<antirez>

In defense of linked lists

antirez 3 hours ago. 18814 views.

A few days ago, on Twitter (oh, dear Twitter: whatever happens I'll be there as long as possible - if you care about people that put a lot of energy in creating it, think twice before leaving the platform). So, on Twitter, I was talking about a very bad implementation of linked lists written in Rust. From the tone of certain replies, I got the feeling that many people think linked lists are like a joke. A trivial data structure that is only good for coding interviews, otherwise totally useless. In a word: the bubble sort of data structures. I disagree, so I thought of writing this blog post full of all the things I lov about linked lists.

So, get ready to read a sentimental post about a data structure, and don't tell I didn't wa you.

Linked lists are educational. When your teacher, or the page of a book, or anything that exposes you for the first time to linked lists shows you this little circle with an arrow pointing to another circle, something immense happens in your mind. Similar to what happens when you understand recursion for the first time. You get what data structures made of link truly are: the triviality of a single node that becomes a lot more powerful and complex onc it references another one. Linked lists show the new programmer fundamental things about space and time in computation: how it is possible to add elements in a constant time, and h order is fundamentally costly, because if you want to insert an element "in place" you have to go from one node to the other. You immediately start thinking of ways to speed up the process (preparing you for the next things), and at the same time you understand, deeply, what O(1) and O(N) really mean.

Linked lists are augmentable. Add a pointer to the previous element, and now it is possible to go both sides. Add "far" pointers from time to time, and you have a skip list with completely different properties. Change every node to hold multiple items and your linked list becomes unrolled, providing very different cache obviousness properties. Linked lists can be embedded, too. The Linux kernel, for instance, has macros to add a field to any structures in order to link them together. There is more: linked lists are composable. This is a bold property: you can split a linked list into two in O(1), and you can glue two link lists in O(1) as well. If you make judicious use of this property, interesting things are possible. For instance, in Redis modules implementing threaded operations, the thread processing the slow request dealt with a fake client structure (this way there was no locking, no contention). When the threaded command finally ended its execution, the output buffer of the client could be glued together to the actual buffer of the real client. This was easy because the output buffer was represented with a linked list.

Linked lists are useful: Redis can be wrong, but both Redis and the Linux kernel can't. The

Linked lists are useful. Redis can be wrong, but both Redis and the Linux kernel can't. They are useful because they resemble certain natural processes: adding things in the order they arrive, or in the reverse order, is natural even in the physical world. Pulling items incrementally is useful too, as it is moving such items from head to tail, or moving them a position after the current one.

Linked lists are simple. It is one of those rare data structures, together with binary tree and hash tables and a few more, that you can implement just from memory without likely stepping into big errors.

Linked lists are conceptual. A node pointing to itself is the most self centered thing I ca imagine in computing: an ideal representation of the more vulgar infinite loop. A node pointing to NULL is a metaphor of loneliness. A linked list with tail and head connected, a powerful symbol of a closed cycle.

For all those reasons, I love linked lists, and I hope that you will, at least, start smili at them.

: