Competitive Programming in National University of Singapore

Steven HALIM and Felix HALIM

{stevenha, halim}@comp.nus.edu.sg

Department of Computer Science, School of Computing, National University of Singapore

Computing 1, 13 Computing Drive, Singapore 117417

Abstract   University has role in nurturing future programmers. School of Computing (SoC) in National University of Singapore (NUS) offers a special module titled CS3233 – 'Competitive Programming' to nurture its top students. This paper discusses strengths of having such module, the challenges to mount such module, and how we address the challenges with help of some teaching methodology and various Information and Communications Technology (ICT) tools.

1. University Role in Nurturing Future Programmers

The usage of computer has become ubiquitous in many aspects of human life. This increases the need for good software that come from good software engineering and utilizes state-of-the-art data structures and algorithms. On the front-end, scientists also rely more on computers to find better solutions for the ever-growing real world issues. Universities as an institution of higher learning and research have role in nurturing such skilful software developers and computer scientists for future generations.

National University of Singapore (NUS) wants to do well in this aspect. NUS offers basic-level programming methodology modules for many science and engineering disciplines, intermediate-level data structures and algorithms modules mainly for Computer Science students, *and* for the gifted students with special interest in programming contests, NUS offers a module titled CS3233 – "Competitive Programming".

This CS3233 module is the focus of this paper. The NUS module code: CS3233 will be used in this paper whenever we refer to this module. The first author is the coordinator of the current version of the module, which has been offered since ~August 2008. The second author and few other skilled trainers that have prior programming contests experience help the actual running of this module.

CS3233 is a special hands-on programming course geared towards the annual University-level programming contest – ACM International Collegiate Programming Contest (ICPC) [1,2,3]. It has also been customized (i.e. some minor syllabus and assessment tweaks) and used to prepare Singapore International Olympiad in Informatics (IOI) [4] team for IOI 2009 in Plovdiv, Bulgaria. With this module, we attract top students from various faculties in NUS, notably from Computer Science (CS), Computer Engineering, Electrical Engineering, Mathematics, including top high-school students that are strong in Mathematics and Problem Solving, to learn advanced data structures and algorithms normally included in ICPCs or IOIs, and then compete with each other for grades and the rights to represent NUS for ICPC or Singapore for IOI.

This paper is organized as follows: In Section 2, we elaborate some details of CS3233 module: pre-requisites, syllabus, and learning methodology via competition. In Section 3, we present some challenges that we observed when mounting such module. These challenges are addressed one-by-one in Section 4, especially with the aid of various Information and Communications Technology (ICT) tools. Section 5 concludes this paper and present insights on how readers (instructors) can adopt similar module in his / her University.

2. Competitive Programming Module

2.1. Purpose and Syllabus

CS3233 module is *not* for every NUS student! This module is labelled as level-3 (year-3-equivalent) module in NUS. To take this module, most students must satisfy basic prerequisites as listed in the top section of Table 1 plus scoring well in basic and intermediate programming / algorithm modules. However, students with good experience in national or even international programming competition in the past, e.g. National Olympiad in Informatics (NOI) or IOI, can be allowed to take this module from their 1st year with special permission from the instructor.

This filtering is necessary as the syllabus of this 1 semester module shown in Table 1 (13 teaching weeks * 3-4 hours per week) has materials from *several* advanced algorithm classes in CS curriculum, but taught in hands-on and condensed manner, i.e. we teach "single source shortest path algorithm on weighted graph" like Dijkstra algorithm, but rather than going depth with proofs and theories on why such algorithm works, we emphasize more on the efficient implementation of such algorithm, give some references, and then show lots of recent programming contest problems that utilize such algorithm. Without past programming contest experience / good scores in previous algorithm modules plus enthusiasm towards problem solving, students will find this module too difficult and too heavy.

Explanation for Table 1: Column 'references' lists some book chapters, papers, or Internet resources related to the topic. Column 'UVa / LA' shows the problem numbers in UVa online judge [20] / Live Archive [19] website that are of that par-

ticular topic. The syllabus is organized into 13 teaching weeks and focuses on topics that are frequently contested in recent ICPCs [5] or IOIs [6]. As IOI is targeted to high school students, it has *lesser* syllabus compared to ICPC. Topics that are not inside the official IOI Syllabus [6] are listed in *italic*.

Tips: Table 1 can also be used as a checklist for gauging student's strengths. Ask student to give a score to each row (topic) in the table using values like this: 1-never heard about it, 2-have heard about it but not sure about the details, 3-know it, but have never use or code it, 4-have use or code it, 5-very familiar and confident with it. The more the total, the better!

This Table 1 can also be used for measuring ICPC team strengths by taking max(score of student1, student2, student3 in the team) for each row.

Table 1. CS3233 Syllabus as in Academic Year 2009/2010.

Prerequisites: Basic Data Structures & Algorithms	References	UVa / LA
Familiar with Sequence, Selection, Repetition, Recursion	[11] Ch3	161,155
Familiar with ≥ 1 programming language: C / C++ / Java	[11] Ch3	146
Familiar with basic algorithm complexity (big O) analysis	[14] Ch0	108
Array / Matrix (1D / 2D) / Vector / Stack / Queue	[11] Ch3,4	551,200
Basic ideas of: Heap / Priority Queue, Binary Search Tree	[12] Ch6,12	484,487
Graph (Adjacency Matrix / Adjacency List / Edge List)	[17] Part 5	291
$O(n^2)$ Sorting Algorithms: Bubble / Selection / Insertion Sort	[11] Ch8	299
O(n log n) Sorting Algorithms: Heap / Merge / Quick Sort	[12] Ch7	10194
Special Purpose Sorting Algorithms: Counting Sort	[12] Ch8	11462
Week 1: Problem Solving Paradigms	**References**	**UVa / LA**
Overview, *Team strategy in ICPC*, Individual strategy in IOI	[31,5,6]	LA4146
Ad Hoc / Simulation	[31]	100,119
Complete Search / Brute Force / Recursion (Backtracking)	[31]	331,193
Divide (/ Decrease) and Conquer	[12] Ch4	LA4104
a. Binary Search, Ternary Search	[32]	LA2452
b. Counting Inversions using Merge Sort	[33]	11495
Plane Sweep	[12] Ch35	105
Week 2: Libraries and Advanced Data Structures	**References**	**UVa / LA**
Library Implementation: C++ STL Set / Map / PriorityQueue / Algorithm: Sort, etc	www.cppreference.com	146,10226 11549
Union-Find Disjoint Set	[12] Ch21	10583
Fenwick (Binary Indexed) Tree	[27]	11423
Augmenting Data Structure: Segment / Interval Tree	[12] Ch14	11402

