Approximately Optimum Search Trees in External Memory Models

by

Oliver Grant

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We examine optimal and near optimal solutions to the classic binary search tree problem of Knuth [36]. First, we re-examine a solution of Güttler, Melhorn and Schneider [26] which was shown to have a worst case bound of $c \cdot H + 2$ where $c \geq \frac{1}{H(\frac{1}{3},\frac{2}{3})} \approx 1.08$. We give an improved worst case bound on the heuristic of $H + 4$. Next, we examine the optimum BST problem under a model of external memory. We use the Hierarchical Memory Model (HMM) of Aggarwal et al. [2] and propose two approximate solutions which run in $O(n)$ and $O(n \cdot \lg(m_1))$ time where n is the number of words in our data set, and m_1 is the size of the smallest memory in the memory hierarchy. Using these methods, we improve upon a bound given in Thite's 2001 thesis under the related HMM_2 model in the approximate setting. We also examine the optimum BST problem over a multiset of probabilities and provide a simple algorithm which proves to be optimal.

Acknowledgements

I would like to thank all the little people who made this possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we provide an introduction to binary search trees and the optimum binary search tree problem. We also give motivation for studying binary search trees and give an overview of the work presented in this thesis.

1.1 Binary Search Trees

A binary search tree is a simple structure used to store key-value pairs. It was invented in the late 1950s and early 1960s and is generally attributed to the combined efforts of Windley, Booth, Colin and Hibbard [50, 12, 27]. In general, a binary search tree (BST) allows for quick binary searches through data for a specific key. There is a total ordering over the keys of the tree and are typically numbers or words. The value of a node in the BST usually represents some piece of important information, and is often a pointer to large structure somewhere else in memory. Each BST node has at most two children which are generally labelled as the *left* and *right* children. All nodes in the subtree of the *left* child of a specific node p have a key strictly less than the key of p. Similarly, nodes in the subtree of the *right* child of p have a key strictly greater than the key of p. A pointer is typically stored to the root node. Search begins from this root node and is done by recursively searching in either the *left* or *right* child of a node; stopping if the node being searched has the correct key, or if the node reached has no children.

1.2 The Optimum Binary Search Tree Problem

Knuth first proposed the optimum binary search tree problem in 1971 [36]. We are given a set of n keys (originally known as words), $B_1, B_2, ..., B_n$ and $2n+1$ frequencies, $p_1, p_2, ..., p_n,\ q_0, q_1, ..., q_n$ representing the probabilities of searching for each given key and the probabilities of searching in the gaps between and outside of these keys. We have that $\sum_{i=0}^{n} q_i + \sum_{i=1}^{n} p_i = 1$. We also assume that without loss of generality that $q_{i-1} + p_i + q_i \neq 0$ for any $i \in \{1, ..., n\}$. Otherwise, we could simply solve the problem with key p_i removed. The keys (and lexicographic gaps between them) are used as keys and the alphabetic ordering of them provides our order over the keys. The keys must make up the internal nodes of the tree while gaps make up the leaves. Our goal is to construct a binary tree such that the average weighted path length over the nodes is minimized. This weighted average path length is also referred to as the expected cost of search and is formally defined as:

$$P = \sum_{i=1}^{n} p_i \cdot (d_T(B_i) + 1) + \sum_{j=0}^{n} q_j \cdot (d_T(B_{j-1}, B_j)) \tag{1.1}$$

where p_i and q_j are the probabilities of searching for key B_i or gap (B_{i-1}, B_i) respectively and $d_T(B_i)$ and $d_T(B_{j-1}, B_j)$ are the depths of B_i and (B_{j-1}, B_j) respectively in the tree T. The optimal solution of Knuth requires $\Theta(n^2)$ time, and $\Theta(n^2)$ space. This solution is both time and space intensive. We will later examine an approximate solution to this problem of Güttler, Mehlhorn and Schneider (the Modified Entropy Rule) which uses $O(n^2)$ time but $O(n)$ space and improve its worst-case bound [26]. However, these problems were examined under the RAM model which is an inadequate model for many situations. We examine the problem in more realistic models and look at approximate solutions under these settings.

1.3 Three-Way Branching

While modern computers typically only support two-way branching, the optimum BST problem proposed by Knuth is uses the three-way branch model. This model allows a single comparison operation to transfer control to three different locations.

