Advanced data structures in practice [closed]

Asked 16 years ago Modified 6 years, 7 months ago Viewed 11k times

45

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed last year.

Improve this question

In the 10 years I've been programming, I can count the number of data structures I've used on one hand: arrays, linked lists (I'm lumping stacks and queues in with this), and dictionaries. This isn't really surprising given that nearly all of the applications I've written fall into the forms-over-data / CRUD category.

I've never needed to use a red-black tree, skip list, double-ended queue, circularly linked list, priority queue, heaps, graphs, or any of the dozens of exotic data structures that have been researched in the past 50 years. I feel like I'm missing out.

This is an open-ended question, but where are these "exotic" data structures used in practice? Does anyone have any real-world experience using these data structures to solve a particular problem?

data-structures

Share

Improve this question

Follow

asked Dec 23, 2008 at 15:49

Juliet

81.4k ● 46 ● 199 ● 229

14 Answers

Sorted by: Highest score (default)

32

Some examples. They're vague because they were work for employers:

- A heap to get the top N results in a Google-style search. (Starting from candidates in an index, go through them all linearly, sifting them through a min-heap of max size N.) This was for an image-search prototype.
- Bloom filters cut the size of certain data about what millions of users had seen down to an amount that'd fit in existing servers (it all had to be in RAM for speed); the original design would have needed many new servers just for that database.
- A triangular array representation halved the size of a dense symmetrical array for a recommendation

engine (RAM again for the same reason).

- Users had to be grouped according to certain associations; union-find made this easy, quick, and exact instead of slow, hacky, and approximate.
- An app for choosing retail sites according to drive time for people in the neighborhood used Dijkstra shortest-path with priority queues. Other GIS work took advantage of quadtrees and Morton indexes.

Knowing what's out there in data-structures-land comes in handy -- "weeks in the lab can save you hours in the library". The bloom-filter case was only worthwhile because of the scale: if the problem had come up at a startup instead of Yahoo, I'd have used a plain old hashtable. The other examples I think are reasonable anywhere (though nowadays you're less likely to code them yourself).

Share Improve this answer Follow

edited Oct 28, 2009 at 7:45

answered Dec 23, 2008 at 19:24

Darius Bacon

15.1k ● 6 ● 56 ● 53

13

B-trees are in databases.

R-trees are for geographic searches (e.g. if I have 10000 shapes each with a bounding box scattered around a 2-D



plane, which of these shapes intersect an arbitrary bounding box B?)

deques of the form in the C++ STL are growable vectors (more memory-efficient than linked lists, and constant-time to "peek" arbitrary elements in the middle). As far as I can remember, I've never used the deque to its full extent (insert/delete from both ends) but it's general enough that you can use it as a stack (insert/delete from one end) or queue (insert to one end, delete from the other) and also have high-performance access to view arbitrary elements in the middle.

I've just finished reading Java Generics and Collections -- the "generics" part hurts my head, but the collections part was useful & they point out some of the differences between skip lists and trees (both can implement maps/sets): skip lists give you built-in constant time iteration from one element to the next (trees are O(log n)) and are much simpler for implementing lock-free algorithms in multithreaded situations.

Priority queues are used for scheduling among other things (here's a webpage that briefly discusses application); heaps are usually used to implement them. I've also found that the heapsort (for me at least) is the easiest of the O(n log n) sorts to understand and implement.

Share Improve this answer Follow

edited Dec 23, 2008 at 16:37

answered Dec 23, 2008 at 15:53

Jason S

189k ● 171 ● 630 ● 995

9

They are often used behind the scenes in libraries. For example an ordered dictionary data structure (i.e. an associative array that alows sorted traversal by keys) is as likely as not to be implemented using a red-black tree.

Many data structures (splay trees come to mind) are interesting for their optimal behaviour in certain circumstances (temporal locality of reference in the case of splay trees), so they are mainly relevant for use in these cases. In most circumstances the real benefit of a working knowledge of these data structures is to be able to employ them in the right circumstances with a reasonable understanding of their behaviour.

Take sorting, for example:

- In most circumstances quicksort or a modified quicksort that drops to another method when the individual segments get small enough is typically the fastest sorting algorithm for most purposes. However, quicksort tends to show suboptimal behaviour on nearly-sorted data.
- the main advantage of a heap sort is that it can be done in situ with minimal intermediate storage, which makes it quite good for use in memory constrained systems. While it is slower on average (although still

n log(n)), it does not suffer from the poor worst case performance of quicksort.

- A third example is a merge sort, which can be done sequentially, making it the best choice for sorting data sets much larger than your main memory. Another name for this is the 'external sort', meaning you can sort using external storage (disk or tape) for intermediate results.

Share Improve this answer

Follow

edited May 20, 2011 at 8:51

answered Dec 23, 2008 at 15:53

ConcernedOfTunbridge Wells

66.5k ● 15 ● 148 ● 198

5

It depends on the level of abstraction that you work at.

I know I have similar experience as you do. At the current level of abstraction of most software development. Dictionary and List are the main data structures we use.

I think if you look down at lower level code you will see more of the "exotic" data structures.

