Open access • Proceedings Article • DOI:10.1145/364447.364561

Teaching internet algorithmics — **Source link** ⬈

Michael T. Goodrich, Roberto Tamassia

Institutions: Johns Hopkins University, Brown University

Published on: 01 Feb 2001 - Technical Symposium on Computer Science Education

Topics: The Internet and Algorithmics

Related papers:

- Teaching Abstract Thinking in Introduction to Computer Science for 7th Graders

- A game engine to learn computer science languages

- Computer science for non-technological cyber programs

- The novice programmer's "device to think with"

- Learning theory in Computer Science 1: an experiment supporting the use of multiple languages

Teaching Internet Algorithmics

Michael T. Goodrich*

Dept. of Comp. Sci.
Brown and Johns Hopkins Universities
goodrich@cs.jhu.edu

Roberto Tamassia†

Dept. of Comp. Sci.
Brown University
rt@cs.brown.edu

Abstract

We describe an Internet-based approach for teaching important concepts in a Junior-Senior level course on the design and analysis of data structures and algorithms (traditionally called CS7 or DS&A). The main idea of this educational paradigm is twofold. First, it provides fresh motivation for fundamental algorithms and data structures that are finding new applications in the context of the Internet. Second, it provides a source for introducing new algorithms and data structures that are derived from specific Internet applications. In this paper we suggest some key pedegogical and curriculum updates that can be made to the classic CS7/DS&A course to turn it into a course on Internet Algorithmics. We believe that such a course will stimulate new interest and excitement in a course that is perceived by some to be stale, boring, and purely theoretical. Interestingly, we argue that the foundational topics from CS7/DS&A should remain even when it is taught in an Internet-centric manner, which, of course, should come as no surprise to the seasoned computer scientist, who understands the value of algorithmic thinking.

1   Introduction

With the ever-increasing influence of the World Wide Web (WWW) and the Internet, the importance of data structure and algorithm engineering principles has significantly increased. Several new Internet companies, including Akamai (www.akamai.com), Yahoo (www.yahoo.com), Google (www.google.com), Altavista (www.altavista.com), and Intertrust (www.intertrust.com), have based major parts of their business models on new algorithmic ideas. These companies have all hired (sometimes as founders)

well-known algorithm engineering researchers to support this effort. Thus, algorithm engineering is having a major influence on how Internet applications are being implemented. Given the powerful influence that algorithm design is having on the Internet, we feel that it is appropriate for the way that algorithms are taught to be motivated and influenced by Internet applications. Such a pedagogical approach serves both to excite students about the subject matter of algorithm design as well as better prepare students for algorithm implementation at companies with Internet applications. In this paper, we describe such an Internet-based approach to teaching the traditional Junior-Senior algorithms course (CS7/DS&A).

Internet Algorithmics focuses on topics of algorithm and data structure design and engineering for combinatorial problems whose primary motivation comes from the Internet and Internet applications [7]. A course on Internet Algorithmics uses the paradigms of asymptotic analysis and algorithm engineering to the design and analysis of solutions to Internet-related problems, which find applications in the following domains:

- Internet infrastructure management
- Internet routing and multicasting
- Web caching
- Internet searching and information retrieval
- Information and network security
- Electronic commerce
- Web auctions
- Web-based application service providers

While these topics are clearly novel, they nevertheless involve applications of traditional algorithmic topics from the CS7/DS&A curriculum. Thus, with a slight reworking, the CS7/DS&A course can be cast in a new Internet light. While we have not performed a careful study to measure how well this approach improves student interest in the CS7/DS&A course, we have found experientially that such a motivation significantly increases student interest in algorithms topics. Indeed, in a recent introduction of this type of *Internet Algorithmics* course at Johns Hopkins University, enrollment limits had to be imposed on non-CS majors in an attempt to keep the class size at a reasonable level. Even then, enrollment ultimately exceeded the target by 44%. Based on this experience, in this paper we discuss some specific ways of teaching CS7/DS&A from an Internet perspective, so as to change

the perception that some students have that algorithms are boring and have no immediate benefits.

In the remainder of this paper we describe how many traditional algorithms topics can be cast in an Internet light, as well as show in some detail how graph and text processing algorithms specifically benefit from this approach.

2 Traditional Topics Taught in an Internet Light

In this section we briefly highlight how to cast in an Internet light some traditional topics from CS7/DS&A, excluding graph and text processing algorithms, which are specifically treated in subsequent sections.