Examples of this can be seen in FORTRAN IV which describes the arithmetic IF [18]:

```
IF (EXPR) LABEL1, LABEL2, LABEL3
```

Control is transferred to LABEL1, LABEL2 or LABEL3 if $expr < 0$, $expr = 0$, or $expr > 0$ respectively using a single comparison command. While most modern computers do not support such commands using a single three-way branch, many machines in in the FORTRAN IV era did [] TODO. The difference between a two-way branch model is significant and can be seen through a simple example of searching among 3 keys.

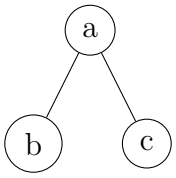


Figure 1.1: A 3 node binary tree.

Assume the probability of search for any of a, b, or c is $\frac{1}{3}$. Under the three-way branch model, this can be done using exactly one comparison by asking if the key we are search for is less than b, equal to b, or strictly greater than b and returning the correct key appropriately. Under a standard two-way branch model, this would require an extra comparison operation $\frac{2}{3}$ of the time as we can only distinguish one of the the three cases from the other two using a single two-way $<, >$, or $=$. We get an expected cost of 1 comparison in the three-way branch model and $\frac{5}{3}$ comparisons in the two-way model. More specifically, each comparison can reveal 3 bits of information in the three-way model, while only 2 bits per comparison can be revealed in the two-way model.

1.4 Why Study Binary Search Trees

Binary search trees are ubiquitous throughout computer science with numerous applications. The basic binary search tree has extended in a number of ways. AVL trees (named after creators AdelsonVelskii and Landis) were the first form of self-balancing binary search trees introduced [1]. This type of tree was invented by the pair in 1963 and maintains a height of $O(\lg(n))$ (where n is the number of nodes in the tree) during insertions and deletions (both of which take $O(\lg(n))$ time). Improved self-balancing binary search trees followed in the form of Symmetric binary B-trees by R. Bayer in 1972 [7]. These are commonly referred to as red-black trees, a term coined by Guibas, Sedgewick and Robert in 1978 [**?**]. Allen and Munro followed with self-organizing binary search trees and examined a move to root heuristic, demonstrating its expected search time was within a constant of the optimal static binary tree [**?**]. The famous splay trees of Sleator and Tarjan followed

in 1985 [43]. Tango trees were invented in 2007 by Demaine et al. and provided the first $O(\lg \lg n)$-competitive binary tree [17]. Here, $O(\lg \lg n)$-competitive means that the tango tree does at most $O(\lg \lg n)$ times more work (pointer movements and rotations) than an optimal offline tree. B trees are among the most commonly used binary tree variant and were invented in 1970 by R. Bayer and McCreight [6].

BST's and their extensions are integral to a large number of applications. For example, a BST variant known as the binary space partition is a method for recursviely subdividing space in order to store information in an easily accessible way. It is used extensively in 3D graphics [42, 41]. Binary tries are similar to binary trees, but only store values for leaf nodes. Binary tries are routinely used in routers and IP lookup data structures [44]. Another example can be seen in the C++ `std::map` data structure, which is usually implemented using a red-black tree (another extension of binary search trees) in order to store its key-value pairs [15]. Finally, syntax trees (trees used in the parsing of various programming languages) are created using binary (and more complicated) tree structures. These trees are used in the parsing of written code during compilation [39].

1.5 Overview

In Chapter 2, we review previous work done in the areas of binary search trees, multiway trees, alphabetic trees and various models of external memory. In Chapter 3, we re-examine the Modified Entropy Rule (ME) of Güttler, Mehlhorn and Schneider [26]. This is an $\Theta(n^2)$ time, $\Theta(n)$ space, algorithm for approximating the optimum binary search tree problem in the RAM model. The method works very well in practice, and the group had great experimental results, but unfortunately they could not bound the worst case expected cost as well as they would have hoped. While simpler solutions like the *Min-max* and *Weight Balanced* techniques of P. Bayer have worst case costs of at most $H + 2$, the trio's ME technique was only shown to have a worst case expected search cost of at most $c \cdot H + 2$ where $c \approx 1.08$ [5, 26]. We provide a new argument of the ME rule's worst case expected search cost and show that it is within an additive factor of entropy: at worst $H + 4$. In Chapter 4, we move on to external memory models, examining the optimum binary search tree problem under the Hierarchical Memory Model of Aggarwal et al. [2]. We provide two algorithms which run in times $O(n \cdot \lg(m_1))$ (m_1 is the size of the smallest memory level in the memory hierarchy) and bound their worst case expected costs. We show that the solutions provided both give a direct improvement over a solution of Thite provided under the related HMM_2 model [45]. In Chapter 5, we consider a variant of the optimum binary

search tree problem (in the RAM model) where the set of probabilities given are from an unordered multiset. we show that a simple approach gives the optimum solution which is unique up to certain permutations. Finally, in Chapter 6, we summarize our findings and discuss several problems which remain open.

