Linked Lists (6/8/2021)

Linked lists involve properties and trade-offs worth considering. In a circular representation a linked list can be searched from any position. A doubly linked list may not be slower due to instruction-level parallelism and is amenable to parallel search in a circular representation (e.g. threads in opposite directions).

A doubly linked list provides a guarantee that a node keeps its address in memory throughout its lifetime. Such a guarantee may be handy if there is a hash table mapping to node addresses for in-list access. In contrast, deleting a node given its pointer (e.g. after a search) in a singly linked list, almost certainly involves copying a node, thus requiring two updates of a hash table in contrast to a doubly linked list.

Induction for Proving Existence of an Invariant during Multithreaded Execution (12/11/2020)

The first step towards proving the correctness or a property of a multithreaded algorithm can be in identifying an invariant that exists at a defined time interval during algorithm execution. Finding a representation that is amenable to an inductive argument can be instrumental.

To prove an approximation bound for the optimality of a greedy scheduler, the choice of an ordering of threads in an execution DAG informed an inductive argument for the existence of a set of threads with a desirable property at any time before the last step of the scheduler.

https://github.com/alfin3/theory-toolbox/blob/master/6_046_notes/22_dyn_multithreading_greedy_scheduler/proof-greedy-scheduler-approx-bound.pdf

Amortized Cost of Access for Algorithms on Distributed Infrastructure (1/25/2020)

Algorithm analysis and therefore design tools in the emerging area of distributed infrastructure remain limited. While algorithm parallelism is a meaningful objective in a multi-node/pod infrastructure setting, latency differences associated with a computation present an additional consideration. A better algorithm uses more bits of a transferred block of bits of a given size (spatial locality) and reuses the transferred bits to a greater extent (temporal locality). A better locality mitigates a higher latency. Amortized cost of access presents a worthwhile objective.

Representation Choice as Blueprint for Amortized Analysis (7/5/2019)

Representation can be central to enabling amortized analysis, providing a chargeable gap between "normal" and "bad" cases.

For example, choosing a representation that i) generally embedded ops of pre-augmented structure, ii) provided the necessary augmentation properties, and iii) provided a chargeable gap between cases for amortized analysis enabled the partially and fully persistent data structure constructions. The guarantee for changing the present and reading the past (partial persistence), and changing and reading the present and the past (full persistence) at a constant amortized overhead in time and cost in space per change, sounds almost miraculous. This general and lasting result is enabled by the choice of representation: https://dl.acm.org/citation.cfm?id=64317

James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. 1989. Making data structures persistent. *J. Comput. Syst. Sci.* 38, 1 (February 1989), 86-124. DOI=http://dx.doi.org/10.1016/0022-0000(89)90034-2