonda. Exploring security vulnerabilities in competitive programming: An empirical study. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, EASE '22, page 110–119, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396134. doi: 10.1145/3530019.3530031. URL `https://doi.org/10.1145/3530019.3530031`.

[5] Tania Di Mascio, Luigi Laura, and Marco Temperini. A framework for personalized competitive programming training. In *2018 17th International Confer-*

ence on Information Technology Based Higher Education and Training (ITHET), pages 1–8, 2018. doi: 10.1109/ITHET.2018.8424620.

[6] Algorithms for Competitive Programming. Algorithms for competitive programming, 2023. URL https://cp-algorithms.com/. Accessed: 2023-01-28.

[7] Jeong-Hoon Ji, Gyun Woo, and Hwan-Gue Cho. An experience of detecting plagiarized source codes in competitive programming contests. *SIGCSE Bull.*, 40(3):369, jun 2008. ISSN 0097-8418. doi: 10.1145/1597849.1384411. URL https://doi.org/10.1145/1597849.1384411.

[8] Antti Laaksonen. *Guide to competitive programming*. Undergraduate Topics in Computer Science. Springer International Publishing, Cham, Switzerland, 1 edition, January 2018.

[9] Antti LAAKSONEN. What is the competitive programming curriculum? *OLYMPIADS IN INFORMATICS*, pages 35–42, 2022. doi: 10.15388/ioi.2022.04. URL https://doi.org/10.15388/ioi.2022.04.

[10] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Ré mi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, dec 2022. doi: 10.1126/science.abq1158. URL https://doi.org/10.1126%2Fscience.abq1158.

[11] Alexandru Rudi. M||e competitive programming github page, 2023. URL

https://github.com/emory-courses/competitive-programming. Accessed: 2023-01-28.

[12] Sudha Subramanian, A. Kumar, M. Nagappan, and R. Suresh. *Classification and Recommendation of Competitive Programming Problems Using CNN*, pages 262–272. 12 2018. ISBN 978-981-10-7634-3. doi: 10.1007/978-981-10-7635-0_20.

[13] USACO. Usaco guide, 2023. URL https://usaco.guide/. Accessed: 2023-01-28.

[14] Tom Verhoeff, Gyula Horváth, Krzysztof Diks, Gordon Cormack, Michal Forišek, Richard Peng, and Jakub Łacki. The international olympiad in informatics syllabus, 2022. URL https://ioinformatics.org/files/ioi-syllabus-2023.pdf.

[15] Nick Wu. Icpc north america qualifier 2022 problem m, 2023. URL https://naq22.kattis.com/contests/naq22/problems/tollroads. Accessed: 2023-03-22.