Week 3-5: Greedy & Dynamic Programming (DP)	References	UVa / LA
Greedy: optimal substructure & greedy property	[14] Ch5	120,10020
Introduction to classical greedy algorithms		
a. Kruskal's for Minimum Spanning Tree (MST) (basic)	[14] Ch5	908,10034
DP: optimal substructure & overlapping subproblems	[14] Ch6	116,10337
Top-down versus Bottom-up DP		
Introduction to classical DP solutions		
a. Coin Change (DP variant)	[11] Ch11	147,357
b. Subset Sum	[31]	562,574
c. Longest Increasing Subsequence (LIS) in O(n log n)	[14] Ch6	481,497
d. 0-1 Knapsack	[14] Ch6	990,10130
e. Matrix Chain Multiplication (MCM)	[14] Ch6	442
f. Optimal Binary Search Tree	[12] Ch15	442
Multi-dimensional DP states, various DP formulations	[27]	LA4143
On-the-fly-computation to reduce 1 dimension	[27]	LA4106
DP on Tree	[14] Ch6	LA3685
Speed Up DP computation: Knuth-Yao, Convex property	[27]	
Week 6-9: Graph Algorithms	**Reference**	**UVa / LA**
Depth First Search (DFS)	[12] Ch22	10452
a. Flood Fill	[11] Ch12	352,469
b. Longest Path (Diameter) in Tree	[32]	10308
Topological Sort	[12] Ch22	124,10305
a. Longest Path in a Directed Acyclic Graph (DAG) + DP	[14] Ch6	103,10000
Tarjan's Algorithm to find Articulation Point / Bridge	[32]	315,10199
Tarjan's Algorithm to Find Strongly Connected Components	[32]	11504
Kruskal's / Prim's Algorithm for MST Problem (advanced)	[12] Ch23	11631
Breadth First Search (BFS) for unweighted SSSP	[12] Ch22	336
Dijkstra for weighted Single Source Shortest Path (SSSP)	[12] Ch24	421,11492
Bellman Ford for weighted SSSP with negative weight	[12] Ch24	558
Floyd Warshall for weighted All Pairs Shortest Path	[12] Ch25	186,11463
2nd / k-th best shortest path / MST	[33]	10600
Euler Path / Circuit	[32]	117,302
Ford Fulkerson / Edmonds Karp / Dinic for Network Flow	[27]	
a. Max Flow / Min Cut	[12] Ch26	820,11506
b. Maximum (Cardinality / Weighted) Bipartite Matching	[12] Ch26	670,10092
c. Maximum (Cardinality / Weighted) Independent Set	[32]	11159
d. Minimum Cost Maximum Flow	[28]	10594

Hungarian Algorithm (Kuhn Munkres)	[32]	10072
Edmonds' Blossom Shrinking for General Matching	[32]	11439
Floyd's Cycle Finding Algorithm	[32]	408,11549
Tarjan's Least Common Ancestor (LCA)	[32]	10938
Week 10: Mathematics	**Reference**	**UVa / LA**
Prime Numbers: Generation (Sieve), Testing, Factoring	[12] Ch31	406,583
Big Integer, Java BigInteger API	java.sun.com	324,623
Basics of Modulo Arithmetic	[12] Ch31	374
Number Theory: GCD / LCM / Divisibility	[12] Ch31	294,332
Linear Algebra: Gaussian Elimination	[14] Ch7	10109
Floating Point Precision Issues in Programming Contest	[11] Ch7	10060
Base Number Conversion	[11] Ch7	343,377
Factorial / Fibonacci / Josephus / Polynomial / Trigonometry	[11] Ch7	324,495
Week 11: String / Bioinformatics (Guest Lecture)	**Reference**	**UVa / LA**
String Alignment (Edit Distance / Levenshtein)	[14] Ch6	164
Longest Common Subsequence (LCS)	[12] Ch25	10405
Suffix Tree / Array	[32]	11512
Knuth Morris Pratt (KMP)	[12] Ch32	11475
Aho Corasick	[32]	10679
Parsing, Backus Naur Form (BNF) Grammar	[32]	325
Week 12: Computational Geometry (Guest Lecture)	**Reference**	**UVa / LA**
Geometric Tests: CCW	[12] Ch33	478
Area of Arbitrary Polygon	[33]	11447
Line Segment Intersection	[12] Ch33	866
Closest Pair of Points	[12] Ch33	11378
Convex Hull	[12] Ch33	681
Week 13: Final Contest (Miscellaneous Topics)	**Reference**	**UVa / LA**
Game Playing: Minimax, Board / Card / Chess / AI Games	[33]	162,750
Advanced Search Algorithms: A* Search, Branch & Bound	[33]	

Note: It is true that there are many more known data structures that are not included in Table 1, e.g. Linked List and its variants, Hash table and hashing techniques, variants of balanced BST: AVL, Red-Black-Tree, etc, Fibonacci heap, Leftist heap, etc. Other than the fact that we cannot squeeze everything into a single semester, the other reason is that they are rarely used in programming contests (and even if they are used, usually they can be replaced using some library, e.g. C++ STL tricks). There are also many more algorithms / classical problems that are not currently included in Table 1.

Customizing this module for ICPC and IOI is quite straightforward. Simply omit the italic parts in Table 1 and change the programming problems to IOI style (i.e. judge solutions after contest and give partial marks).

Teaching or motivating students with a certain Computer Science topic can also be done via programming problems, e.g. UVa [20] problem number 336 – 'A Node Too Far' teaches about 'Time To Live' taught in Computer Network, UVa 11512 – 'GATTACA' teaches students about bioinformatics, UVa 116 – 'Unidirectional TSP' is an introduction to NP-complete problems where a general version of the problem must be solved with more sophisticated algorithms. In fact, the authors' research works are partly on these NP-complete problems [7,8,9].

2.2. Teaching Mode

The teaching mode of CS3233 is not just via lectures, but – as it name implies – via competition! Throughout the semester, we organize several local contests for the students, pitting them against each other to solve a given set of programming problems related to the materials that have been recently taught. In CS3233, these students are fighting for their own grades that will be reflected in their University diplomas! We believe that competition like this is good to push learning as it is human's nature to achieve the best, especially if his grade is put on stake.

We have about 4-5 local contests per semester of about 4-6 problems each, an example of one such contest is shown below. We elaborate how the problem set is chosen and describe the learning insights obtained by conducting such contest.

2.2.1. NUS ICPC Selection Test 1, 12 September 2009, 9am-2pm, 23 students

This contest is the first of three tests used to determine the students who should be selected to represent NUS in ACM ICPC regional contests 2009. These are the brief description of the 6 problems used in the contest and the *expected learning outcome* of each of them.

