Adv. Data Structures: van Emde Boas Trees

Anders Ingemann (20052979)
Peter E. Hvidgaard (20062546)

November 21, 2011

Project

yadyyady om projektet praktiske ting som hvor kildekode ligger hvilken test maskine vi bruger hvordan man compiler og kør stødte vi ind i nogen problemer andet

van Emde Boas Trees

The van Emde Boas Tree data structure is recursive, this means we have to decide when to end the recursion. A natural starting point is to let the recursion end with leafs consisting of the 2 elements, *min* and *max*, but since a tree with 3 elements will have a TOP and $BOTTOM$ with a single element, that must also be supported. However, as stated in the note from G. Frandsen, and at the lectures, there is a point where you will gain better performance by reverting to an array and a liniar scan. Our implementation allows for this *threshold* to be set upon creation of the tree, or by passing a value of -1 to let the system set the size such that, the leaf will fit inside a single memory page on the system. Normally, a page are of size 4 KiB, but it's possible to have a page size 2-4 MiB - therefore the default value will not make the leafs larger than 4KiB. Our hope is that this will improve performance, since the system can load the entire page to the CPU cache and scan it.

Supported operations

Our implementation support the following operations on a vEB tree:

- $uint32\_t$ **veb_insert**$(uint32\_tindex, void * data, vebtree * tree)$
- $void$ **veb_delete**$(uint32\_tindex, vebtree * tree)$
- $int32\_t$ **veb_findsucc**$(uint32\_tindex, void * data, vebtree * tree)$
- $int32\_t$ **veb_findpred**$(uint32\_tindex, void * data, vebtree * tree)$

delete_min and **find_min** can both be implementeted with the above operations, by first calling **veb_findsucc** on 0, and if you want to delete it, call **veb_delete** on the returned index.

Handling the leafs

Our leafs are a simple array of elements and as such will not be explained in more detail than this part. The operations to **insert** and **delete** are a matter of inserting and deleting in the correct place in the array, this is done in constant time. The operations **find_pred** and **find_succ** are a matter of scanning the array from a certain posistion to find the first, if any, element that is before or after. Determining *min* and *max* is also done by scanning the array. Since the arrays have a fixed length, independent from the size of the universe, **find_pred**, **find_succ** and determining *min* and *max* are constant time.

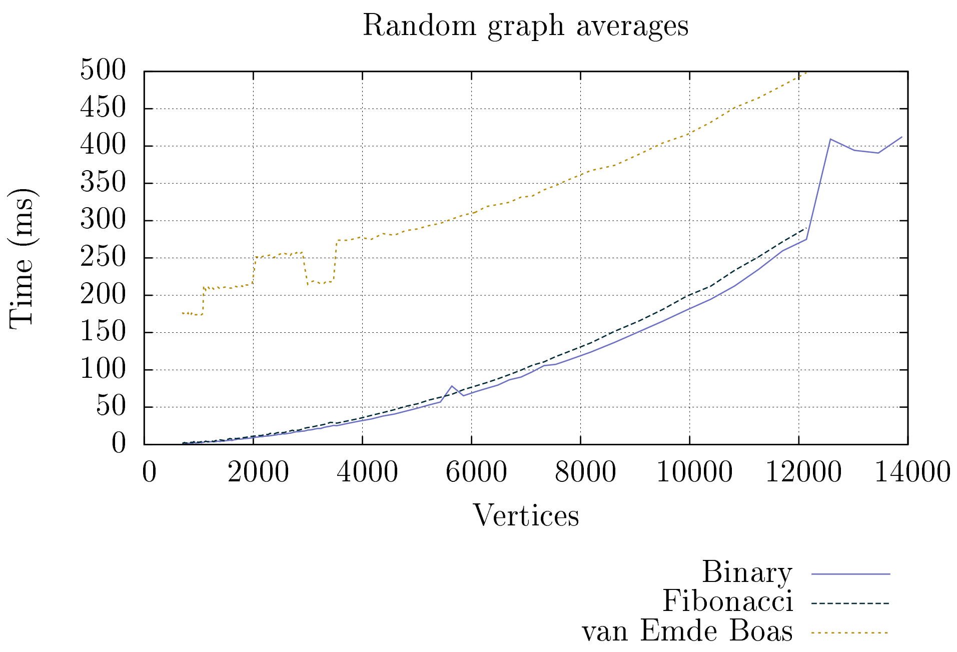
Handling the recursive nodes

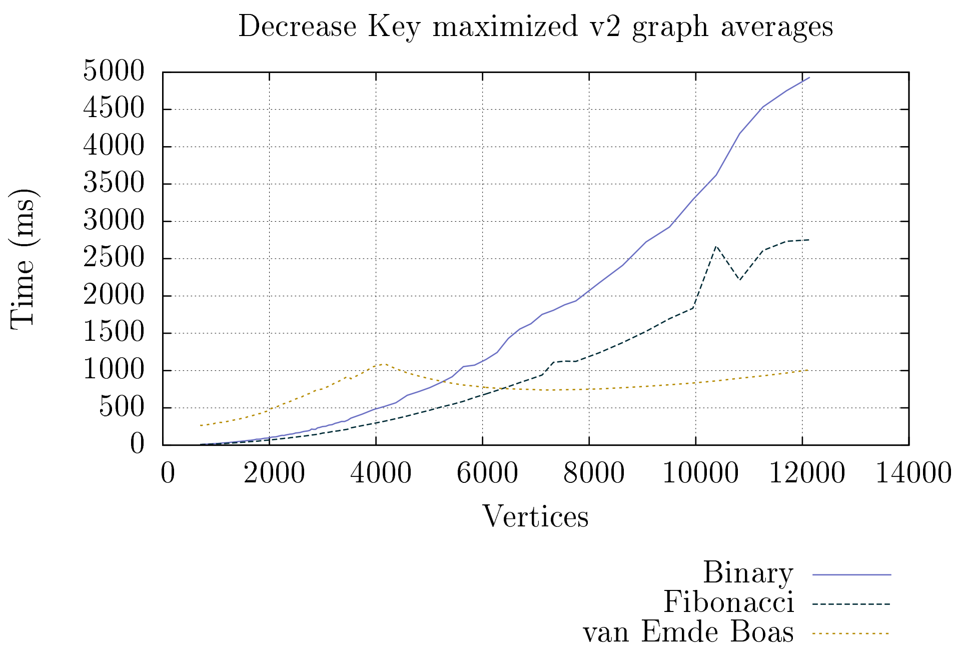
The trees are build as described on the slides from the lectures - and **insert**, **delete** and **find_succ** are all implemented as described on the slides as well. The operation **find_pred**, is mostly identical to **find_succ**, but obviously look for the first element preceeding the index.

