Advanced Datastructures: Project 2

Jan H. Knudsen (20092926)
Roland L. Pedersen (20092817)
Kris V. Ebbesen (20094539)

November 12, 2013

Contents

1 Implementation Details

1.1 The Basic Van Emde Boas Tree

The Van Emde Boas Tree was implemented following the description in Cormon et al.[1]. This version of the tree is fairly basic, but provides the full features of the Van Emde Boas tree.

A few design choices of note are:

The Base Case When the Van Emde Boas Tree reaches 1 bit required for its universe (a 2-element universe), we skip most of the normal operations done in the tree, and maintain the tree fully in the `min` and `max` values of the tree.

Element Data Type Since we need to support integers of up to 24 bit, we store data values as unsigned 32 bit integers.

The `null` Element When we wish to signal that the `min` or `max` element is empty, we set its value to -1, which when converted to an unsigned integer will be the maximum value possible, and outside of the 24 bit universe.

Specifying Universe Size When the Van Emde Boas recieves the information about the required bits for its input, is does it by a templated variable. This ensures full compiler optimization of branches based on bit size, and saves storage space.

Splitting the Elements When elements are split into a high and low part, they are done so by selecting the high and low half of the bits. In case of an uneven amount of bits, the high part receives the most bits.

Input Validation When testing the tree, assertions check that we do not specify values that are outside of the universe. These assertions are disabled when doing performance testing.

1.2 Making a Heap based on the Van Emde Boas Tree

Implementing a heap on top of a Van Emde Boas Tree is fairly simple. Since we already know that a Van Emde Boas Tree requires storage space linear to the universe size, we simply allocate an array of the size of the universe, and use this for storage, while using the tree to store any keys.

Now most heap operations can be performed by doing a single operation on the Van Emde Boas Tree, and a single read or write on the array.

Note that DecreaseKeys are not supported, since they make little sense on a Van Emde Boas Tree. Instead, if one wishes to decrease a key, simply do an insert and a delete.

1.3 Small Universe Van Emde Boas Trees

When the universe size of a Van Emde Boas Tree falls below 6 bits, we can store the entire tree in a single 32 bit integer. Any operation on the tree can now be performed using a constant number of bitwise operations, assuming we can find the first and last bit set in constant time. Luckily, most processors support these operations.

This should save not only space, but also execution time.

Note that we store the 32-bit integer in the space normally holding the pointer to the top tree, to save memory. Also, since we specify the number of bits required for the universe, we get full compiler optimization for the branches determining whether or not to use integer instead of the normal Van Emde Boas tree operations.

1.4 Future Improvements

Our Van Emde Boas Tree lacks an important optimization often used in practical implementations of the data structure, which is the on-demand storage of the subtrees. Basically, the idea behind this optimization is to only store subtrees that actually contain elements. This is done by keep a map on the subtrees rather than an array, and adding or removing trees as needed.

This however requires HashMaps that grow and shrink as needed, which the unordered sets of the standard library does not. Also, this might make the worst-case performance of the tree worse, if the map does not do these operations in constant worst-case time.

2 Tests

2.1 Experiment Details

The tests were run on a laptop using a few scripts to automate the testing and formatting of data. Valgrind with the Massif heap profiler was used to obtain data about the size of memory used during testing. This heap profiling was done in a seperate test run to not affect the running times of the main test. Each test was run with various amounts of operations, all powers of 2:

$2^{18} = 262144$, $2^{19} = 524288$, $2^{20} = 1048576$, $2^{21} = 2097152$, $2^{22} = 4194304$, $2^{23} = 8388608$ and $2^{24} = 16777216$. The number of operations is given to each test as input parameter n.

2.1.1 Test Types

Three types of test was done:

TestInserts Testing the running time of Insert operations by measuring the time it takes to insert elements $n - 1$ to 0 in that order.

TestDeleteMin Testing the running time of DeleteMin operations by first inserting elements like in the TestInserts test without measuring the time and then running DeleteMin the same amount of times while measuring time.

TestInterleaved Testing the running time of interleaved Insert and DeleteMin operations by first inserting elements $n - 1$ to $\frac{n-1}{2}$ elements without measuring time, and then interleaved calling DeleteMin and Insert on elements $\frac{n-1}{2}$ to 0 while measuring time.

2.1.2 Computer Specs:

OS:	Ubuntu 13.10
CPU:	Intel Core i7-3517U – Quadcore – 1.90 GHz
Memory:	6 GB

2.2 Expectations

We expect the Van Emde Boas Trees to perform quite well in terms of speed. Their upper bound on comparisons and general operations of $O(\log(\log U))$, will most likely make them outperform the more general heaps, especially as data sizes grow.

In terms of memory, we known that the Van Emde Boas Trees will use a large constant amount of memory, though we expect the Van Emde Boas tree with bit magic will use a smaller amount of memory than the other heaps as the universe grows full, due to not actually storing the entire value of the elements.

2.3 Data

TODO Jan

2.4 Results

We see that our experiments behave much as we expected. The Van Emde Boas Tree based heap are much faster than the ordinary heaps, and using bit magic gives a small, yet significant speed boost. They even outperform Fibonacci Heaps on inserts, which is quite impressive.

In terms of memory we can see that the ordinary Van Emde Boas Tree uses a huge amount of memory, only being surpassed by the Fibonacci Heap at a full universe. The bit magic Van Emde boas tree on the other hand uses a quite acceptable amount of memory, being roughly equal with the other heaps at about 2 million elements.

References

[1] Cormen et al. *Introduction To Algorithms.* MIT Press, 2009, Third edition.