Chapter 2

Background and Related Work

In this Chapter we provide an overview of relevant work on binary search trees, alphabetic trees, multiway search trees and models of external memory.

2.1 Binary Search Trees

In 1971, C. Gotlieb and Walker's approximate solution to the optimum binary search tree problem [**?**]. Knuth shortly thereafter gave the first optimal solution [36]. Knuth's optimal solution requires $O(n^2)$ time and space which is too costly in many situations. Several others have since examined the approximate version of the problem. While unable to bound an approximate algorithm within a constant of the optimal solution, many authors have been able to bound the cost based on the entropy of the distribution of probabilities, H. Specifically,

$$H = \sum_{i=1}^{n} p_i \cdot \lg(\frac{1}{p_i}) + \sum_{j=0}^{n} q_i \cdot \lg(\frac{1}{q_j}).$$

In 1975, P. Bayer showed that

$$H - \lg H - (\lg e - 1) \leq C_{Opt} \leq C_{WB}, C_{MM} \leq H + 2$$

where C_{Opt}, C_{WB}, and C_{MM} are costs for the optimal solution, as well as weight-balanced and min-max heuristic methods respectively [5]. Weight-balanced and min-max cost heuristics are greedy and require both $O(n)$ time and $O(n)$ space to run. These greedy heuristics

use a top-down approach where the tree root is selected from among the n keys, and we recurse in both the left and right subtrees. Let $P_L(B_i)$ and $P_R(B_i)$ represent the probabilities of searching for a key before or after key B_i respectively. The Weight-balanced approach, makes this greedy root selection by picking the root B_i such that $|P_L(B_i) - P_R(B_i)|$ is minimized. In 1980, Güttler, Mehlhorn and Schneider presented a new heuristic, the Modified Entropy Rule (ME) [26] which built upon the ideas of Horibe [28]. The Entropy Rule greedily selects B_i as the root such that $H(P_L(B_i), p_i, P_R(B_i))$ is maximized (as discussed in Chapter 3, this was modified to improve its performance). Güttler, Mehlhorn and Schneider gave empirical evidence that the heuristic out-performed others [26]. While the heuristic took $O(n^2)$ time, it only required $O(n)$ space, a huge savings over the optimal solution. However, they were unable to prove that $C_{ME} \leq H + 2$ (unlike previous weight-balanced and min-max heuristics) and settled with $C_{ME} \leq c_1 \cdot H + 2$ where $c_1 = \frac{1}{H(\frac{1}{3}, \frac{2}{3})} \approx 1.08$. We re-examine this method and provide a new bound of $H + 4$ in Chapter 3. In 1993, De Prisco and De Santis presented a new heuristic for constructing a near-optimum binary search tree [16]. The method is discussed in more detail in section 4.6 and has an upper bounded cost of at most $H+1-q_0-q_n+q_{max}$ where q_{max} is the maximum weight leaf node. This method was later updated by Bose and Douïeb (and is also discussed in section 4.6) to have a worst case cost of

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}.$$

Here, $m' = \max(2n - 3P, P) - 1 \geq \frac{n}{2} - 1$ where P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves and. $pq_{\text{rank}[i]}$ is the i^{th} smallest access probability among all keys and leaves except q_0 and q_n.

2.2 Alphabetic Trees

Determining the optimum alphabetic tree is an important related problem relevant later in this thesis. Given a set of n keys (keys $B_1, ..., B_n$) with various probabilities, we wish to build a binary search tree where every internal node has two children, leaves have no children, and the n keys described are the leaves. We wish to build the tree with the minimum expected search cost. The expected search cost is

$$\sum_{i=1}^{i=n} p_i \cdot d_T(B_i)$$

where p_i is the probability of searching for key B_i and $d_T(B_i)$ is the depth of the leaf representing the key B_i in the tree T. The alphabetic ordering of the leaves must be maintained. This is the same as the binary search tree problem with all internal node weights zero.