- **Priority queues and dictionaries.** The key-based containers known as priority queues and dictionaries arise naturally in the context of Web auctions. The software administrating a Web auction must be able to map userid's to passwords and auctioning history, which is exactly the dictionary problem. Likewise, keeping track of the top bid (or top k bids) requires the use of a priority queue. Indeed, most web auctions allow a user to update his or her bid, which makes use of the increaseKey operation, which is optimized by such data structures as the binomial and Fibonacci heaps (e.g., see [2]).
- **Hashing.** Hashing is used extensively on the Internet. Perhaps the most natural place where it arises is in domain and URL caching. Every domain name server (DNS) maintains a collections of recently requested domains (such as www.yahoo.com or cs.caltech.edu) mapped to their IP addresses (such as 216.32.74.50 or 131.215.131.131). When a new domain name request arrives at a DNS, it first does a lookup in its cache to see if the domain can be immediately mapped to its IP address. This lookup is usually done using a hash table. Likewise, most web browsers keep a cache that maps URLs to files stored in a local directory on disk and the lookup process is typically done via hashing.
- **Sorting.** The need for fast sorting is naturally motivated from search engines. The best search engines on the Internet, including www.google.com, advertise that they return only the most pertinent reponses to a set of query words. Doing so online requires quickly sorting by some *rank* or *relevance* key the collection of all web pages that are determined to contain the query words. Incidentally, sorting is also used in the building of the data structure, called an *inverted file* (discussed below in Section 4), which is used to create this collection of Web pages.
- **Dynamic programming.** The algorithm engineering paradigm known as dynamic programming arises naturally in the context of Web crawling for information retrieval. In order to keep from revisiting what are essentially the same Web pages at different locations (from either plagarism or mirroring), a good Web crawler must be able to determine when a Web page is similar to one it has already explored. One of the most accurate ways of doing this is by solving the longest common subsequence (LCS) problem on the text of the two web pages. The LCS problem is in turn solved most efficiently by a simple dynamic program. This problem also arises in the Unix command diff and in the comparison of DNA sequences.
- **Divide-and-conquer.** Besides being the primary algorithm engineering design pattern used by the sorting algorithms merge-sort and quick-sort (e.g., see [6]), divide-and-conquer also arises in fast methods for performing the multiplication of large integers. Such computations arise on the Internet from information security contexts where large integers are used as cryptographic keys to digitally sign documemts and to keep sensitive personal and electronic commerce information private.
- **NP-completeness.** The concept that some problems are inherantly difficult to solve efficiently is no stranger to the Internet. For example, suppose we are given a set of web pages with known access statistics and we would like to balance perfectly the load of serving these pages between two servers. This problem is actually an instance of the PARTITION problem, which is known to be NP-complete (e.g., see [5]). Other such NP-complete problems are also easy to define from an Internet perspective.

Having briefly outlined how some traditional topics from CS7/DS&A can be cast in an Internet light, let us now study more in depth from an Internet perspective how we might teach graph and text processing algorithms, which are particularly relevant to the Internet.

3 Graph Algorithms

Graph algorithms topics can be clearly motivated from the networking infrastructure of the Internet itself, that is, the way that packets are routed around on the Internet. Good routing algorithms should route packets so as to arrive at their destinations quickly and reliably, while also being fair to other packets in the network (e.g., see [9]).

3.1 Flooding

In a flooding protocol, a message is sent to every other computer in a network as a broadcast. This computation can be performed as either a breadth-first or depth-first traversal; hence, graph traversal algorithms are immediately motivated by flooding communications.

The *flooding* algorithm for routing a message in a network is very simple and requires virtually no setup. A router x wishing to send a message M to a router y simply sends this message to all the routers that x is connected to. If y is one of these routers, then it receives the message. If a router $z \neq y$ receives a flooding message M from an adjacent router x, then z simply rebroadcasts M to all of the routers that z is connected to, except for x itself. Of course, left unmodified, this algorithm will cause an "infinite loop" of packet messages on any network that contains a cycle.

To avoid the infinite looping problem, we must associate some memory, or *state*, with the main actors in this algorithm. One possibility, which gives rise to a breadth-first type search of the network graph, is to store a list at each router x that keeps track of which flooding messages x has already processed. For example, one way to do this is to have each router assign a *sequence number* to each of the different flooding messages it originates. In this case, if the network is synchronous, or at least nearly-synchronous, then the packets exactly mimic the breadth-first search algorithm traditionally taught in the CS7 curriculum. (e.g., see [2, 6]).

