A Penny in Every Fuse Box

By Terence Kelly

“Do classic textbook algorithms leave major room for improvement?” The question seems rhetorical, and the answer obvious: no. The timeless bedrock techniques that computer scientists learn as sophomores and memorize anew on the eve of programming job interviews have surely been honed to perfection by generations of critical scrutiny. Opportunities for improving the asymptotic efficiency of classic techniques while preserving their elegance were all surely spotted long ago and duly installed in curricula worldwide. Right? In fact, a recent paper reminds us that the situation is much more interesting than that, and it offers a teachable moment for students and educators alike.

For the most part, the knowledge taught under the heading of “data structures & algorithms” is quite mature, having in recent decades seldom changed in ways that greatly affect the undergraduate curriculum. Lookups in balanced binary search trees are still $O(\log N)$, and so on. A core topic, sorting, illustrates a trajectory culminating in diminishing returns: Between the 1990 first edition of the widely used Cormen et al. textbook [1] and the 2001 second edition [2], the state of the art in sorting algorithms advanced. Many of the improvements, however, were mainly of theoretical interest, because they are too complicated to be used in practice. A further decade on, the third edition (2009) reports little change to this situation [3]. And so it goes in many other “data structures & algorithms” topics: The material that *must* be taught to *all* undergraduates has not changed much lately.

For educators, of course, the stability that comes with maturity can be a good thing. Instead of constantly scrambling to update the curriculum to keep pace with a rapidly evolving state of the art, we can instead focus our efforts on ordinary diligence: hammering home eternal verities and implementing evidence-based instructional methods and assessment instruments. As with *Hamlet*, performances improve precisely because the script does not change.

Once in a long while, however, comes news that jolts us out of familiar routines. Such is the case with a recent paper [4] that presents a quite simple variation on classic textbook breadth-first search (BFS). This variation sometimes improves the asymptotic lower bound on running time, which surprised

Classic BFS [breadth-first search] terminates when the first-in/first-out (FIFO) queue at the heart of the algorithm empties. By contrast, “Efficient BFS” (E-BFS) terminates when the output reaches its final state, which may occur long before the queue drains.

several veteran CS educators, researchers, and programmers with whom I discussed the matter. Figure 1 shows classic textbook BFS in black and the improvement in **red**. Classic BFS terminates when the first-in/first-out (FIFO) queue at the heart of the algorithm empties. By contrast, “Efficient BFS” (E-BFS) *terminates when the output reaches its final state*, which may occur long before the queue drains. In terms of the explanatory coloring scheme used in some textbooks [3]—undiscovered vertices are white, enqueued vertices gray, and dequeued vertices black—E-BFS exploits the fact that per-vertex output fields are assigned at the white→gray transition: The *entire* output reaches its final state, and therefore the search can terminate, when the last white vertex turns gray.

The paper shows that whereas the asymptotic running time of classic BFS on a connected undirected graph is $\Theta(V+E)$, that of E-BFS is $\Omega(V)$. In words: Classic BFS always requires time proportional to the number of vertices plus the number of edges, but E-BFS sometimes requires time proportional to the number of vertices alone, which may be much smaller than $V+E$. Walk both algorithms through a complete graph—where $E=V \times (V-1)/2$ —to gain intuition for the difference. The paper’s experiments, which are easy to reproduce using the source code provided in [4], confirm that E-BFS is sometimes much faster on realistic sparse graphs and never much slower. Compared to E-BFS, classic BFS is like a penny in a fuse box—never better and sometimes catastrophically worse.

