HP35 - A Stack Structure

Fredrik Berzins

Fall 2023

Introduction

The task has been to make a time and memory efficient stack structure? To test different stack structures I made both a static, dynamic and no array dynamic.

Stack explanation

A Stack structure works as a pile of documents wear the most recent item is on top. When getting the top element of the stack you "Pop" it, witch both returns it to you and removes it from the stack. When putting en element on the Stack you "Push" it to the top of the stack.

Static or Dynamic

A static stack structure is very simple since the stack never changes size that means it has some limitations, since it can't expand it can get overfilled and will also always take up a fixed amount of space that might be unnecessarily large.

Wile a dynamic structure can be smaller and expand when needed and shrink when not the down side is that changing size take time and it has a risks wasting time shrinking and expanding unnecessarily.

A no array dynamic structure is very time consuming but has the benefit of never wasting memory since it expands and shrinks as the stack changes size.

Stack implementation

Static

The static structure is an array with a int to keep track of the index of the top value in the stack.

Dynamic

The dynamic structure is built as the static but it has a extra method to resize. It resizes when the top reaches the limit or the array, at the point it doubles the array size. If it is shrinking it only dose that when the top value is one third of the array length, at that point it halves the previous array length.

No array Dynamic

I made the Dynamic structure without an array fundamentally different as it stores Stack objects which is a object with a data type and reference to the element above and bellow it. It works like a linked node structure but with references in both directions. This makes it the exact length needed and can't be overfilled.

Benchmarks

When bench marking, 1024 values wore pushed then pooped from the stack, this was done 100 times, then repeated seven times and the results shown in the tables below wore from the last five loops of a 100 tests. This was then done again to get five more results for a total of ten per stack structure. This gives more consistent values since it negates the time delay to move stuff to the cache. For consistency `System.gc()` was ran in between all 100 tests.

Result

The result from the benchmark was that static median was 2µs, the dynamic structures has a median of 23µs and the non array dynamic was even slower at a median of 51µs. This gives that the dynamic structure is about ten times slower compared to the static and the non array dynamic is even slower at about 2 times compared to the dynamic. The major time difference between static and dynamic array based stack structures comes from the copying of the array when resizing and having to check if it needs resizing.

One thing that stands out from the results is the difference in max time from each test where the difference isn't as drastic compare to the difference between medians. The median of the max value from each stack structure was 42µs for static, 212µs for dynamic and 194µs for the non array dynamic, this shows that dynamic is about five times slower and that the non array dynamic stack is slightly faster then the dynamic.

Static

A static structure the time for each pop and push will be very stable and not change drastically. It will also have the fastest times as seen below in tables 1, 2 and 3.

Try	**Min**	**Median**	Mean	**Max**
1	2.2	3.1	5.3	38.5
2	2.2	3.3	4.5	50.3
3	2.1	3.1	5.0	39.4
7	2.2	3.1	3.9	35.3
4	2.1	3.1	5.3	44.5
5	2.1	3.0	4.7	56.1
6	2.0	3.0	3.8	23.6
8	2.0	3.0	4.5	50.4
9	2.1	2.9	4.2	32.2
10	2.2	3.3	4.7	50.8

Table 1: Min, median, mean and max time for a static structure, all times are in microsecond.

Dynamic

A Dynamic structure will be slower compared to static (about one tenth of the speed) since it has to check if it needs to expand or contract each time. Dynamic structure are also not be as stable from time to time since it will some times need to copy the array over to a new smaller or larger array.

Try	**Min**	**Median**	Mean	**Max**
1	23.8	27.0	33.7	134.4
2	22.4	34.0	37.2	141.7
3	24.8	33.5	38.5	239.2
4	24.7	34.9	40.0	164.4
5	24.6	35.1	42.6	417.4
6	23.5	27.9	38.5	309.2
7	23.0	29.1	38.3	155.3
8	23.8	29.4	37.5	282.5
9	21.8	31.0	38.4	361.1
10	22.4	27.1	38.5	184.6

Table 2: Min, median, mean and max time for a dynamic structure, all times are in microsecond.

No array dynamic

In this example it always changes the size to be exactly what its using at that time this means its really slow but it never expands and takes up unneeded space. It takes about double the time compared to the Dynamic structure.

Try	**Min**	**Median**	Mean	**Max**
1	52.5	60.8	65.7	187.6
2	50.7	55.3	60.2	201.0
3	50.9	55.3	64.6	166.3
4	50.5	58.3	63.4	194.2
5	51.8	59.6	66.1	239.2
6	50.7	54.6	60.2	140.1
7	50.9	55.1	65.3	200.3
8	50.5	60.7	66.3	208.8
9	51.2	54.4	62.2	155.5
10	50.6	54.6	62.0	287.1

Table 3: Min, median, mean and max time for a non array dynamic structure, all times are in microsecond.

Conclusion

A static structure is optimal when it's a known max size of the same data type, it's also useful when you have a strict time/processing limit and an abundance of memory, since the static structure has the lowest time/processor cost.

A dynamic structure is useful when working with unknown data sizes or changing sizes of the same data type. It's also useful for a situation with a tighter limit on memory usage. It's general a good all around solution with more processing needed but can use less memory compared to a static structure.

The non array structure has limited benefits sine it uses double the time compared to dynamic array structure and about 20 times slower compared to the static. In the way I designed the non array structure it is better for expanding to multiple different data types in on the same stack item.

Missing case for optimising

If you need a solution for a low processing, high memory stack with multiple data types a good option is to use multiple linked static structures with the same length and top value but different data types and stored values.