In 1952, Huffman developed the well known Huffman tree, which solved the same problem without a lexicographic ordering constraint on leaves [33]. Gilbert and Moore examined the problem with the added alphabetic constraint and developed a $O(n^3)$ algorithm which solved the problem optimally [23]. Hu and Tucker gave a $O(n^2)$ time and space algorithm in 1971 [32] which was improved by Knuth to take only $O(n \lg n)$ time and $O(n)$ space in 1973 [35]. The original proof of Hu and Tucker was extremely complicated, but was later simplified by Hu [29] and Hu et al. [31]. Garsia and Wachs gave an independent $O(n \lg n)$ time, $O(n)$ space algorithm in 1977 [22]. This new algorithm by Garsia and Wachs was shown to be equivalent to the Hu and Tucker algorithm in 1982 by Hu [30] and also went through a proof simplification [34] by Kingston in 1988.

In 1991, Yeung proposed an approximate solution which solved the problem in $O(n)$ time and space [51]. The algorithm produced a tree with worst case cost $H + 2 - q_1 - q_n$. This algorithm was later improved by De Prisco and De Santis who created an $O(n)$ time algorithm which has a worst case cost of $H + 1 - q_0 - q_n + q_{max}$ [16]. The method was improved one more time by Bose and Douïeb who improved upon Yeung's method by decreasing the bound by $\sum_{i=0}^{m} q_{\text{rank}[i]}$ where $m = \max(n - 3P, P) - 1 \geq \frac{n}{4} - 1$, P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves and $q_{\text{rank}[i]}$ is the i^{th} smallest access probability among all leaves except q_0 and q_n [13]. Replacing Yeung's method with the improved algorithm of Bose and Douïeb in the De Prisco and De Santis algorithm gave the tightest bound seen so far of

$$H + 1 + \sum_{i=1}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}.$$

2.3 Multiway Trees

Another related problem is the static k-ary or multiway search tree problem. It is similar to optimum binary search tree problem with the added constraint that up to k keys can be placed into a single node, and the cost of search within a node is constant. Multiway search trees maintain an ordering property similar to that of traditional binary search trees. Each key in every page g that isn't the root page must have its keys lie between some keys l and l', two keys located in $g's$ parent. Each internal node of the k-ary tree contains at least one

and at most $k-1$ keys while a leaf node contains no keys and represents searching for an item a gap between keys. Successful searches end in an internal node while unsuccessful searches end in one of the $n+1$ leaves of the tree. The cost of search is the average path depth which is defined as:

$$\sum_{i=1}^{n} p_i(d_T(B_i)+1) + \sum_{j=0}^{n} q_j(d_T(B_{i-1}, B_i))$$

where B_i's represent successful search keys, pairs (B_{i-1}, B_i) represent gaps and $d_T(B_i)$ or $d_T(B_{i-1}, B_i)$ represent the depth of keys or gaps respectively for a key B_i or a (B_{i-1}, B_i) in the tree T.

Vishnavi et al. [46], and Gotlieb [24] in 1980 and 1981 respectively independently solved the problem optimally in $O(k \cdot n^3)$ time. In a slightly modified B-tree model (every leaf has the same depth, every internal node is at least half full), Becker's 1994 work gave a $O(kn^\alpha)$ time algorithm where $\alpha = 2 + log_k 2$ [8]. Later, in 1997, Becker propsed an $O(Dkn)$ time algorithm where D is the height of the resulting tree [9]. The algorithm did not produce an optimal tree but was thought to be empirically close despite having no strong upper bound. In 2009, Bose and Douïeb gave both an upper and lower bound on the optimal search tree in terms of the entropy of the probability distribution as well as an $O(n)$ time algorithm to build a near-optimal tree [13]. Their bounds of

$$\frac{H}{\lg(2k-1)} \leq P_{OPT} \leq P_T \leq \frac{H}{\lg k} + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}$$

is discussed in more detail in section 4.3 of this paper.

2.4 Memory Models

In the typical RAM model, we assume that all reads and writes from memory take a constant amount of time. While this is a valid assumption in many situations (both in the context of theory and programming) when dealing with very large data sets, this is simply not the case. A typical computer has a memory hierarchy with CPU registers, various levels of cache, RAM, SSD and/or hard drives. Each of these memory levels has increasing size but decreasing I/O speed. Typically, the difference between the levels is dramatic (reading from disk takes roughly a million times longer than accessing a CPU register)

[47]. Moreover, many memory hierarchies allow blocks of memory to be moved quickly after a single key or cache line has been accessed. It is thus possible to take advantage of the locality of data during computations. Moreover, it is imperative to consider memory I/O speeds, especially when the dataset being worked on does not fit on internal memory. *External memory* algorithms and data structures refer to those methods and structures which explicitly manage data placement and movement [47]. Various authors have created models to properly reflect the performance of such algorithms and data structures, and we consider the optimum binary search tree problem under such a model.