A: This problem requires the usage of binary search to search a value on a tree data structure with special ordered property from root to leaf. While binary search on a 1-D array is a classic algorithm, applying it on a tree data structure is not common. Naïve algorithm is to apply sequential search from root to target node, but that is expected to get a frustrating Time Limit Exceeded (TLE) response. Students have to realize using complexity analysis that without using binary search, it is impossible to pass the time limit given the huge size of input data.

B: A classical Min Cost Max Flow (MCMF) problem. However, we only expect students that have coded such MCMF solution before that can solve this problem. This is to encourage students to build code libraries and bring hardcopy to contests as this is allowed in ICPC. Students without such library code will end up leaving this problem to the very last or give up.

C: Hard DP problem where normal (straightforward) solution is $O(n^3)$ but utilizing convex property, this can be speed up to $O(n^2 \log n)$ or $O(n^2)$. The input size is sufficiently large to make $O(n^3)$ algorithm receives a Time Limit Exceeded (TLE) response. This type of DP problem emerges recently in recent programming contests so only the best student is expected to solve this problem as his last problem, and if he managed to do so, he likely wins the contest.

D: Variant of classical DP problem: Longest Common Subsequence (LCS) problem. But this time, we want to count number of distinct LCS of two given strings. Students must be able to tweak the known DP solution for LCS for this problem. It is expected that students with strengths in DP techniques will attempt this problem in the middle of contest, after easier problems have been solved.

E: Ad-hoc sorting and string formatting problem to display F1 Championship table from raw individual race results. This problem is very easy if student knows how to use C / C++ to format the string output and use STL sort algorithm to sort the data. It is expected that almost all students will be able to solve this problem. It is just a matter of who can solve it faster, thus have more time to solve harder problems in the contest.

F: Data structure problem, maintain 3 BSTs with 3 ordering criteria. Easy problem if student knows how to use C++ STL libraries. Problem E and F are expected to be solved by most students during contest time.
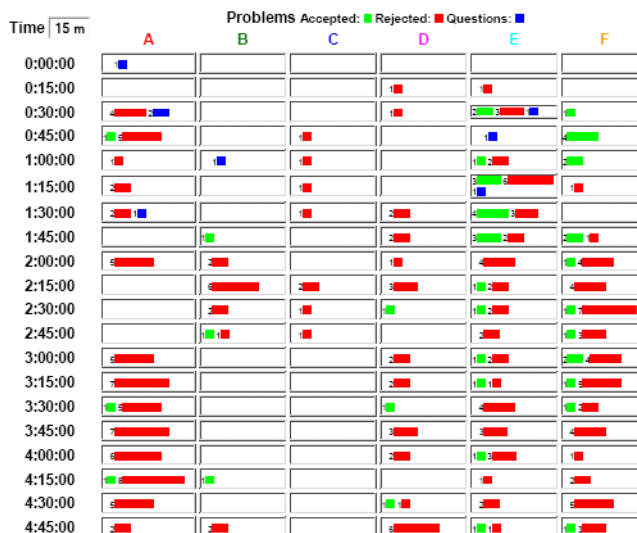

Figure 1. Contest Evolution per 15 Minutes

Figure 1 shows a visualization of contest dynamics as it unfolds. As expected, many students straightforwardly attempted problem E and F, as they are the easiest. Problem F has clearer description, thus the first 7 submissions were Accepted without problem. Problem E has longer description and a bit complicated output

format and thus several students raised questions / clarifications, obtained several Wrong Answers, before eventually got it Accepted. Some students address his Presentation Error issue by creatively using tool like fc (file compare) in Windows or diff in UNIX to compare their program's output with the sample output. Problem C was attempted by few students, but midway through the contest, this problem was abandoned by all students – a note that we should teach them this DP technique in the future. Problem B and D have normal characteristics, i.e. they were solved only by few better students after few attempts in the latter phase of contest. Problem A happened to be the most frustrating problem as naïve solution was easy to get but the expected solution was not, as illustrated by a lot of Wrong Answer / Time Limit Exceeded responses in the last two hours for problem A.

Contest with mixed problem types and difficulty levels like this expose the strengths and weaknesses (as well as the luck) of different students. The snapshot of the result of this contest is shown in Figure 2. We can see tight result where only close time separates the students (especially rank 1-2 and 8-10).

#	Country	Team	A	B	C	D	E	F	Solved	Points
1		NUS Ngo Minh Duc	4:17:09 (0)		------ (1)	2:30:19 (1)	0:40:12 (0)	0:51:16 (0)	4	8:38:56
2		NUS Loh Bo Huai, Victor	3:39:54 (1)	1:51:53 (0)			0:43:27 (4)	0:57:12 (0)	4	8:52:26
3		NUS Bi Ran	------ (2)	2:50:14 (3)		3:39:46 (8)	1:17:47 (0)	0:51:27 (0)	4	12:19:14
4		NUS Nguyen Hoanh Tien	0:48:26 (0)	------ (1)		------ (6)	1:31:48 (2)	1:46:28 (0)	3	4:46:42
5		NUS Ahmed Shafeeq		4:29:39 (0)			1:31:09 (0)	1:47:35 (0)	3	7:48:23
6		NUS Adhiraj Somani			------ (2)	4:44:18 (3)	2:33:47 (0)	0:31:05 (0)	3	8:49:10
7		NUS Lai Wui Shing	------ (13)				1:28:43 (2)	2:05:27 (0)	2	4:14:10
8		NUS Huang Wenjun	------ (2)				1:45:14 (0)	3:06:36 (1)	2	5:11:50
9		NUS Dang Cao Khoa		------ (2)			2:18:14 (1)	2:48:45 (0)	2	5:26:59
10		NUS Nguyen Duc Phong	------ (4)			------ (3)	1:54:35 (0)	2:33:55 (3)	2	5:28:30
11		NUS Liau Yung Siang					1:55:27 (1)	3:38:48 (3)	2	6:54:15
12		NUS Doan Manh Hung	------ (5)				4:57:16 (10)	0:54:04 (0)	2	9:11:20
13		NUS Daniel Aris					3:21:11 (3)	4:49:12 (4)	2	10:30:23

Figure 2. Selection Test 1 Result

The advantages of having such special module in a CS department of a University are that it can be used to distinguish the minority talented programmers from the majority of the cohort that are taught using standard CS syllabus, train them more intensively, give each other comparably strong 'sparring partners' to keep them motivated, and ultimately the top ones will perform better in ICPCs, bringing good reputation to the University.

