“Dictionary Wars”: An Inverted, Leaderboard-Driven Project for Learning Dictionary Data Structures

Kuba Karpierz
University of British Columbia
201-2366 Main Mall
Vancouver, BC, Canada
mr.karpierz@gmail.com

Joel Kitching
University of British Columbia
201-2366 Main Mall
Vancouver, BC, Canada
joel@jkweb.ca

Brendan Shillingford
University of British Columbia
201-2366 Main Mall
Vancouver, BC, Canada
br2609@interchange.ubc.ca

Elizabeth Patitsas
University of Toronto
40 St George St.
Toronto, ON, Canada
Ph: 416-978-3610
Fax: 416-978-4765
patitsas@cs.toronto.edu

Steven A. Wolfman
University of British Columbia
201-2366 Main Mall
Vancouver, BC, Canada
Ph: 604-822-0407
Fax: 604-822-5485
wolf@cs.ubc.ca

ABSTRACT

We present a highly reusable "inverted" project in which students learn asymptotic and practical behaviour of dictionary data structures—linked-lists, arrays, balanced trees, and hash tables—in an atmosphere of mild competition. Much like David Levine's Nifty Assignment "Sort Detective" [2], rather than implementing the dictionaries, students' programs generate input to our (unlabeled) implementations, and students use timing data to label the implementations. Much like Bryant and O'Halloran's computer architecture labs [1], students also compete to "convince" a web-based, automated system that their input generators distinguish the dictionaries based on trend-line behaviour. UBC has used the project in three terms, and we plan to use it at UBC and U Toronto in coming terms.

1. SIGNIFICANCE AND RELEVANCE

Dictionary data structures (also known as maps) are commonly used in a wide variety of computing applications as well as being an important topic in algorithms and data structures courses in the ACM CS2008 curriculum (e.g., under AL/FundamentalAlgorithms, PF/DataStructures, and various places for analysis of performance and tradeoffs). Modern languages generally support them either as built-in structures (as in Python) or as part of the core libraries (as in Java).

A long-running issue in computing education has been to find ways to help students learn to understand these data structures from above the implementation level [3]. Our project builds on the inverted assignment style of Levine [2] and Bryant and O'Hallaron [1] to provide a learning experience for students where they use these data structures from the outside, yet exercise deep and thorough understanding of the crucial idiosyncracies of each implementation's function.

Using a simple command syntax—one command per line where each command is composed of an 'I' for insert, 'F' for find, or 'R' for remove plus a numeric key[1]—students craft experiments to distinguish the dictionary implementations from each other. To do so student must understand best-, average-, and worst-case behaviours of the structures, the form of inputs that produce those behaviours, the performance curves these varied inputs produce, and various practical ramifications of the structures' function (e.g., the relatively large delays generated by rehashing vs. resizing of arrays).

To excel in the competition, students also make educated guesses augmented by further experiments into the structure of implementations. (For example, to force a hash table to generate a "clean" n^2-shaped curve for a series of n insertions, students might first reverse engineer the hash table's initial size and resizing strategy.)

Finally, this project forces our Computer Science students to coherently describe an experimental plan and use its results to justify their choices, a lamentably rare occurrence in CS projects!

2. POSTER CONTENT

We anticipate dividing the poster into 4 parts:

1. A brief description of the assignment itself.

   The core of this is: "Each student group gets an executable that (1) accepts command-line flags to choose among a set of 'mystery' dictionary implementations, (2) parses simple commands—one per line, each command composed of an 'I' (insert), 'F' (find), or 'R' (remove) plus a numeric key—and (3) feeds the operations into the dictionary implementation. The executable outputs performance (timing) data which students use to match the mystery to actual implementations and justify that matching. With high probability, the permutation between mystery and actual implementations is unique to each group."

2. Examples of (real) student strategies for solving the problem, illustrating the experiment design and reasoning students employ.

   We anticipate showing a handful of examples from students' submissions such as Figures 1 and 2, which show two steps of one student team's[2] reasoning. Note: the team preprocessed to show per-operation (really, per-block) rather than cumulative performance.