In Chapter 4, we discuss the optimum binary search tree problem under the 1987 Hierarchical Memory Model of Aggarwal et al. [2]. The model is described thoroughly in section 4.1, but essentially provides an alternative to the classic RAM model. It simulates a memory hierarchy with various memory sizes and different access times for each type of memory. The model does have its shortcomings though as it does not provide us with the ability to move blocks of memory between these different memory types (as in a typical computer). The HMM model was later extended to the Hierarchical Memory with Block Transfer Model of Aggarwal, Chandra, and Snir [3]. This model provides a less artificial setting by allowing for contiguous blocks of memory to be copied from one location to another for a cheaper price. The cost of this copy equal to the cost to access the most expensive location being copied (to or from), plus the size of the block.

As explained in the survey of Vitter, several other models of external memory have followed [47]. Work has been done to consider models with parallelism. Vitter and Shriver built upon the HMM and HMBTM models of Aggarwal et al. [2, 3] in order to allow parallelism [49]. This updated model connects P parallel memory hierarchies at their base memory levels. In 1994, Alpern et al. introduced the Uniform Memory Hierarchy (UMH) [4]. The UMH considers block sizes and bandwidths between memory levels, and allows for simultaneous transfer between pairs of memory levels. The UMH was considered with additional parallelization by Vitter and Nodine [48].

Cache-oblivious algorithms were introduced by Frigo, Leiserson, Prokop and Ramachandran in 1999 [20]. The model used is the ideal-cache model which has a two-level memory hierarchy. The internal memory (named cache) has Z words and the main memory is arbitrarily large. The cache is divided into cache lines of size L and it is assumed that $Z = \Omega(L^2)$ (the tall cache assumption). Frigo et. al examined the fast fourier transform and matrix multiplication under this model. Many others have since used this model and examined problems in a cache-oblivious setting such as the cache-oblivious b-trees of Bender et. al [10], the funnel heap of Brodal et. al [14], or the locality preserving cache-oblivious dynamic dictionary of Bender et. al [11].

A complete and thorough explanation of memory hierarchies (especially those considered before cache-oblivious settings) can be found in the survey of Vitter [47].

Chapter 3

An Improved Bound for the Modified Minimum Entropy Heuristic

In this Chapter we show that the Modified Minimum Entropy Heuristic of Güttler, Mehlhorn and Schneider [26] is within an additive factor of entropy: at worst $H + 4$.

3.1 Preliminaries

Recall 1.1, $H = \sum_{i=1}^{n} p_i \cdot \lg(\frac{1}{p_i}) + \sum_{j=0}^{n} q_i \cdot \lg(\frac{1}{q_j})$. We also use

$$H(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i \cdot \lg(\frac{1}{x_i})$$

to describe the entropy of any probability distribution $(x_1, x_2, ..., x_n)$. For subtree t, we let

$$p_t = \sum_{i:B_i \in t} p_i + \sum_{i:(B_i, B_{i+1}) \in t} q_i$$

be its total probability (the sum of the probability of all nodes within the subtree). $P_L(B_i)$ and $P_R(B_i)$ are probabilities of searching for a key lexicographically before or after (respectively) key B_i. $P_L(B_i, B_{i+1})$ and $P_R(B_i, B_{i+1})$ are probabilities of searching for a key lexicographically before (or equal to) B_i and after (or equal to) B_{i+1} respectively. For a

subtree t $P_{L_t}(B_i)$ and $P_{R_t}(B_i)$ describe the normalized probabilities of searching for a key to the left or right of the root respectively. Formally,

$$P_{L_t}(B_i) = \frac{\sum_{i:B_i \in t} p_i + \sum_{i:(B_i,B_{i+1}) \in t} q_i}{p_t}.$$

$P_{L_t}(B_i, B_{i+1})$ and $P_{R_t}(B_i, B_{i+1})$ have analogous definitions. We let

$$E_t = H(P_{L_t}(B_i), \frac{p_i}{p_t}, P_{R_t}(B_i)) \tag{3.1}$$

be the local entropy of a subtree t rooted at key B_i.