A dedicated module also allows University administration to specifically allocate resources, e.g. the instructors, teaching assistants, usage of computer laboratories, etc whereas these things may be difficult to obtain if programming contests are just listed as optional "Extra Curricular Activities" in that University.

3. Challenges to Mount Competitive Programming Module

Albeit the positives of having such competitive programming module in a University, we have identified four challenges in mounting a module such as CS3233: 1). manpower, 2). students, 3). assessment, and 4). course materials challenges.

The first real challenge is the manpower to teach the module. The instructor of such course himself needs to have a (very) strong background in algorithmic / programming skills. Preferably, he himself has some programming contests experience in the past. Tenured CS professors may know a lot of algorithms, but if he has not done the coding himself for a long time (although he can always asks his assistants to do the coding job), he will have serious difficulties teaching such module as his solution may be not implementation-friendly whereas competitive programming is a combination of both algorithm *and* implementation! Then, it is hard even for a genius professor to know everything in depth about each topic listed in Table 1, as they are usually specialized in their research field.

Finding appropriate Teaching Assistants (TAs) is also another issue. In normal "programming methodology" course, we can pick anyone with sufficient programming background and ask him to lead a small class of basic programming course and he will be just fine. In competitive programming module, this is not possible as he cannot teach how to solve problems that he cannot solve yet! Preferably, we want to have an ex-ICPC world finalist as TA, which may be a luxury for most Universities.

The second challenge is about the students. Getting top students in University to join such highly competitive module is not easy. In students' grade-conscious term: "hard work needed to fight the other good programmers in class but A+ is not guaranteed". Another challenge is that the student's initial strengths are uneven. Some have strong programming background, and some are not. Some are IOI medallist whereas some have just completed "basic programming methodology" class. Some know various advanced algorithms whereas some barely heard them. It is very hard to teach an unbalanced class like this. Pitching the difficulty level of the module too hard will 'kill' the weaker ones whereas too easy will make the top students bored and unmotivated.

The third challenge is the assessment, which in this case: programming contests. We have to produce original problem set for the local contests – which is not easy. If we simply use past contest problems that are available in Internet verbatim, we may run into issues as some students may have already solved such problem before, thereby gaining advantage (and grades) over their classmates.

The fourth challenge is course materials. Typical programming or algorithm text books available are mainly about 'basic data structures' and 'basic classical algorithms' [12-18], which are not enough if student wants to do well in today's highly competitive programming contests. While there are dedicated books like Programming Challenges in 2003 [10], Art of Programming Contest [11] in 2006, they are considered 'outdated' and cannot keep pace with the novel programming problems that keep appearing in recent programming contests.

4. Addressing the Challenges:
Teaching Methodology and Usage of ICT Tools

In 2008, NUS is lucky to have 3 ex-ICPC World Finalists arriving from other Universities (1 new PhD student and 2 new Research Assistants - including the second author) to add into the pool of existing teaching staffs; their names are listed in the acknowledgements. This, coupled with the timing of first author completion of his PhD study, form the base for running the competitive programming module in NUS. But this one time lucky should not be taken for granted, but instead, University must keep grooming local talents to achieve continuity.

For the domain-specific knowledge, we also grateful for the willingness of several professors and lecturers in NUS to devote their time to share their expertise to CS3233 students, see rows indicated as 'guest lecture' in Table 1 (i.e. bioinformatics and computational geometry).

For the second challenge, we advertise this module University-wide at the beginning of an academic year. This is because top programmers do not always study Computer Science (CS) discipline. Some are studying Engineering (Electrical or Computer Engineering) or Science (usually Mathematics). We allow them to take this module as 'cross-faculty' module to satisfy their graduation requirement. We also ask ex-students to 'advertise' this module to their juniors.

So far, with 'active students recruitment strategy' like this, coupled with potential benefits of representing NUS / Singapore in ICPC / IOI – which is good for student's CV – this module has managed to attract top students in its recent offerings. While in the past finding 3 good programmers to represent NUS in ICPC is hard, now we have a pool of about 15-20 students per year to choose from.

To address the class performance imbalance issue, we adopt a strategy to split the class into two: 'experienced' and 'standard' groups. Students will compete with other students with roughly similar level of expertise. This two-tiered approach allows a more balanced competition and reducing the fear of genius-but-still-new-programmer-and-grade-conscious students in taking this module.

For the third challenge of preparing original problems for local contest, we use a combination of 1). original efforts by teaching staffs, 2). paraphrasing old problems so that students are not immediately aware of the problem even if they have solved it before, and 3). have a homework titled "be a problem setter" where we ask current students to share each other's past knowledge, e.g. from their respective countries' NOI or IOI, by setting one original problem each for their juniors. Instructors will then verify their problems and it is not common to find student setting problem type that the instructor was not aware yet. If he has 20 students, he will have ~20 original problems to be used for the *next iteration* of this module.

For the fourth challenge regarding course materials, we admit that textbooks cannot keep pace with the increasing level of difficulties and novelties of recent programming contest problems. Thus, the best way is to use the recent ICPC and IOI problems themselves as the course materials on top of existing textbooks!

There are abundant Information and Communications Technology (ICT) tools to achieve this purpose: the ACM ICPC Live Archive [19] which archives recent ICPC problems – as it name implies, UVa Online Judge [20,21] especially the ever growing Volume Contest, Hunting UVa Problems [22] by the second author, Methods to Solve [23] by the first author, and many others tools that are not built by the authors like: UVaToolkit [24], π Algorithmist [25], Mooshak [26], TopCoder [27], Igor's Code Archive [28], etc. How each of these individual ICT tool can be helpful in the actual running of competitive programming module is elaborated in the boxes below.

4.1. ICT Tools Used in CS3233

4.1.1. ACM ICPC Live Archive and UVa Online Judge

Figure 3. Screen shot of ACM ICPC Live Archive Webpage

Figure 4. UVa Online Judge Logo

The core tool to conduct CS3233 module is a library of programming contest problems! There are several online judges available in the Internet. However, for CS3233, we use ACM ICPC Live Archive (http://acm.uva.es/archive/nuevoportal) and UVa Online Judge (http://uva.onlinejudge.org). Sample problems and homework (see Table 1 column 'UVa / LA') are taken from these two online judges.

4.1.2. Felix Halim's UVa scripts: Hunting Problems, Coaching Tool, etc

The UVa tools in http://felix-halim.net was first born because the second author (Felix Halim) had difficulties in filtering the enormous amount of UVa problemsets to be solved during his own self-training. At that time, there were more than a

thousand problems in UVa site. As a new user that was very interested in having fun in solving problems, selecting the next problem to solve from a vast amount of choice can be disheartening. One big reason is because there was no immediate guidance on the difficulties of a problem. If the next problem selected was too hard, he could lose his interest in solving more problems later.