3.2 The Distance Vector Algorithm

Another popular Internet routing algorithm is a distributed version of a classic algorithm for finding shortest paths in a graph due to Bellman and Ford [1, 4]. We model the problem as a graph problem, where nodes of the graph represent routers and edges represent connections between adjacent routers. We further assume that we have a weight assigned to each edge, which represents the cost of sending traffic down the corresponding connection. For the discussion here we will also suppose that the graph is undirected, indicating that sending a packet on a connection is equally costly in both directions.

The cost of a routing path is equal to the sum of the weights of the edges that make up that path. Thus, weights should be positive and should represent a reasonable metric for distance. For example, if the metric is "number of hops," then each edge has weight 1. Alternatively, if the metric is "average latency," then the cost of each edge could be the average queuing and sending latency for the connection represented by that edge, or it could be half of the average time for a collection of recent "ping" messages to go and return along this connection.

The main idea of the distance vector algorithm is for each router x to store a table, called its *distance vector*, D_x, which stores the length of the best known path from x to every other router in the network, and for each entry $D_x[y]$, the connection, $C_x[y]$, from x to use to begin the path to y. Initially, for each connection (x, y), we assign $D_x[y]$ and $D_y[x]$ equal to the weight of the edge (x, y), which we denote $W(x, y)$. All other D_x entries are set to "infinity." We then perform a series of rounds that iter-

atively refine each distance vector to find possibly better paths until every distance vector stores the true distance to all routers.

Each round of the distance vector algorithm consists of a collection of *relaxation* steps. At the beginning of a round, each router x sends its distance vector to all of its immediate neighbors in the network. After a router x has received the current distance vectors from each of its neighbors, it performs the following computations (the test and update represented by the if-statement is called a "relaxation"):

> **for** each router w connected to x **do**
>   **for** each router y in the network **do**
>   **if** $W(x, w) + D_w[y] < D_x[y]$ **then**
>   {We have just found a better route from x to y, through w.}
>   Set $D_x[y] \leftarrow W(x, w) + D_w[y]$.
>   Set $C_x[y] \leftarrow w$.
>   **end if**
>   **end for**
> **end for**

Thus, the time (and message complexity) for x to complete a round is $O(d_x n)$, where d_x is the degree of x and n is the number of routers in the network. We iterate this algorithm for δ rounds, where δ is the diameter of the network (if we don't know the diameter of the network, then we must choose $\delta = n - 1$).

The correctness of the distance vector algorithm follows by a simple inductive argument: at the end of round i, each distance vector stores the shortest path to every other router restricted to visit at most i other routers along the way. This fact is true at the beginning of the algorithm, and the relaxations done in each round ensure that this will be true after the round as well. Thus, after we have performed a number of rounds equal to the diameter of the network, each distance vector stores the true distance to each router in the network. Therefore, once we have performed this setup, the routing algorithm is quite simple: if a router x receives a message intended for router y, x sends y along the connection $C_x[y]$.

3.3 The Link-State Algorithm

Another imporant routing algorithm is a distributed version of a classic shortest path algorithm due to Dijkstra [3]. As in the distance vector algorithm description, we model the network as a weighted graph. Whereas the distance vector algorithm performed its setup in a series of rounds that each required only local communication between adjacent routers, the link-state algorithm computes in a single communication round that requires lots of global communication throughout the entire network.

The *link-state algorithm* proceeds as follows: each router x determines the weight of all its adjacent connections (either by experimental tests using "ping" messages

or *a priori* knowledge). It then packs all of these weights into a single message, W_x, and it broadcasts this message to all other routers in the network using a flooding routing protocol (which requires no prior setup, see Section 3.1). The message W_x is a representation of the state of each link adjacent to x. After a router x has recieved the link-state message, M_y, from every other router in the network, it has recieved a complete description of all the connections in the entire network, including the weight of every connection. Given this complete knowledge, the router x can then perform Dijkstra's shortest path algorithm on its internal representation of the network, using x as the starting vertex, to determine the shortest path from x to every other node in the network. This internal computation takes $O(m \log n)$ time using standard implementations of Dijstra's algorithm, or $O(n \log n + m)$ using more sophisticated data structures (e.g., see [2]). Once this algorithm completes, x stores the connection, $C_x[y]$, for each router y, to use for routing any message to y.

4 Text Processing Algorithms

Document processing is rapidly becoming one of the dominant functions of computers on the Internet. Computers are used to edit documents, to search documents, to transport documents over the Internet, and to display documents on printers and computer screens. Web "surfing" and Web searching are becoming significant and important computer applications, and many of the key computations in all of this document processing involve character strings and string pattern matching. For example, the Internet document formats HTML and XML are primarily text formats, with added tags for multimedia content. Making sense of the many terabytes of information on the Internet requires a considerable amount of text processing.

