THE UNIVERSITY OF MELBOURNE

COMP20007
Project Report 1

Design of Algorithms
March 30, 2018

Report project 1:
Search Engine Algorithms

1 Introduction

This report analyses the functioning and implementation of a search engine. Two different approaches are implemented for this task. The aim of this report is to provide a better understanding of the algorithms implemented in each approach.

First of all an analysis of the time complexity is performed regarding the two approaches. Next empirical evidence is provided for the effect of various input configurations and lastly the performance of each algorithm is discussed.

2 Time complexity analysis

The first query algorithm is based on initializing an array of size n with values set to 0, which takes $\mathcal{O}(n)$ time. For all terms m in the query, the documents are iterated and their scores are added to the array with index corresponding to the document ID. Each term in the query can contain n documents, so the running time of this operation is $\mathcal{O}(nm)$. Then a top-k algorithm is applied to find the documents with the highest k scores. For this algorithm the top-k heap is updated each time an element i in the array is larger than the root. With a probability of $\frac{k}{i}$ that the heap needs to be updated, this means that on average the heap is updated $\sum_{i=0}^{n-1} \frac{k}{i} \leq k\log(n)$ times. With one update of the heap taking $\mathcal{O}(\log(k))$, this operations runs in $\mathcal{O}(k\log(k)\log(n))$. In the worst case, however, the heap must be updated at every element, which means that the top-k algorithm becomes $\mathcal{O}(n\log(k))$. Finally, the top-k heap needs to be sorted using Heapsort, which has $\mathcal{O}(k\log(k))$ complexity.

Adding all these operations together means that on average the first searching algorithm has a complexity of $\mathcal{O}(n) + \mathcal{O}(nm) + \mathcal{O}(k\log(k)\log(n)) + \mathcal{O}(k\log(k))$. Because $n \leq nm$ and $k\log(k) \leq k\log(k)\log(n)$ this reduces to a time complexity of $\mathcal{O}(nm + k\log(k)\log(n))$. The worst case scenario, on the other hand, reduces to a time complexity of $\mathcal{O}(nm + \log(k)n)$.

The second algorithm uses a heap for ordering the document lists by document ID, which takes $\mathcal{O}(m)$ time to initialize. The heap is continuously updated until all documents in each list are traversed. This means that this heap is updated nm times which has time complexity $\mathcal{O}(nm\log(m))$. While traversing the documents, the score values with identical document ID are added and inserted in the top-k heap. Again, this heap is updated on average $k\log(n)$ times and in the worst case n times, with each update taking $\mathcal{O}(log(k))$ time. Lastly, the heap is sorted using Heapsort, which has a complexity of $\mathcal{O}(k\log(k)$.

Thus, adding all operations results in an average time of $\mathcal{O}(m)+\mathcal{O}(nm\log(m))+\mathcal{O}(k\log(k)\log(n))+\mathcal{O}(k\log(k)$, which reduces to $\mathcal{O}(nm\log(m) + k\log(k)\log(n))$ due to dominance relations. The worst case scenario results in a time of $\mathcal{O}(m) + \mathcal{O}(nm\log(m)) + \mathcal{O}(n\log(k)) + \mathcal{O}(k\log(k)$ and this reduces to a time complexity of $\mathcal{O}(nm\log(m) + n\log(k))$. All results are summarized in Table 1.

These results would indicate that the array based algorithm has a smaller asymptotic complexity, because $\log(m) \geq 1$. However, this assumption holds for values of m, k and n that are asymptotically large. In reality the values of m and k are usually relatively small (say between 1 and 100) when compared to n. The effects of these parameters are therefore evaluated in the next section.