Fortunately, UVa provides statistics of each problem (such as the number of Accepted, Wrong Answer, etc...) that can be used to roughly tell the level of difficulty of the problem. Therefore, the second author created a simple script to extract the UVa statistics and generate the problem ranking based on the number of Accepted users for that problem.

This tool is useful for students' self-learning. Having a problem ranking on difficulties is important for the students in deciding what problem to solve next. Solving problems with increasing difficulties can be more rewarding for beginners since there is no point solving harder problems if they cannot solve the easy ones. However, as the user becomes more experienced, this tool becomes less important and less accurate since at some point there will be no more "easy" problems and the "hard" problem can be easy for some students and hard for the others simply because they have not encounter such type of problem before (e.g. Max Flow problems can be very hard if student solve it *for the first time*, but it will be easier next time). At this point, the problems have to be categorized by algorithm type (as in Table 1). Then student can improve his algorithms skills by solving problem type that he has not solved before.

The tool to rank the problems based on the number of Accepted users is called "Hunting UVa Problems!". It summarizes, processes, and presents the raw statistics given in the UVa site in a more informative way. The tool gives "information" rather than "data" to the users. For example, the tool gives information about what are the 20 next easiest problems to solve, what is your progress over the years, user's personalized world rank, and the performance of user solutions compared to other users. All the information is densely packed in a single page. The tool quickly becomes popular among UVa users.

The Hunting UVa Problems acquires the raw statistic data directly from UVa site. Since the tool does not have access to UVa database, it has to do screen-scrapping technique to parse and process the HTML statistics from UVa user statistic pages which is not really efficient for both the tool and UVa site. To mitigate this issue, the tool was designed to minimize contact with UVa site by updating only the users that are interested to use the tool rather than the whole users in UVa and cached the user statistics. The consequence is that the user data in the cache can be stale and out of date. This is not a concern, since the corresponding user can manually update his statistics directly from the tool user interface. Since the user knows when his statistics changes (when he solves a new problem), the user will likely update his statistics only when he solves a new problem, making the update mechanism a lot more effective.

Below are few screenshots for the features of this tool. Figure 5 displays a user statistics acquired from UVa site. Using the date of the Accepted submission, the Accepted problem number is colored differently. Red for the problems accepted

less than 2 days ago, green for less than 1 week, blue for less than one month, black for more than one month. By coloring the numbers, we can see the activity of the users whether he / she is recently active or not.

Solved : **58**, Tried : **85**, Submissions : **262**

100 **102** 113 136 146 **160** 272 294 299 **357** 369 374 382 **443** 445 446 458 **483 488 494** 495 530 543 575 579 **583** 591 **673 674** 686 834 **10006** 10018 10035 10038 10041 10055 10071 10079 10082 10110 10127 **10190** 10222 **10235** 10260 10302 **10338** 10340 **10346** 10370 10424 10450 10469 **10783** 10924 10970 **10994**

Figure 5. User Statistics

Figure 6 shows the core of the "Hunting UVa Problems!" tool. The table in Figure 6 ranks the problem difficulty (roughly by the number of distinct Accepted user) by volumes, or for all available problems. The "dacu" in the table column header means distinct accepted user, "nacu" means number of accepted submissions, "%dacu" is dacu/nacu*100%. Similarly, "nos" means number of submissions, "anos" means number of accepted submissions and "%anos" is anos/nos*100%.

"dacu" might not be the best way to rank the difficulty of the problem since other factors such as number of wrong answers, time limit exceeded, number of submissions can contribute to the difficulty and trickiness of the problem. A more accurate ranking of the problem can be generated by aggregating the statistics for that problem. However, "dacu" alone suffices to make most the users happy.

The next 20 problems are presented by volumes. It may be the case that even the easiest 20 problems are hard and the user is stuck with all of them. It is encouraged to look for other problems in other volumes for some time and get back later if the user has become more experienced.

These are the next 20 problems for you by Volumes!!!

Click on the problem's volume on the right to show the next 20 problems for that volume.

No	Number	Problem Title	nos	anos	%anos	nacu	dacu	%dacu	Volumes
1	11332	Summing Digits	2121	1383	65.25%	1192	1156	97.06%	v1 (89/100)
2	489	Hangman Judge	6650	1359	20.45%	3050	1127	36.98%	v2 (45/100)
3	11417	GCD	1902	1420	74.71%	1081	1050	97.22%	v3 (61/100)
4	11461	Square Numbers	1774	1042	58.74%	914	876	95.95%	v4 (72/100)
5	11219	How old are you?	2537	972	38.35%	905	791	87.51%	v5 (59/100)
6	10963	The Swallowing Ground	4420	1238	28.01%	915	767	83.93%	v6 (57/100)
									v7 (42/100)

Figure 6. Next 20 Problems

Figure 7 shows the progress of the user year by year since his / her registration on the UVa site. 'Progress Chart' like this helps motivate students to do better. Figure 7 is actually the year by year progress of the second author. There is a steep increase in number of solved problems around late 2006. This is where the author was very active in practicing for ACM ICPC regional Kaohsiung 2006. The training was to solve as many problems in UVa as possible to improve coding skills (efficiency and accuracy) by solving many easy problems (using no other than this "Hunting UVa Problems!" tool). The intensive practice was proven to be helpful and became the deciding factor for the champion of the regional. The second author's team managed to solve 7 problems and became the champion in that regional by having lower time penalty points from the runner-up, thanks to the intensive practice on coding accuracy, speed, and efficiency using this tool.
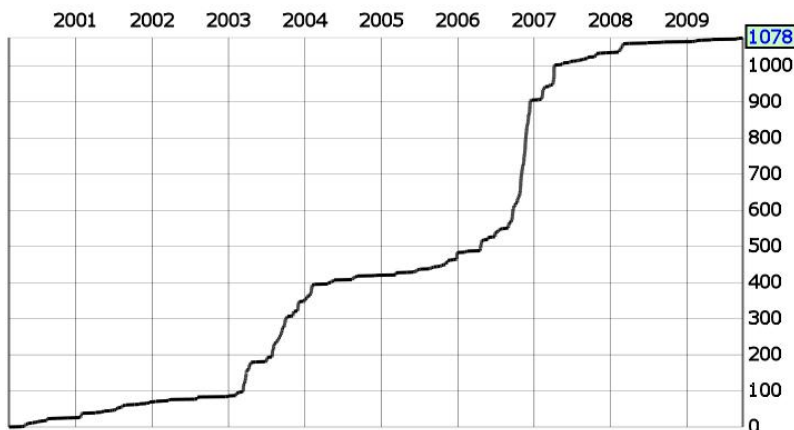
Your progress in UVA, year by year!!