Breadth-First Search: inputs:	graph with vertex set V , source vertex s
data structures:	initially empty first-in/first-out queue FIFO, counter N
outputs:	distance and parent fields of vertices reachable from s
1 for each vertex u in V except s	# initialize non-source vertices
2 $u.parent \leftarrow \text{INVALID}$	# eventually contains parent vertex in BFS tree
3 $u.dist \leftarrow \infty$	# eventually contains distance from source
4 $s.parent \leftarrow \text{INVALID}$	# source has no parent
5 $s.dist \leftarrow 0$	# source is distance zero from itself
6 $N \leftarrow 1$	# number of vertices found (thus far, only s)
7 FIFO.INSERT(s)	# neighbors of vertices in queue may be undiscovered
8 while ! FIFO.EMPTY()	# classic (inefficient) termination test
9 $u \leftarrow \text{FIFO.REMOVE}()$	# we must check u 's neighbors for undiscovered vertices
10 for each vertex v adjacent to u	# walk down u 's adjacency list
11 if $v.dist == \infty$	# we're finding v for the first time
12 $v.parent \leftarrow u$	# v was found via u , so u is its parent
13 $v.dist \leftarrow u.dist + 1$	# v is one edge further from source than u
14 $N \leftarrow N + 1$	# another vertex has been found
15 if $N == V $	# efficient termination test: <i>every</i> vertex found...
16 terminate	# ...means all output finalized; nothing left to do
17 FIFO.INSERT(v)	# v 's neighbors might not have all been explored

Figure 1

If you're dismayed to realize that you've been teaching the penny rather than the fuse, take comfort: You've got plenty of company. My extensive informal survey of computer science educators, researchers, and working software engineers—including one SIAM Fellow, two ACM Fellows, one Ph.D. operations researcher, one Ph.D. mathematician, and easily a dozen proficient programmers, all deeply familiar with graph algorithms—found no one who had ever heard of E-BFS. An exhaustive search of my large personal library turned up many textbooks at both the undergraduate and graduate level that present classic BFS but none that mention E-BFS. The scholarly question of when and where E-BFS first appeared is beyond the scope of the present column; the important point is that large numbers of people who have needed it for decades didn't get the memo until very recently.

The extraordinary case of Efficient BFS holds several lessons, some obvious and others less so. Clearly, every course and textbook that covers graph search should show how to transform classic BFS into E-BFS (and the analogous transformation for depth-first search). My favorite pedagogical technique is to invite students to discover the enhancement for themselves after showing them classic BFS juxtaposed with my "efficiency haiku":

*to skip needless work
fix the termination test
stop when you are done*

A broader discussion of efficiency can follow. Many a data analysis platform has gone off the rails when a misguided quest for performance led to a tragicomic frenzy of unnecessary work [5]. To paraphrase Ben Franklin, those who would give up essential efficiency to purchase a little speed deserve neither. As

data centers account for an ever-increasing share of humanity's carbon footprint, efficiency becomes a moral issue. Good solutions begin with education.

The deeper lesson of the E-BFS affair is that we must train our students—and ourselves—to question authority. The classic algorithms in top-notch textbooks that have survived the scrutiny of generations of attentive readers probably admit little further refinement, but we should nonetheless approach them critically and check whether they leave money on the table. If nothing else, E-BFS should shake us out of the drowsy complacency that comes with familiarity and repetition. Finally, instructors or students who see opportunities for improvement must communicate the good news quickly and effectively to all who might benefit. If you see something, say something, lest pennies remain in our fuse boxes. ❖

References

1. Cormen, T.H., Leiserson, C.E., and Rivest, R.L. *Introduction to Algorithms*. (Cambridge, MA: MIT Press, 1990).
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. *Introduction to Algorithms*. 2nd ed. (Cambridge, MA: MIT Press, 2001).
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. *Introduction to Algorithms*. 3rd ed. (Cambridge, MA: MIT Press, 2009).
4. Kelly, T. Efficient graph search. *ACM Queue*, 18, 4 (July–August 2020); <https://queue.acm.org/detail.cfm?id=3424304>. Accessed 2020 Oct 28.
5. McSherry, F., Isard, M., and Murray, D.G. Scalability! But at what COST? in *Proceedings of the Fifteenth USENIX Workshop on Hot Topics in Operating Systems*. (Berkeley, CA: USENIX Association, 2015). <https://www.usenix.org/system/files/conference/hotos15/hotos15-paper-mcsherry.pdf>. Accessed 2020 Oct 28.

Terence Kelly

P.O. Box 61210, Palo Alto, CA 94306
 tpkelly@{acm.org, cs.princeton.edu, eeecs.umich.edu}

DOI: 10.1145/3433694

Copyright held by author/owner.