Table 1: Asymptotic complexities for search algorithms

	Average Case	Worst Case
Array based	$\mathcal{O}(nm + k\log(k)\log(n))$	$\mathcal{O}(nm + \log(k)n)$
Priority queue based	$\mathcal{O}(nm\log(m) + k\log(k)\log(n))$	$\mathcal{O}(nm\log(m) + n\log(k))$

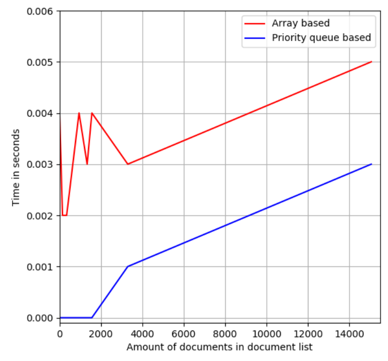
3 Empirical input analysis

The several input parameters also play an important role in the performance of the two different algorithmic approaches. This section discusses the effect of the number of results k, the amount of terms m and the amount of documents associated to each term (Which is a fraction of the n total documents).

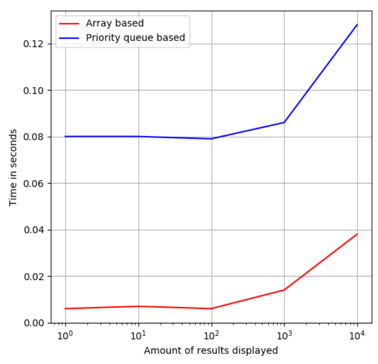
Figure 1a shows the relation between the amount of documents in the list and the computation time. It can be observed that the computation time of the priority queue based algorithm is less than that for the array based algorithm for every value of documents in the document list. This can be explained by the fact that the array based algorithm traverses an array of length n for each term in the query. The priority queue based algorithm, on the other hand, only traverses the documents which are contained in the document lists. This means that the amount of operations per term performed by the priority queue based algorithm is always less than or equal to n , while the array based algorithm always performs n operations per term to fill the array.

Furthermore, the effect of the number of results k is analyzed. The influence of the results can be addressed to the constant updating of the heap for the top-k algorithm. Although both approaches update the top-k heap on average a similar amount of times, Figure 1b shows that the computation time for the first approach is always smaller than that of the second. However, the slope of both curves is approximately equal. Therefore, the effect of the amount of results could be stated to be similar for both approaches. The larger computation time for the priority queue is most likely due to the effect of the other parameters.

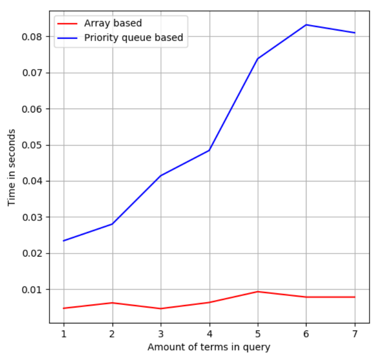
Lastly, Figure 1c shows the effect of an increasing amount of terms in the query. The computation time for the second algorithm is constantly larger than the first and increases more rapidly as well. This could be explained by the fact that the amount of terms has a bigger impact on the asymptotic complexity of the priority queue based algorithm. After all, the priority queue needs to be updated for all documents in the document lists. It holds that this operation has $\mathcal{O}(nm\log(m))$, whereas the array based algorithm performs the filling of the array in $\mathcal{O}(nm)$ time.



(a) Time effect of documents



(b) Time effect of result numbers



(c) Time effect of term numbers

Figure 1: Effect of input parameters on computational time, with time displayed as mean values over 10 iterations

4 Conclusion

This report discussed the effect of applying different algorithms for the implementation of a search engine. For analyzing the computation times of the algorithms, the theoretical and empirical effects of several parameters were discussed.

Considering the different consequences of the input parameters, a conclusion can be drawn about the use of each algorithm. Because of the smaller effect of the amount of documents in the document list on the priority queue algorithm, this approach should be implemented for queries with small amount of terms that are rather rare (e.g. "cryptocurrency" or "omnipotent"). The amount of terms in the query, on the other hand, had a much larger effect on the priority queue based algorithm. Therefore, the array based algorithm should be a viable option when implementing a query with many relatively common terms (e.g. "the", "this", "with", "hello", "world").

To summarize, it can be stated that each algorithm has its advantages and disadvantages which should be considered given the circumstances. When taken these considerations into account, both approaches can be viable options for implementing a search engine.