CMPT 225 Assignment 2 Experiment Report

Gurveen Nanua
301546459

Question: Compared to simple arrays and linked lists, are unrolled linked the best?

General Understanding: Traversal of a list in an ULL will be much faster than traversal of the same list in a simple LL, but not too much slower than a traversal of the same list in an array.

Note: I made changes to the node capacity for the ULL when I implemented the experiment, but the default node capacity = 10. (for test file of the ULL, etc)

List Size:

- To find how much my cache holds on my computer I ran the command: sysctl -a | grep hw.l1 in my terminal, then added the two numbers.
- To estimate an appropriate size for my lists, (where there is guaranteed to be a cache miss) I divided the number by the memory size of an int (4 bytes) and then multiplied it by 10.

```
2
3   using namespace chrono;
4
5   //const int SIZE = 100000;
6   const int CACHE_SIZE = 196608; // using command sysctl -a | grep hw.l1 in terminal to find cache size
7   const int INT_SIZE = sizeof(int);
8   const int TEN = 10;
9   const int SIZE = CACHE_SIZE/INT_SIZE * TEN; //muliply by number to garuntee chache misses
0
```

Here are my findings:

CACHE_SIZE = 196608
INT_SIZE = sizeof(int) = 4
SIZE = CACHE_SIZE / INT_SIZE * 10 (approx 491,520)
Node capacity = 100 (unrolled linked)

Inserting elements into the list:

- I inserted the elements 0 < i < SIZE into the array, linked list, and unrolled linked list

	array	LL	ULL
Average time of inserting a value	0.00266702	309.0955	0.2974175

I've observed that the array insertion time is the fastest, ULL the second most and LL being the slowest, taking about 5 minutes for all the elements to be inserted.

Finding an item the list:

- I traversed each list to find the value of SIZE-1.

	array	LL	ULL
Average time of finding the item	0.0017007	0.003602	0.000000445

I've observed the ULL is significantly faster than the other lists, this could be because the element I was searching for could've been in one of the first partially filled nodes, making the run time of c.find(SIZE-1) function the fastest. So in my case, the ULL is the best for searching for an item generally found at the end of the array or linked list. Also, the simple array is about 2 times faster than the LL.

traversing each element in the list:

- I traversed every element in the list by printing them. (which is a lot of values)

	array	LL	ULL
Average time of traversing the list	0.941013	0.433216	0.380787

I've observed that the ULL is the fastest at printing its elements. It is about 1.13 times faster than the LL and 2.5 times faster than the array. This is because the node capacity is 100, so each node can hold more elements and print them quickly.

I also experimented where the node capacity = 10. Here are my findings:

Inserting values

	array	LL	ULL
Average time of inserting a value	0.00482625	313.60	0.0572351

Insertion time for ULL is almost as fast as array time.

Traversing the lists and finding a value

	array	LL	ULL
Average time of finding the item	0.0029679	0.00128786	0.000000361

ULL is still very fast because the element could be in one of the first nodes.

Printing the list traversal:

	array	LL	ULL
Average time of traversing the list	1.00369	0.390495	0.46972

Here the ULL is just as slow, (actually a bit slower) than the LL, because the node capacity is smaller so it behaves more like a LL. the array is still the slowest in traversing each element, about 1 min.

Conclusion:

- The runtimes of each list depends on the type of operation being performed and the parameters such as size and node capacity being set.
- For **inserting elements** into the list, it showed that the array performed the fastest, followed by ULL, and LL resulted with the slowest insertion time. However, the runtime of the insert function for ULL largely depends on the node capacity. A higher capacity resulted in slower insertion rate than with a smaller capacity.
- For **finding an element** in a list, the ULL was the fastest with respect to each node capacity, my suspicion is that the item being searched was found within the first couple of nodes, so I failed at traversing nearly the entire list. This means it's faster to find elements at the end of a list in an ULL and in a simple array or LL.
- For **traversing the entire list** and printing out each element, the ULL was fasterest when its node capacity was 100, but behaved more like a LL when its capacity was at 10. Overall, the array traversal time was the slowest at printing out each element. And LL being the second slowest, but more closer to the traversal time of the ULL.

Overall, my findings in this experiment demonstrates that even though arrays are the most efficient in inserting items, and LLs are moderate in some scenarios, the ULLs seem to be the optimal choice. Depending on the node capacity, it can behave like an array list for insertions and is fastest in traversing the list. Therefore, this data structure is the better option than operating with a simple array or a LL.