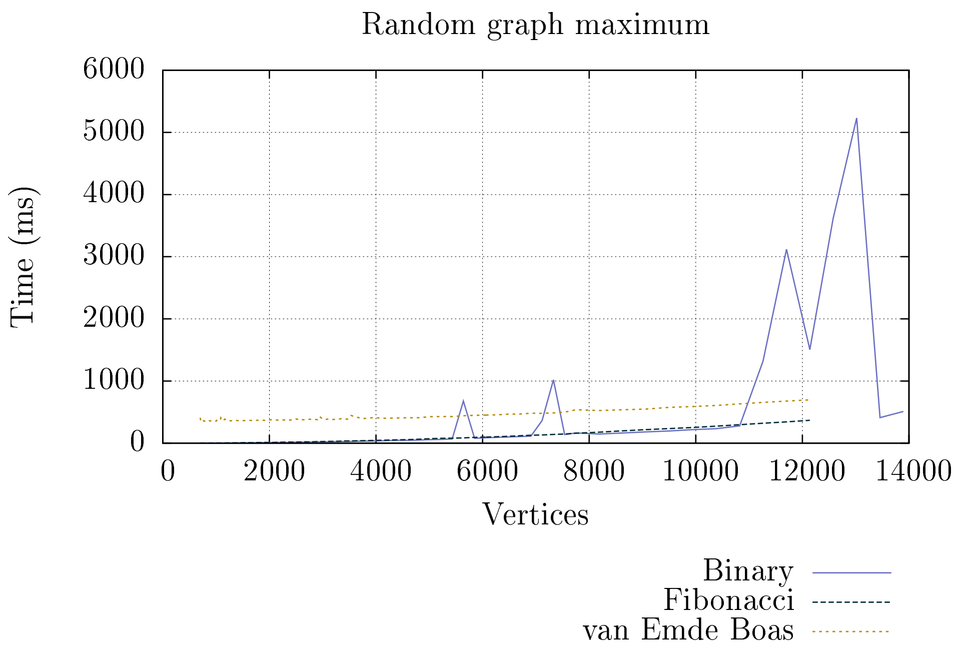
Comparison with priority queues

kort om BinHeap og FibHeap

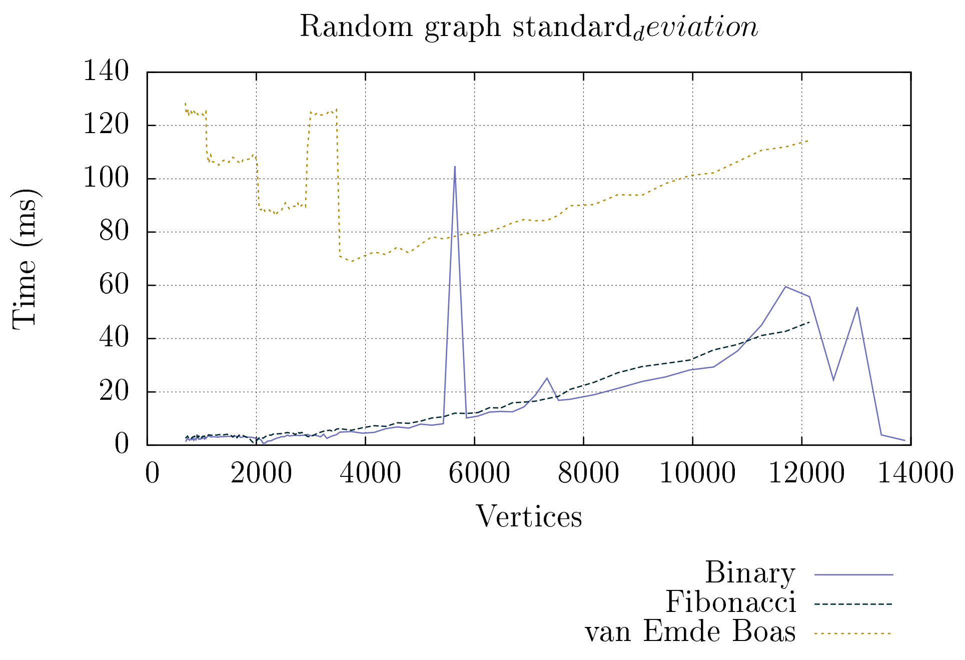
We have chosen to benchmark the van Emde Boas tree with the two graph generation algorithms we implemented in the previous assignment. Those include a random graph generator and a generator which should cause a high number of decrease_key calls when using dijkstra on the graph to solve the single source shortest path problem. The random graph generator generates a graph, where each vertice has a 15% chance of being connected to another vertice with the weight being between 1 and a chosen maximum.



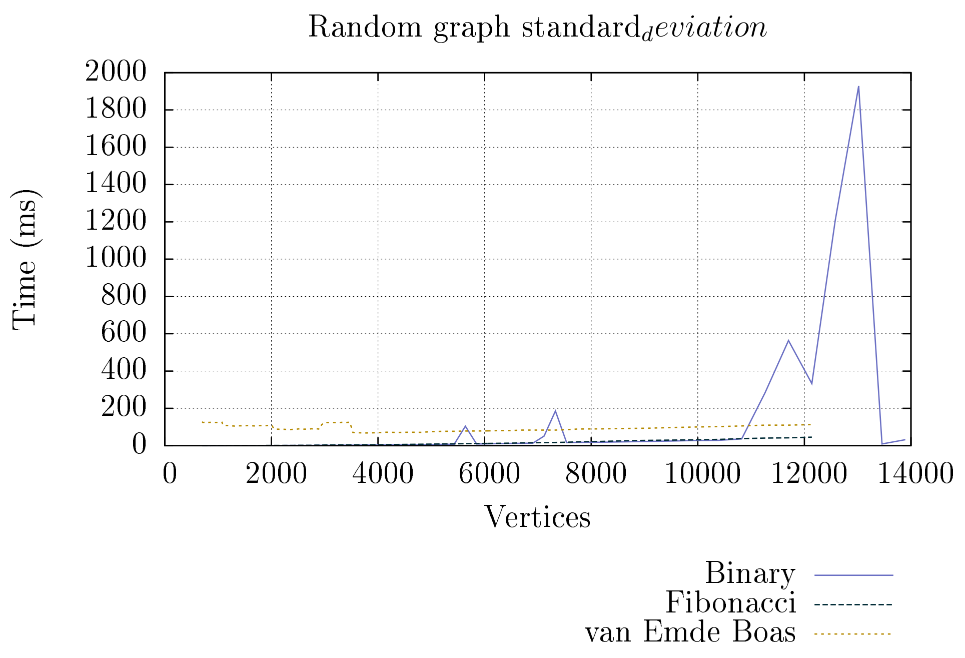




We measured the running time using two different methods. One using normal absolute time measurement, the other one counting clock cycles. Curios to see which method had the lowest standard deviation we were surprised to discover that it increases using both methods quite rapidly as the graph sizes grow bigger.



The only exception being the random graph, when counting clock cycles.



We have yet to explain why this is happening.

hvilke test cases har vi og hvorfor? test og plots konklutioner på testne

Comparison with Red-Black Trees

Kort om red-black trees hvilke test cases har vi og hvorfor?

In order to compare Red-Black trees to van Emde Boas trees we constructed a simple sorting benchmark. To do that we implemented a list generator, whose product should be sorted by either algorithm. The generator generates a list of random numbers from 0 to a given maximum and terminates once a given size is reached.

test og plots konklutioner på testne

Appendix

6

Bibliography

[1] Thomas H. Cormen et. al., *Introduction to algorithms*, 2nd Edition.