3.2 The Modified Entropy Rule

As described in [26], I first describe the entropy rule for greedy root selection then explain how it was modified. For a subtree t with probability p_t, the entropy rule greedily chooses the key B_i as the root such that $H(P_{L_t}(B_i), \frac{p_i}{p_t}, P_{R_t}(B_i))$ is maximized. While this rule behaves quite well in practice, certain cases cause it to have poor performance (refer to Figure 3.1).

(a) Entropy rule tree: $C = \frac{8}{5}$ (b) Modified entropy rule tree: $C = \frac{7}{5}$

Figure 3.1: Comparison of entropy and modified entropy rule heuristics

Figure 3.1 demonstrates the shortcomings of the entropy rule heuristic. Given the probability set $\{q_0 = \frac{1}{5}, p_1 = \frac{1}{5}, q_1 = 0, p_2 = 0, q_3 = \frac{3}{5}\}$ the entropy rule will mistakenly choose key B_1 as the root while selecting B_2 as the root produces a better tree. This mistake is remedied in the modified entropy rule of Güttler, Mehlhorn and Schneider [26]. The modified entropy heuristic chooses the root in of the following three ways:

a) If there exists key B_i such that $\frac{p_i}{p_t} > \max(P_{L_t}(p_i), P_{R_t}(p_i))$ we always select B_i as the root.

b) If there exists a gap (B_i, B_{i+1}) such that $\frac{q_i}{p_t} > max(P_{L_t}(B_i, B_{i+1}), P_{R_t}(B_i, B_{i+1}))$ then we select the root from among B_i and B_{i+1}. B_i is chosen if $P_{L_t}(B_i, B_{i+1}) > P_{R_t}(B_i, B_{i+1})$ and B_{i+1} is chosen otherwise.

c) Otherwise, B_i is selected such that $H(P_{L_t}(B_i), \frac{p_i}{p_t}, P_{R_t}(B_i))$ is maximized (as in the original entropy rule).

The approach proposed by Güttler, Mehlhorn and Schneider takes linear time and constant space to find the all of the optimal local entropy splits in each level of the tree. TODO $\Theta(n^2)$ total time and $O(n)$ space algorithm.

3.3 ME is Within 4 of Entropy

First, we review a quick Lemma about entropy (nearly identical to Lemma 2.3 in [5]).

Lemma 3.3.1. *If $x \leq \frac{1}{2}$ then $H(x, 1-x) \geq 2x$.*

Proof. We refer the reader to Gallager's 1968 work [21]. □

Next, we describe a Lemma which breaks our choice of root in the greedy ME heuristic into one of three cases. Note that these are not the same as cases a), b) and c) in the ME rule description in 3.2.

Lemma 3.3.2. *When using the ME Rule to choose the root B_r of a binary search tree t with total probability p_t, one of the following three cases must occur:*
Case 1) $E_t \geq 1 - 2\frac{p_r}{p_t}$
Case 2) There exists gap (B_i, B_{i+1}) *such that* $\frac{q_i}{p_t} > max(P_{L_t}(B_i, B_{i+1}), P_{R_t}(B_i, B_{i+1}))$
Case 3) $\max(P_{L_t}(B_r), P_{R_t}(B_r)) < \frac{4}{5}$

Proof. First, we show when *Case 1* occurs. Then we show that if *Case 1* does not occur, and *Case 2* also does not occur, then *Case 3* must, completing the proof.

Case 1
First, suppose there exists some p_i such that $\frac{p_i}{p_t} > \max(P_{L_t}(B_i), P_{R_t}(B_i))$. By the ME

Rule case c), it must be selected as the root and thus $r = i$. Moreover, both $P_{L_t}(B_i)$ and $P_{R_t}(B_i))$ must be less than one half. Thus, using 3.3.1 we have:

$$\begin{aligned} E_t &\geq H(max(P_{L_t}(p_i), P_{R_t}(p_i)), 1 - max(P_{L_t}(p_i), P_{R_t}(p_i)) \\ &\geq 2 \cdot \max(P_{L_t}(p_i), P_{R_t}(p_i)) \\ &\geq 1 - \frac{p_i}{p_t} \\ &\geq 1 - 2\frac{p_i}{p_t} \geq 1 - 2\frac{p_r}{p_t} \end{aligned}$$

as required.