3. Description of the automated scoring system. We will briefly describe the technique used for curve-fitting; the technologies used to build the submission, autoscoring, and web interface components of the leaderboard; and screenshots and description of the leaderboard interface, e.g., the ranking screen, overall dictionary vs. dictionary comparison screen, individual student's overall summary screen, and individual student's report for a single input generator. Due to space restrictions, we show here only the leaderboard and detail view in Figure 3.

4. Closing comments. We will close with discussion of student participation rates over the terms in the optional bonus component of the project; known issues and future work (including, e.g., increased automation in grading and security issues with running student code); the URL for a sample leaderboard (`http://www.ugrad.cs.ubc.ca/~cs221/2011W2/fun/`) and for code download (in progress).

3. REFERENCES

[1] R. E. Bryant and D. R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Addison-Wesley, 2/e edition, 2010.

[2] N. Parlante, J. K. Estell, D. Reed, D. Levine, D. Garcia, and J. Zelenski. Nifty assignments. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, SIGCSE '02, pages 319–320, New York, NY, USA, 2002. ACM.

[3] B. Simon, M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders. Making sense of data structures exams. In *Proceedings of the Sixth international workshop on Computing education research*, ICER '10, pages 97–106, New York, NY, USA, 2010. ACM.

[1] Technically, this syntax is an interface to sets rather than dictionaries. Conflating these is common in data structures courses. Whether it's also problematic is outside the scope of this poster!

[2] The first author's team from when he was assigned the project as a student is Spring 2012.

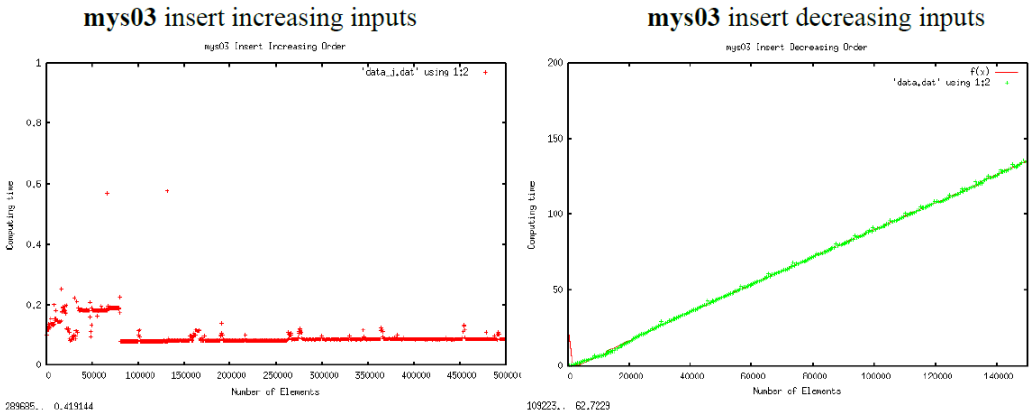


Figure 1: Two figures the students used to illustrate that the third mystery dictionary is not an unsorted linked list. From their report: "Since the list is not sorted, this implementation will give the same performance (for insertions) regardless of the order of inputs. This ruled out . . . mys03."

