iset64: A Set Class

Nikhil Ragde #12

June 2018

iset64: A Set Class

A brief description of the data structure decisions for the `iset64` class.

1 The Data Structure

There were multiple options for the data structure to be used in an implementation of the `iset64` class. I chose to use a fixed-length, dynamically allocated `int` array.

1.1 Pros

- The indices directly represent the numbers in the set. A value of zero at the index means it is not a part of the set, and a non-zero value means it is.
- Algorithms can loop through the array without having to keep track of what is and what is not already seen.
- The implementation of this data structure is trivial. It doesn't require any fancy memory allocation or deletion.
- It can be easily extended out to include more values in the set, provided these integers are >= 0.
- The algorithms for the overloaded operators are very simple because of the simplicity of such a data structure. They are all essentially a matter

of checking whether an index is non-zero or not and then performing some sort of action as a result of that.

1.2 Cons

- This does not extend out well to sets that do not just span from 0 to 63.
- There is "wasted time" in algorithms checking all of the empty values, especially if the only non-zero element is at 63.
- The order of numbers in a set is completely lost without adding additional data structures to the class. The integers are printed out from the lowest to the highest integer, regardless of how they were ordered when the object was instantiated.

After completing the class, I realized that an array of `bool` elements would have been even better, as this would prevent any accidental cases of including a number multiple times. However, I did not go back and change things because of the method in which I checked each index. In something like the print function, for example, I check whether the index is zero or non-zero. In the case of the increment and decrement functions, I explicitly set indices to zero. As a result, I'm effectively treating the `int` values as `bool` values, thus preventing the case of a duplicated integer in a `iset64` object.

2 Other Options

The other potential data structures added complexity that did not seem necessary for this version of this project. A linked list, for example, would help preserve the order of numbers when printing. But this is one of the rare "pros", as there would be significantly more complex algorithms required to overload operators. If a linked list was to be used, a new class would have to be created as well just to implement the data structure. Issues with the custom data structure, then, could stack up on top of issues in the `iset64` implementation.

This ended up being the case when contemplating most alternative data structures. Because the parameters of this class explicitly states that the integers can only be between 0 and 63, inclusive, the simplest data structure was chosen. Its pros greatly outweigh its cons, and it was almost a given that it would be better to use compared to any alternative, custom data structure.

3 Implications of the Data Structure

The use of the fixed-length, dynamically allocated int array means that it is possible to have the “worst-case” performances be equal to the length of the array (i.e. 64 elements). Due to the way that operators behavior has been implemented, the loops that increment, decrement, invert, etc. are effectively executed in “constant time”. This is true because the loops only have to pass through the array once, completely. In the case of incrementing or decrementing, there may be one more addition after this to handle the ”rollover” case, but even that is bound by the length of the array such that the worst case number of operations would be 65 on the array.

The ”best case” algorithm would be for something like the boolean operator. The implementation of that operator would return `false` upon checking the very first index if “0” was included in the set.

The ~operator is also simple to implement with this data structure. All zeroes become non-zero and all non-zero elements are set to zero.

The drawback of using this type of data structure, however, is that the order of the set is lost after the object is instantiated. The print function is implemented in such a way that it will always print out in a sorted manner, from the lowest to the highest integer. This could be changed, however, if additional strings and arrays were added to the class. This can be handled in future enhancements, if necessary.