If we do not have some p_i such that $\frac{p_i}{p_t} > \max(P_{L_t}(B_i), P_{R_t}(B_i))$ but do have some B_i such that $P_{L_t}(B_i) \leq \frac{1}{2}$ and $P_{R_t}(B_i) \leq \frac{1}{2}$) then we have:

$$\begin{aligned} E_t &\geq H(P_{L_t}(B_i), \frac{B_i}{p_t}, P_{R_t}(B_i) \\ 0 &\leq P_{L_t}(B_i) \leq 0.5 \\ 0 &\leq \frac{p_i}{p_t} \leq 0.5 \\ 0 &\leq P_{R_t}(B_i) \leq 0.5 \end{aligned}$$

then we know that

$$H(P_{L_t}(p_i), \frac{p_i}{p_t}, P_{R_t}(p_i)) \geq H(1/2, 1/2) = 1$$

in this case. Thus, $E_t \geq 1 - 2p_r$ as required.

Case 3
Otherwise, we must have some gap (B_i, B_{i+1}) spanning the middle of the data set (i.e. $P_{L_t}(B_i, B_{i+1}) < \frac{1}{2}$ and $P_{R_t}(B_i, B_{i+1}) < \frac{1}{2}$). Suppose that *Case 2)* does not occur: there does not exist a (B_i, B_{i+1}) such that $\frac{q_i}{p_t} > max(P_{L_t}(B_i, B_{i+1}), P_{R_t}(B_i, B_{i+1}))$. Then, for any root r of t we have that

$$\begin{aligned} &\max(P_{L_t}(B_r), P_{R_t}(B_r)) \geq \min(P_{L_t}(B_r), P_{R_t}(B_r)) \\ &\max(P_{L_t}(B_r), P_{R_t}(B_r)) \geq q_i \end{aligned}$$

Thus,

$$\max(P_{L_t}(p_r), P_{R_t}(p_r)) \geq 1/3$$

and by our assumption

$$\max(P_{L_t}(p_r), P_{R_t}(p_r)) < \frac{1}{2}.$$

So, as in the proof of table 3 (5.3) in [26]

$$E_t \geq H(1/3, 2/3) \approx 0.92.$$

Since *Case 1* does not occur, we have that:

$$\begin{aligned} E_t &< 1 - 2\frac{p_r}{q_t} \\ \implies \frac{p_r}{q_t} &< \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2} \approx 0.04. \end{aligned}$$

Suppose for contradiction that $\max(P_{L_t}(p_r), P_{R_t}(p_r)) \geq \frac{4}{5}p_t$ then we have:

$$E_t \leq H(\frac{4}{5}, \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2}, \frac{1}{5} - \frac{1 - H(\frac{1}{3}, \frac{2}{3})}{2}) \approx 0.87 < 0.92 \approx H(\frac{1}{3}, \frac{2}{3}) \leq E_t$$

which is a contradiction. Thus, if we do not have *Case 1* or *Case 2* we must have *Case 3* which completes the proof.

□

Before we examine the main theorem we show a small claim.

Claim 1. *$H(\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x) \geq 1 - \frac{4}{5}(x)^2$ when $0 < x < \frac{1}{2}$*

Proof. In order to prove the claim, we find the minimum of

$$H(\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x) - (1 - \frac{4}{5}(x)^2)$$

when $0 < x < \frac{1}{2}$. To do this, we define $F(x)$ and take the derivative with respect to x

$$\begin{aligned} F(x) &= H(\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x) - (1 - \frac{4}{5}(x)^2) \\ F(x) &= -(\frac{1}{2} - \frac{1}{2}x) \cdot \lg(\frac{1}{2} - \frac{1}{2}x) - (\frac{1}{2} + \frac{1}{2}x) \cdot \lg(\frac{1}{2} + \frac{1}{2}x) - (1 - \frac{4}{5}(x)^2) \\ \implies F'(x) &= \lg(\frac{1}{2} - \frac{1}{2}x) - \lg(\frac{1}{2} + \frac{1}{2}x) + \frac{8}{5}x \end{aligned}$$

The only root occurs when $x = 0$. Thus, we check when $x \to 0$ and $x \to \frac{1}{2}$. We note that:
$F'(x) \xrightarrow{x \to 0} 0^+$ and
$F'(x) \xrightarrow{x \to \frac{1}{2}} 0.0112781 > 0$.
Thus, $H(\frac{1}{2} - \frac{1}{2}x, \frac{1}{2} + \frac{1}{2}x) - (1 - \frac{4}{5}(x)^2) > 0$ for $0 < x < \frac{1}{2}$ which proves the claim. □

Theorem 3.3.3. $C_{ME} \leq H + 4$

Proof. This uses a similar style to the proof of theorem 4.4 in [5]. We bind each E_t for each subtree of our BST on a case by case basis using the cases of Lemma 3.3.2.
If *Case 1* occurs, we obviously have that

$$E_t \geq 1 - 2\frac{p_r}{p_t} \tag{3.2}$$

Note that this can only happen once for each key (a key can only be root once).