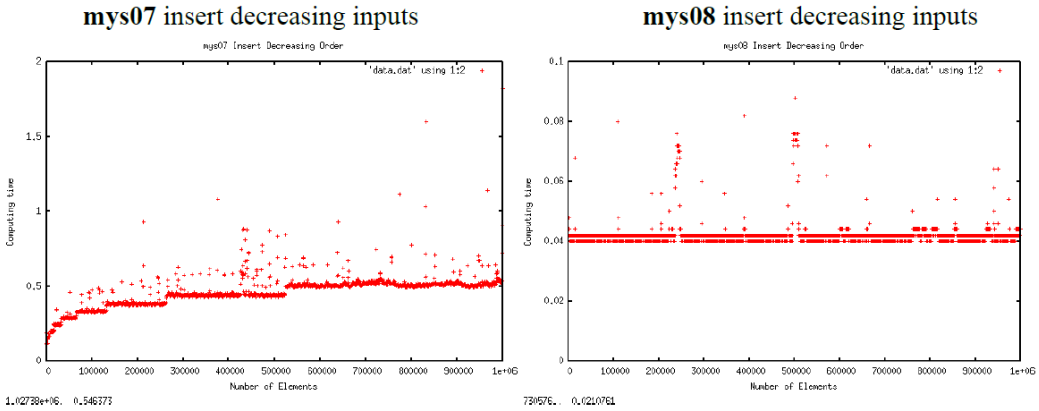
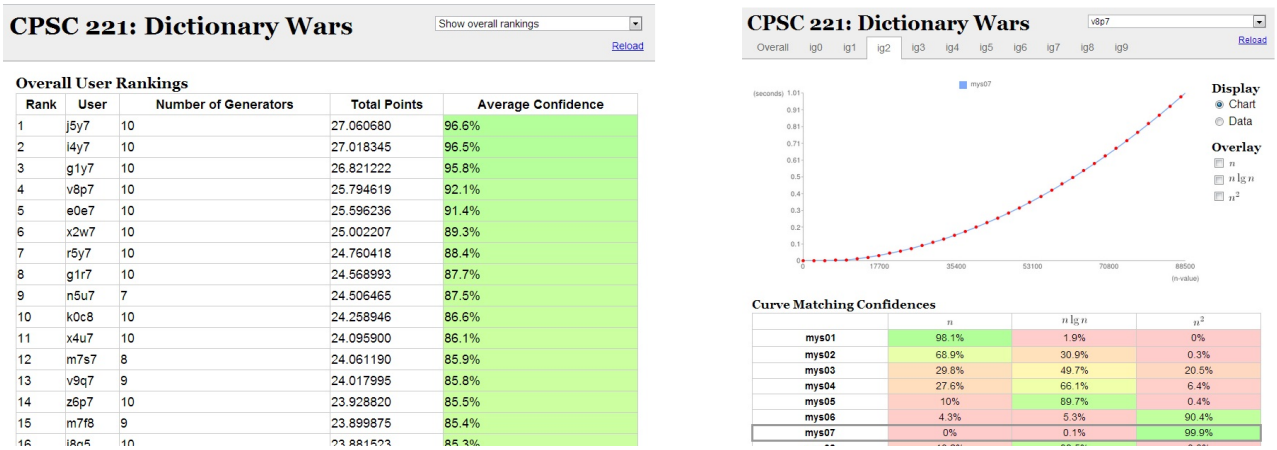


Figure 2: Two figures the students used to illustrate that the seventh and eigth mystery dictionaries are not the unsorted linked list. From their report: "Since there is no resizing needed for a linked list, there should be no jumps (or random, scattered dots) in the graphs. This ruled out . . . mys07, and mys08. . . "



CPSC 221: Dictionary Wars

Show overall rankings

Reload

Overall User Rankings

Rank	User	Number of Generators	Total Points	Average Confidence
1	j5y7	10	27.060680	96.6%
2	l4y7	10	27.018345	96.5%
3	g1y7	10	26.821222	95.8%
4	v8p7	10	25.794619	92.1%
5	e0e7	10	25.596236	91.4%
6	x2w7	10	25.002207	89.3%
7	r5y7	10	24.760418	88.4%
8	g1r7	10	24.568993	87.7%
9	n5u7	7	24.506465	87.5%
10	k0c8	10	24.258946	86.6%
11	x4u7	10	24.095900	86.1%
12	m7s7	8	24.061190	85.9%
13	v9q7	9	24.017995	85.8%
14	z6p7	10	23.928820	85.5%
15	m7f8	9	23.899875	85.4%
16	i8o5	10	23.881523	85.3%

	n	$n \lg n$	n^2
mys01	98.1%	1.9%	0%
mys02	68.9%	30.9%	0.3%
mys03	29.8%	49.7%	20.5%
mys04	27.6%	66.1%	6.4%
mys05	10%	89.7%	0.4%
mys06	4.3%	5.3%	90.4%
mys07	0%	0.1%	99.9%

Figure 3: A snapshot of the web-based leaderboard at the end of spring of 2012 (left). Clicking a row accesses a group's detailed results (right), giving access to cumulative time taken vs. number of operations performed for each input generator/dictionary pairing from that group. Another display uses an upper-triangular matrix to show the best results so far comparing each pair of dictionaries. Students track the leaderboard both to see how they're doing and to deduce and recreate other students' successful strategies.