Figure 7. Second Author's Progress in UVa since 2000-2009

Figure 8 shows the user ranklist among other users that uses this tool. The old UVa user statistics contains "country" attributes for the users so that a ranklist by country can be made. The ranklist by country can be a good motivator for the users to be the top of his / her own country. This is because the users from the same country usually knows each other and it will be more fun and challenging to compete with the user' friends rather than with strangers.

Your ranklist in World

Rank	Name	Solved	Subs	User ID
60	Dong Zhou (Loner)	1089	2547	291
61	Emilio Jose Garcia Tormo (EmilioJose)	1081	13157	371
62	Sqr5 team (Sqr5)	1080	2046	1338
63	**Felix Halim (felix_halim)**	**1078**	**5185**	**339**
64	Yeomin Choi (Toivoa)	1075	2931	2868
65	N.M. Mosharaf Kabir Chowdhury (rawc)	1066	1937	5237
66	Andrey Adaikin (anadan)	1064	2681	12149

Figure 8. World Ranklist

Figure 9 shows the list of problems that have been solved by the users but not efficient enough since there are other users that can solve the problem in a much faster runtime. A difference of more than 3 seconds in runtime is considered as a huge difference. It gives a hint that a better (faster) algorithm exists to solve the problem and it gives motivation to the user to revisit his solution and improve it. As mentioned earlier, competition is the nature of people. A user ranklist for each problem based on the users' solution runtime can be a good motivator for competitions among users to produce the fastest performing solution for that problem. Based on the second author's experience, it is fun to beat each other's runtime. This involving tweaks in the code, thinking a faster algorithm, and promoting discussions among the users about the tricks to make the solution runs faster.

Improve your running time!!

Below is the top 20 list of problems that you have solved
sorted by running time difference with the (cached) best solution.

No	Problem	Title	Your's	Best	Difference
0	10304	Optimal Binary Search Tree	0:28.986	0:00.230	0:28.756
1	10181	15-Puzzle Problem	0:20.869	0:00.020	0:20.849
2	10040	Ouroboros Snake	0:09.766	0:00.000	0:09.766
3	10373	The Brick Stops Here	0:09.295	0:00.000	0:09.295
4	619	Numerically Speaking	0:09.125	0:00.010	0:09.115
5	11120	Very Funny Mr. Feynman	0:08.928	0:00.170	0:08.758

Figure 9. Improve Your Running Time

Another tool that promotes competition among users is "Statistic Comparer" tool. With this tool, user can select a number of users (limited to 5) to be compared. For example, if the user wants to know what are the problems that user A has solved but B has not, what are the problems that user A and B have not solve, what are the problems that have been solved by A and B (as in Figure 10, it shows the problems that have been solved by the first and the second author together), etc...

Result of *felix_halim* & *stevenhalim* : (205 items)

100 101 102 104 105 106 107 108 110 111 112 113 114 115 116 117 118 120 121 122 128 130 133 136 138
232 253 255 256 259 260 263 264 271 272 276 278 280 291 294 297 299 300 305 306 311 315 320 324 325
481 483 484 485 486 487 488 492 494 495 496 497 498 499 541 558 579 583 591 610 673 674 686 694 793
10192 10195 10199 10222 10226 10260 10285 10295 10300 10327 10330 10337 10340 10346 10369 10370
10810 10815 10852 10878 10879 10902 10921 10924 10929 11000 11044 11057 11059 11067 11137 11172

Figure 10. Statistic Comparer

Another UVa tools was created after the second author became a coach for ACM ICPC teams (for Bina Nusantara University), hence the tool is named "Coaching Tool" (Figure 11). The tool helps the coach in selecting a subset of problems in UVa site to be solved by students in the ACM ICPC training. This tool uses the code base from the "Hunting UVa Problems!" in acquiring the user statistics. Then, it takes other inputs: the list of problem sets selected by the coach and the list of students, and then displays a table showing which students have solved which problem sets. This eases the burden in maintaining the statistics and keeping track of the students progress during the training.

Set names	#1			#2A		
Authors	198	512	735	707	10364	10501
244 – Suhendry Effendy (suhendry) (13/13)	0	160	113	545	39	74
339 – Felix Halim (felix_halim) (13/13)	2	584	33	1775	428	480
637 – Andrian Kurniady (kurniady) (3/3)				346		
667 – Timotius Sakti Wirjawan (Timo) (10/10)			5258	129	1406	
650 – Eko Wibowo (marcadian) (0/0)						
1027038 - evan leonardi (13/13)	10	342	5695	293	365	400
1081 – Lego Haryanto (turuthok) (4/4)					160	

Figure 11. Coaching Tool

ACM ICPC Live Archive provides a collection of past years ACM ICPC regional and world final contests. This resource is interesting for teams that are preparing themselves for certain regional sites by practicing the past regional contest problems for those sites. Regional contest problems on some regional sites sometimes contain several interesting new problems worth looking at. Similar with the "Coaching Tool", Figure 12 shows the tool for Live Archive and present it in a real-time contest settings. The Live Archive coaching tool has been used by NUS for the ACM ICPC training to practice various past regional contest problems.

Contest Ranklist [Update All] Refreshing in 66 secs.

No	Author ID and Name	A	B	C	D	E	F	G	H	Solved
1	18871 – Nguyen Hoanh Tien	AC		AC	AC				AC	4
2	20701 – Trinh Tuan Phuong	AC		AC	AC			TL	AC	4
3	20618 – Liau Yung Siang	AC		AC	AC					3
4	20684 – Wu Biao	AC		AC	AC	WA				3
5	20689 – Adhiraj Somani	AC		AC	AC					3
6	20690 – BiRan	AC		AC	AC			WA	WA	3
7	20700 – Hung Doan	AC		AC	AC				WA	3
8	3326 – Victor Loh	AC		TL	AC				WA	2
9	17976 – Ahmed Shafeeq Bin Mohd Shariff	AC		ML	AC					2
10	20659 – Lai Wui Shing	AC			AC				WA	2

Figure 12. ACM ICPC Live Archive Contest Ranklist

For the future, there are some features that are worth to have. Including virtual contest (similar to virtual contest feature in Tianjin University TJU online judge [34]) where several problems from the archive can be used to make a "virtual" contest for practice. As well as re-running the past contest with original user that participated in that contest becomes the "shadow" to simulate as if the contest is running with their presence. This can be used as a measure how good our team compared to those who participated during that past contest.