Share Improve this answer

Follow

answered Dec 23, 2008 at 15:53

John Sonmez

7,196 ● 5 ● 39 ● 56

I agree. Given how high up my code is in the software stack, if there's a data structure I need and it's not present in an existing library beneath my code, then that's usually a shortcoming of the libraries. – reuben Dec 25, 2008 at 6:03

3

I use ring buffers/circular queues constantly in embedded work to service interrupts (e.g. serial ports).

Tree structures are heavily used in computer graphics.

If you use STL map or set data structures, then you are probably using a red-black tree without even knowing it!

Share Improve this answer

Follow

answered Dec 23, 2008 at 15:55

Judge Maygarden

27.5k • 9 • 83 • 100

3

I think you see fancy data structures used most some higher level algorithms. The main example that comes to mind for me is A* which uses a Graph and a Priority Queue implemented by a Heap.

Share Improve this answer

Follow

answered Dec 23, 2008 at 18:33

Jon Walsh

In finance you need to use a tree to calculate the value of an instrument that depends on many other dynamic

3

values. Spreadsheets have a similar tree of dependencies, and compilers create an abstract syntax tree before translating to machine code.

Share Improve this answer

Follow

answered Feb 18, 2009 at 3:36

RossFabricant

12.5k ● 3 ● 43 ● 50

2

[Fibonacci heaps](#) are used for efficient implementations of [Dijkstra's algorithm](#).

Share Improve this answer

Follow

answered Apr 21, 2010 at 22:44

kolistivra

4,419 ● 10 ● 49 ● 60

Is that actually true? ISTR that Fibonacci heaps are only fast in theory and not in practice. – J D Jan 4, 2013 at 21:25

You might wanna look at this: stackoverflow.com/questions/504823/… – kolistivra Jan 4, 2013 at 22:10

1

Yes, sometimes. The problem I see is that a number of people although they know them, they don't know how to really apply them. Most people revert back to arrays linked lists etc. They'll get the job done in most cases as a more advanced data structure (sometimes you really have to "kick it" into place), they are just less efficient.

People tend to do what is easier for them, but it isn't necessarily the best way of doing something. I can't fault them, I am sure I do it too, but that is why you don't see a lot of the "advanced" concepts in programming.

Share Improve this answer

Follow

answered Dec 23, 2008 at 15:54

kemiller2002

115k ● 28 ● 199 ● 253

1

I often use sets, sorted collections (always keep their elements in sorted order, and support fast element insertion) and lazy lists.

Share Improve this answer

Follow

answered Dec 23, 2008 at 16:38

Jules

6,346 ● 2 ● 32 ● 43

1

Balanced trees (Red-black etc) are typically used in the implementation of an abstract data type.

There are only a relatively small number of abstract data types, such as

- list
- map
- ordered map
- multi map

- ordered multi map
- priority queue (which looks a lot like an ordered multi map)

Likewise, a set looks a lot like a map, but you don't need the values, only keys.

I have found most of these useful from time to time; a priority queue is a very useful data structure and has applications in all sorts of algorithms (For example scheduling, path-finding etc).

You said "Dictionary", you probably meant either a map or an ordered map.

Some maps are unordered (typically implemented as a hash) - this is a useful subset of an ordered map.

Share Improve this answer

Follow

answered Oct 28, 2009 at 7:51

MarkR

63.5k ● 15 ● 119 ● 154

1

I have used a **circular list** for caching.

A C++ class template provides interface for getting objects (`Cache<Obj, Len>`). Several instatinations of it return different types of 'screens' as in different views of a graphical interface. Behind the scenes, if the requested 'screen' is not available, it gets created (expensive operation) and pushed to the head of the ring buffer, pushing the oldest one out (unloading its textures etc).

Thus a compromise is achieved between always reading a bunch of image files from hard disk, and just loading all the images in RAM and keeping them forever. The compromise is controlled by the length of the various buffers.

Share Improve this answer

Follow

edited May 6, 2018 at 10:13

answered May 6, 2018 at 10:06

Vorac

9,034 12 63 107

0

I've used circular linked lists to implement queues (in C) that I'm going to iterate over forever, i.e. a network connection queue.

But I find that when I use higher-level languages, I don't find myself bothering to implement queues in this manner, because I can dynamically grow and shrink a list without worrying too much about it. Of course, there's a performance price for this, because I have less control over when memory allocation is occurring, but that's one of the prices we pay for being able to have very flexible lists.

Share Improve this answer

Follow

answered Dec 23, 2008 at 16:04

Daniel Papasian

16.4k 6 31 32

0

You will tend to see more complicated data structures when it is dictated by the needs of the code. Usually I'll see this when you're dealing with more complex code at lower levels, i.e. in the core operating system, writing fundamental parts of a class library (implementing string, array, etc), write extermely performant or multi-threaded code, etc. The other place I think they play an important role is in implementing specific algorithms, searching, sampling, statistical analysis, optimization, etc algorithms are often written with particular data structures in mind.

Share Improve this answer

Follow

answered Dec 23, 2008 at 16:16

Peter Oehlert
16.9k ● 7 ● 46 ● 49