Practical vEB Tree

The goal of this project is to implement a version (variant) of van Emde Boas tree that can outperform standard (comparison based) containers.

Currently we're testing our implementation against `std::set` and `__gnu_pbds::tree`. We may add hand-written data structures for comparisons at a later stage. We focus particularly on three operations that van Emde Boas trees natively support: `insert`, `delete`, `successor`.

Scenario 1

Keys are integers in $U = [0, 2^{30})$.

Implementations are tested on 2^{25} random operations. 30% insert, 15% delete, 55% successor.

Implementation Detail (full_veb.hpp)

- Based on the $O(|U|)$ space simplified version discussed in-class
- Use template meta programming to help optimize constants
- When $|U|$ is small enough to fit into a word ($|U| \leq 32$), we encode the existence of elements simply into a word (similar to `std::bitset`), lowering the space to $\sim |U|/32$ and vastly improve constant factor

Testing Detail & Environment (test_1.cpp)

Random operations are first generated as described. Implementations are tested on the same set of operations and results are checked.

Testing environment: Windows, Laptop with CPU Intel i7-7700HQ @ 2.80GHz, MinGW x64 (`-ofast`)

Result Summary

Key space	# of Operations	Final size	set	pb_ds	*vEB*
$2^{30} = 1073741824$	$2^{25} = 33554432$	9991769	48.36s	39.91s (1.21x)	7.60s (6.36x)

Even including the 1.23s took to build the giant vEB tree, this version of vEB tree still got a 6.36x speedup against `std::set`.

Scenario 2

Keys are integers in $U = [0, 2^{60})$ (so we cannot really afford to build the whole vEB tree).

Implementations are tested on 2^{25} random operations. 30% insert, 15% delete, 55% successor.

[Work in progress...]