4.1.3. Steven Halim's Methods to Solve

'Methods to Solve' is a website that contains hints on how to solve **>1000** out of ~2500 problems in UVa (~40%). Probably >20.000 different hosts have visited this page since 2001 with approximately 100 hits per day. Even after removing the hits from crawler bots, this is still substantial. This popularity is partly because it is cited by UVa online judge, ICPC wiki page [29], QuestToSolve [30], etc.

The first author (Steven Halim) created this website after suggestion from his friend Ilham W. Kurnia. The main purpose of this website is to assist newbies to intermediate programmers / students to overcome the learning barrier as the first author has encountered before. The hints for those 1000 problems allow beginner students to avoid dead ends, especially in handling the tricky cases.

   This website also has algorithm notes or links that can point students to relevant materials to continue their pursuit to solve the problem. For example, see a snapshot of a hint of UVa problem 11504 below.

11504 - Dominos

The question has to be modeled as a directed graph. We cannot search for nodes that have no in-degree edges and DFS from there, because the graph is not acyclic. So we have to decompose the graph to a DAG (directed acyclic graph) first before we can search for the number of nodes with no in-degree edges. So to be able to do it, we have to decompose the graph to strongly connected components. Once the directed graph is transformed into another graph that composes only of the strongly connected component, the graph becomes a DAG. However, is there really a need to do DFS after all the strongly connected component is found? The answer is no. All we are seeking for are the number of strongly connected component that does not have any in-degree edge from a node that is outside the component. To count the number of such strongly connected components, use Tarjan's algorithm:

http://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm

VOL: AVAILABLE HINTS		VOL: AVAILABLE HINTS	
	1: 52		100: 44
	2: 27		101: 43
	3: 51		102: 31
	4: 71		103: 30
	5: 45		104: 27
	6: 36		105: 36
	7: 36		106: 53
	8: 48		107: 30
	9: 29		108: 30
	–		109: 37
	–		110: 44
	–		111: 35
	–		112: 40
	–		113: 30
	–		114: 41
	–		115: 55
	–		116: 00
~395 hints so far		~606 hints so far	

Alternative view: by category (but not up to date)

Total: > **1000+** hints available.

Some of the hints are detailed but some are just general hint. I cannot maintain the same level of writing throughout all hints because some of the hints are written by other people, not by me.

Note: please accept the fact that you will be spoiled a lot if you read the hints before attempting the problem by yourself :$.

Google™   [   ] [Search]

Find your hints here: ○ Web   ● www.comp.nus.edu.sg

Figure 14. Main Page of "Methods to Solve"

It takes a lot of effort to keep maintaining such huge amount of hints in this website. However, this website is not purely a one man effort. Various people from all over the world have e-mailed their hints to be added into this website and ex-CS3233 students have contributed over 300 extra hints in recent semester. However, to catch the pace of ever growing Volume Contest in UVa online judge, a better way must be used, perhaps using wiki style as in π algorithmist [25].

In CS3233, many homework problems for 'standard' group are taken from UVa problems with hints available in 'Methods to Solve' website. This is to ensure that there is a solution for each homework problem as the instructor (first author) has solved them before.

4.1.4. Tools Not Written by the Authors

In this subsection, we show few other tools that are not written by the authors to assist the running of CS3233 module.

A.   Mark Greve's UVa Toolkit

UVa toolkit (http://www.uvatoolkit.com/problemssolve.php) by Mark Greve from Aarhus University, Denmark is a good tool for finding problems of similar types. For example, typing "graph bipartite matching" in the given text box quickly shows UVa online judge (http://uva.onlinejudge.org/) problems that has these keywords (see Figure 15). This tool is also can be used to clarify input / output doubts as user can test his inputs, run Mark's accepted program to produce outputs, and compare his outputs w.r.t Mark's accepted program's outputs.
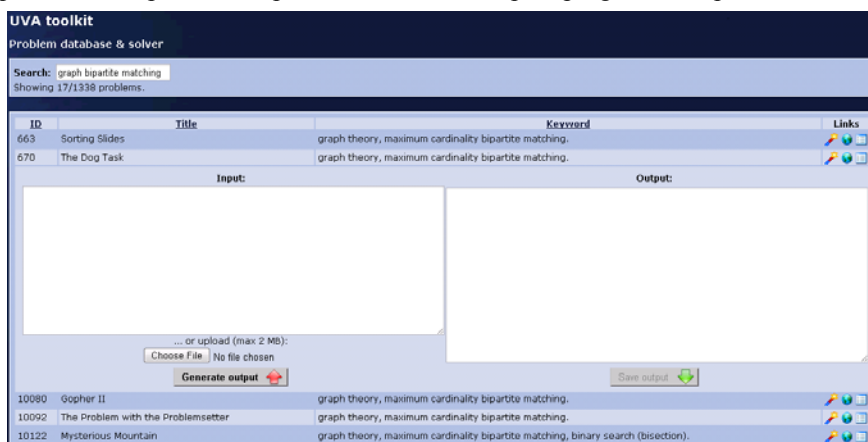

Figure 15. **Example** of using UVa Toolkit

This tool is not only useful for students who want to practice certain type of problem, but also for instructors in finding relevant examples problem for algorithm illustration or for homework assignments. UVa problem examples shown in Table 1 of this paper are mostly filled with the help of this tool.

B.   π Algorithmist Wiki Page

π Algorithmist (http://www.algorithmist.com) has similar features to 'Methods to Solve' and 'UVa Toolkit' and complements each other. The usage of wiki makes this website easier to expand.

C. José Paulo Leal's Mooshak

For arranging our local contests, we use Mooshak built by José Paulo Leal and team (http://mooshak.dcc.fc.up.pt/~zp/mooshak) from Universidade do Porto, Portugal. Mooshak allows us to have greater control over the contests as the data are hosted locally. Screen shots in Figure 1 and 2 shown earlier uses Mooshak.

D. TopCoder Algorithm Tutorial Page

TopCoder has a good algorithm tutorial page contributed by its top coders (http://www.topcoder.com/tc?d1=tutorials&d2=alg_index&module=Static)! This compilation can be treated as an online e-textbook about recent programming contest techniques. Students are encouraged to read these materials.

E. Igor Naverniouk's Code Archive

Igor Naverniouk (http://shygypsy.com/tools/) shares a compilation of his codes. Students can learn from his codes.

5. Conclusion

Universities have role to nurture future programmers for a world with increasing dependency on computers. NUS offers a special module titled 'CS3233 – Competitive Programming' to train talented students with the state-of-the-art data structures and algorithms during their undergraduate studies. Students do not just learn course materials but put this knowledge into practice by means of competitions, be it local ones within this module or international ones when they get selected to represent NUS / Singapore for ICPCs / IOIs.

