A random-access pool of similar objects

NEIL A. EDELMAN

2022-03-01

Abstract

A pool is parameterized to one type, offering packed random-access insertion and deletion. Pointers to valid items in the pool are stable, but not generally in any order. When removal is ongoing and uniformly sampled while reaching a steady-state size, it will eventually settle in one contiguous region.

1 MOTIVATION

In many applications, we would like a stream of many similar object's addresses to be stable throughout each of their lifetimes, but we can not tell, *a priori* how many objects at one time will be needed. Dynamic arrays are not suited for this because, in order there to be a contiguity guarantee, the pointers are not guaranteed to be stable. C^{++}'s `std::deqeue` is close, but it only allows deletion from the ends.

The pool, therefore, must not be contiguous, but we want blocks of data to be in one section of memory for fast cached-access and low storage-overhead. This suggests an an array of chunks, each one exponentially bigger than the last full chunk.

To reach the ideal contiguous chunk of memory, we only allocate memory to a chunk from the primary, biggest, chunk_0. When data is deleted from a secondary chunk, it is unused until all the data is gone and we can free that chunk.

2 DETAILS

We face a similar problem as *OCaml*'s garbage collection: in chunk_0, we need some way to tell which are deleted. The first choice is a free-list. When adding an entry, check if the list is not empty, and if it is not, we recycle deleted entries by shifting the list. Alternately, popping the free-list works, too, but on average, shifting yields lower, more compact addresses.

The free-list is $\mathcal{O}(1)$ run-time, but hard $\Theta(\sum_n \text{capacity of chunk}_n)$[1] space requirement, (all analysis amortized to the number of entries in chunk_0.) Alternately, we could use a free-heap, who's run-time is worse, $\mathcal{O}(\log \text{chunk}_0)$, but space requirement is much more reasonable, $\mathcal{O}(\text{chunk}_0)$, and a reference count on the secondary chunks.

Sort by memory, $\mathcal{O}(\log \text{chunks})$.

Approximation to golden ratio for array.

$if(r > c.size/(growth = 50.0))$

Figure 1 shows a diagram of a pool, the same type that was tested in Figure **??**.

3 PERFORMANCE

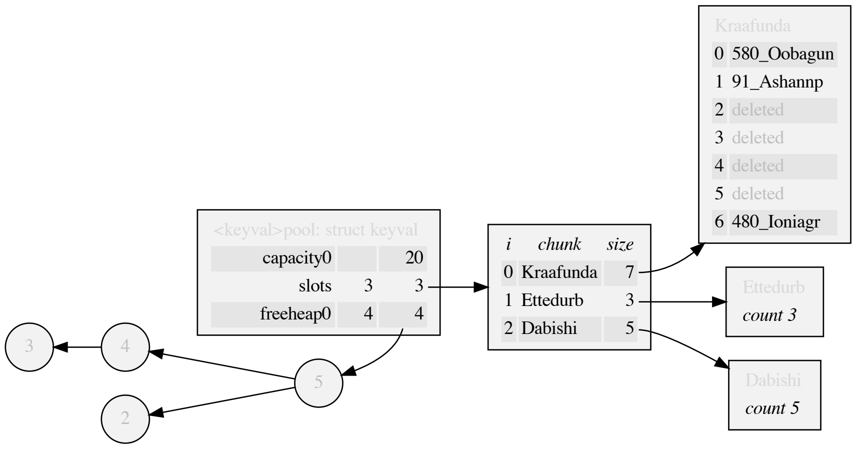

Figure 1: A pool consists of several blocks of packed data and a free-heap for block-0.

Figure 2: A comparison of chained techniques in a C^{++} benchmark with a log-x scale.

4 IMPLEMENTATION

5 CONCLUSION

REFERENCES

[1] D. E. Knuth, "Big omicron and big omega and big theta," *SIGACT News*, vol. 8, p. 18–24, apr 1976.