In teaching text processing algorithms, therefore, we believe it is important to motivate the discussions from an Internet perspective, focusing on the study of the fundamental text processing algorithms for quickly performing important string operations. Of particular importance are algorithms for string searching and pattern matching, since these can often be computational bottlenecks in many document-processing applications. We also believe it possible to introduce some new Internet-based data structure and algorithmic issues involved in text processing, as well.

The progression of text processing topics studied in an Internet-based algorithms course can follow a simple and natural schedule. It begins with terminology and notation for the string ADT, followed by methods for string pattern matching. In addition, it is useful to include a study of the trie data structure, which is a tree-based structure that allows for fast searching in a collection of strings. In addition, some instructors may wish to include the important text processing problem of compressing a document of text so that it fits more efficiently in storage or can be transmitted more efficiently over a network. Finally, we recommend the study of another text processing problem, called the longest common subsequence problem (as discussed above in Section 2), that deals with how we can measure the similarity between two documents. All of these problems are topics that arise often in Internet computations, such as Web crawlers, search engines, document distribution, and information retrieval.

In addition to having interesting applications, the topics of Internet-based text processing highlight some important algorithmic design patterns. In particular, string pattern matching highlights the *brute-force method*, which is often inefficient but has wide applicability. In text compression we can introduce the *greedy method*, which often allows us to approximate solutions to hard problems, and for some problems (such as in text compression) actually gives rise to optimal algorithms. Finally, as mentioned earlier, in discussing text similarity, we can introduce the *dynamic programming* design pattern, which can be applied in some special instances to solve a problem in polynomial time that appears at first to require exponential time to solve.

In addition to the traditional text processing topics, taught in a new light, as highlighted above, an Internet-based algorithms course can introduce new topics, such as we mention briefly in the subsection below.

4.1 Search Engines and Inverted Files

The World Wide Web contains a huge collection of text documents (Web pages). Information about these pages are gathered by a program called a *Web crawler*, which then stores this information in a special dictionary database. A Web *search engine* allows users to retrieve relevant information from this database, thereby identifying relevant pages on the Web containing given keywords.

The core information stored by a search engine is a dictionary, called an *inverted index* or *inverted file*, storing key-value pairs (w, L), where w is a word and L is a collection of pages containing word w. The keys (words) in this dictionary are called *index terms* and should be a set of vocabulary entries and proper nouns as large as possible. The elements in this dictionary are called *occurrence lists* and should cover as many Web pages as possible.

We can efficiently implement an inverted index with a data structure consisting of:

1. An array storing the occurrence lists of the terms (in no particular order).
2. A compressed trie for the set of index terms, where each external node stores the index of the occurrence list of the associated term.

The reason for storing the occurrence lists outside the trie is to keep the size of the trie data structure sufficiently small to fit in internal memory. Instead, because of their large total size, the occurrence lists have to be stored on disk. We note in passing that the construction of an inverted file provides another Internet-based application of sorting.

With the inverted file data structure, a query for a single keyword is similar to a word matching query in a trie. Namely, we find the keyword in the trie and we return the associated occurrence list. When multiple keywords are given and the desired output are the pages containing *all* the given keywords, we retrieve the occurrence list of each keyword using the trie and return their intersection. To facilitate the intersection computation, each occurrence list should be implemented with a sequence sorted by address or with a dictionary, which could movitave, for example, the generic merge computation used in the well-known merge-sort algorithm.

In addition to the basic task of returning a list of pages containing given keywords, search engines provide an important additional service by *ranking* the pages returned by relevance. Devising fast and accurate ranking algorithms for search engines is a major challenge for computer researchers and electronic commerce companies.

5   Conclusion

We have presented an approach to the teaching of a Junior-Senior course on data structures and algorithms, which is often referred to as CS7/DS&A. We have shown how the traditional topics from this course can remain in an Internet-centered teaching of it, but that these topics can receive fresh motivation from Internet applications. In addition, we have shown how some new topics, such as network routing algorithms and inverted files (which are actually not new topics at all; e.g., see [8, 9]), can be added to the curriculum of the CS7/DS&A course without much displacement of other more-traditional topics, such as NP-completeness. Indeed, this Internet-centered approach is reflected in a recent book by the authors on *Algorithm Engineering* [7].

References

[1] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[6] M. T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java*. John Wiley and Sons, New York, 1998.

[7] M. T. Goodrich and R. Tamassia. *Algorithm Engineering*. John Wiley and Sons, New York, 2001.

[8] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[9] M. Steenstrup. *Routing in Communications Networks*. Prentice Hall, Englewood Cliffs, 1995.