Interested readers who want to set up similar module in their University can contact the authors directly to obtain some PowerPoint slides, sample codes, and sample contests data prepared by the authors.

Notable Achievements

The current version of Competitive Programming module (CS3233) in NUS has been running since mid 2008. The size of the programming community in NUS also grow from about <10 students 2 years ago to about ~15-20 students today.

In the past one and half year, with the contributions from all the coaches and trainers involved in this module, NUS ICPC teams enrolled in this module managed to obtain 6th place in Amritapuri Regional 2008, 15th & 16th place in Kuala Lumpur Regional 2008, 3rd place in Kanpur Regional 2008, Honorable Mention in ICPC World Final in Stockholm, Sweden, 2009, 7th & 10th place in Jakarta Regional 2009, and 3rd place in Manila Regional 2009. Singapore IOI team managed to obtain 2 silvers and 2 bronzes in 21st IOI in Plovdiv, Bulgaria, 2009.

Selected Students Feedbacks

Figure 15. CS3233 Students and Staffs in Semester 2, AY2008/2009

These are some selected feedbacks the students:

- The module is interesting and exciting. Many useful and uncommon algorithms and problems are taught in the module, and since the module is geared towards solving problems, the rate of application is higher (students can absorb more).
- Many topics are covered. Provide a good starting point for those who have not learned about competitive programming before and provide good exercise for those who involved in programming competition before.
- Learn a lot of interesting algorithm with interesting name. Test a lot on intelligence in understanding these algorithms.
- This module is very fun. We are introduced to many interesting algorithms.
- This module is too heavy but we did learn a lot in this module.
- It is hard to come up with a good syllabus for this module as the scope of ACM-ICPC is extremely wide and because the people who are taking the course have extremely different level of skills. I think the module did a good job in striking the balance. I do wish that more difficult things are taught, but there is a limit if we take the students into consideration.
- Good start for the ACM ICPC.
- I learned quite a lot of basic algorithm ideas.

Acknowledgements

References

[1] ACM International Collegiate Programming Contest (ICPC), http://cm2prod.baylor.edu
[2] IBM ACM ICPC Webpage, http://www.ibm.com/university/acmcontest
[3] ICPC Podcast, http://battleofthebrains.**podbean**.com
[4] International Olympiad in Informatics (IOI), http://ioinformatics.org
[5] Rujia Liu, Training ICPC Teams: A Technical Guide, CLIS, Banff
[6] Michal Forišek, IOI Syllabus, http://people.ksp.sk/~misof/ioi-syllabus/ioi-syllabus-2009.pdf
[7] Steven Halim, Roland H.C. Yap. 2007. Designing and Tuning SLS through Animation and Graphics: an Extended Walk-through. Engineering Stochastic Local Search Workshop, Brussels, Belgium, 2007: 16-30
[8] Steven Halim, Roland H.C. Yap, Hoong Chuin Lau. 2007. An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search. Principles and Practice of Constraint Programming, Providence, Rhode Island, USA, 2007: 332-347
[9] Steven Halim, Roland H.C. Yap, Felix Halim. 2008. Engineering Stochastic Strategies for the Low Autocorrelation Binary Sequence Problem. Principles and Practice of Constraint Programming, Sydney, Australia 2008: 640-645
[10] Miguel A. Revilla and Steven S. Skiena, *Programming Challenges*, Springer, 2003.
[11] Ahmed Shamsul Arefin, *Art of Programming Contest*, 2nd edition, ACM Solver, 2006.
[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliff Stein, *Introduction to Algorithms*, 2nd edition, MIT Press, 2001.
[13] Jan Kleinberg and Eva Tardos, *Algorithm Design*, Addison Wesley, 2005.
[14] Sanjoy Dasgupta, Christos Papadimitrou, Umesh Vazirani, *Algorithms*, McGraw Hill, 2008.
[15] Anany Levitin, *Introduction to The Design and Analysis of Algorithms*, Addison Wesley, 2003.
[16] Steven S. Skiena, *Algorithm Design Manual*, Springer, 1998.
[17] Robert Sedgewick, *Algorithms in C++ (Part 1-4 + Part 5)*, Addison Wesley, 1998.
[18] David Bentley, *Programming Pearls*, Addison Wesley, 2000.
[19] ACM ICPC-Live Archive, http://acm.uva.es/archive/nuevoportal
[20] UVa Online Judge, http://uva.onlinejudge.org
[21] Miguel A. Revilla, Shahriar Manzoor, Rujia Liu, Competitive Learning in Informatics: The UVa Online Judge Experience, Olympiad in Informatics, 2008, Vol 2, 131-148
[22] Felix Halim, Hunting UVa Problems++, http://felix-halim.net/uva/hunting.php
[23] Steven Halim, Methods to Solve, http://www.comp.nus.edu.sg/~stevenha/programming/acmoj.html
[24] Mark Greve, UVa Toolkit, http://www.uvatoolkit.com/problemssolve.php
[25] π Algorithmist, http://www.algorithmist.com
[26] Mooshak, http://mooshak.dcc.fc.up.pt/~zp/mooshak
[27] TopCoder, http://www.topcoder.com/tc?d1=tutorials&d2=alg_index&module=Static
[28] Igor Naverniouk's Code Archive, http://shygypsy.com/tools
[29] ACM ICPC in Wikipedia http://en.wikipedia.org/wiki/ACM_International_Collegiate_Programming_Contest
[30] QuestToSolve, http://www.questtosolve.com
[31] USACO Training Program Gateway, http://train.usaco.org/usacogate
[32] Wikipedia, http://en.wikipedia.org
[33] Google, http://www.google.com
[34] Tianjin University Online Judge, http://acm.tju.edu.cn/toj

About the Authors

Steven Halim is a PhD graduate from SoC, NUS. He is currently an instructor in the same University, teaching several programming related modules. He took part in Bina Nusantara (BiNus) high school contest, International Schools' Software Competition @ Singapore 1999, and 3 ACM ICPC regional contests in Singapore 2001, Aizu 2003 and Shanghai 2004. He maintains 'Methods to Solve' webpage that has been used by many programmers worldwide.

Felix Halim is the younger brother of Steven. He is currently pursuing PhD degree in SoC, NUS. His top achievements in programming contests are: Indonesian IOI team 2002, winner of ACM ICPC Regional Contest @ Kaohsiung 2006 with Bina Nusantara University (BiNus), Indonesia, and participated in ACM ICPC World Final @ Tokyo 2007. He maintains 'Hunting UVa problems' and other tools in his website. He actively joins Single Round Matches in TopCoder.