While it wasn't explicitly stated, in Lemma 3.3.2 it was shown that if some B_i spans in the middle of the data set, $P_{L_t}(B_i) \leq \frac{1}{2}$ and $P_{R_t}(B_i) \leq \frac{1}{2}$, then regardless of whether or not $\frac{p_i}{p_t} > \max(P_{L_t}(B_i), P_{R_t}(B_i))$, we can still show that *Case 1* occurs. Suppose for the remainder of the proof that there is no such middle-spanning B_i.

Let (B_m, B_{m+1}) be the unique middle gap (i.e. $P_{L_t}(B_m, B_{m+1}) < \frac{1}{2}$ and $P_{R_t}(B_m, B_{m+1}) < \frac{1}{2}$) when *Case 2* or *Case 3* occurs. When *Case 2* occurs we have that (using Lemma 3.3.1):

$$E_t \geq H(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}, \frac{1}{2} + \frac{1}{2}\frac{q_m}{p_t}) \geq 2(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}) = 1 - \frac{q_m}{p_t}. \tag{3.3}$$

Note that by the definition of the ME Rule case c), when this occurs, (B_m, B_{m+1}) must be a leaf of depth at most 2. Thus, this condition can only happen twice for each (B_m, B_{m+1})

gap.

When *Case 2* does not occur (and we have a (B_m, B_{m+1}) spanning the middle) we must have *Case 3* (since we are still assuming that *Case 1* does not occur). This gives us:

$$E_t \geq H(\frac{1}{2} - \frac{1}{2}\frac{q_m}{p_t}, \frac{1}{2} + \frac{1}{2}\frac{q_m}{p_t})$$

We again apply Claim 1 and get:

$$E_t \geq 1 - \frac{4}{5}(\frac{q_m}{p_t})^2 \tag{3.4}$$

As in [5] we define a b_t for each subtree t as follows. We want to have a value for b_t such that $E_t \geq 1 - b_t$ in all cases. Using *Cases 1,2*, and *3* are their respective equations 3.2, 3.3, and 3.4 we do just that:
Let $b_t = 2 \cdot p_r$ when *Case 1* occurs. B_r is the root of b_t.
Let $b_t = 2 \cdot q_m$ when *Case 2* occurs. (B_m, B_{m+1}) is middle gap of b_t.
Let $b_t = \frac{q_m^2}{p_t}$ when *Case 3* occurs. (B_m, B_{m+1}) is middle gap of b_t.

Note that, in 1975 P. Bayer showed that the cost C of our tree could be defined as (**Lemma 2.3** [5]):

$$C = \sum_{t \in S_T} p_t$$

and the entropy could be calculated by

$$H = \sum_{t \in S_T} p_t \cdot E_t$$

where S_T is the set of all subtrees of our tree T.

Thus, by subbing in $E_t \geq 1 - b_t$ and rearranging we get:

$$H = \sum_{t \in S_T} p_t E_t \geq \sum_{t \in S_T} P_t - \sum_{t \in S_T} b_t = C - \sum_{t \in S_T} b_t$$
$$\implies C \leq H + \sum_{t \in S_T} b_t$$

As mentioned above, *Case 1* and *Case 2* can only occur once and twice respectively for any potential root B_r or gap (B_m, B_{m+1}). *Case 3* however, can occur many times for a gap (B_m, B_{m+1}). Each time it occurs though, $\frac{q_m}{p_t}$ must decrease by a factor of $\frac{5}{4}$ since $\max(P_{L_t}(B_r), P_{R_t}(p_r)) < \frac{4}{5}$ for the root B_r of the subtree by *Case 3)* of Lemma 3.3.2. Moreover, if $\frac{q_m}{p_t} > \frac{1}{2}$ then we will have *Case 2*. Let S_m be the set of all subtrees t for which (B_m, B_{m+1}) is the middle gap and *Case 3* only applies. We have that

$$C \leq H + \sum_{t \in S_T} b_t = H + 2\sum_{r=1}^{n} p_r + 2\sum_{m=0}^{n} q_m + \sum_{m=0}^{n} \sum_{t \in S_m} \frac{4}{5} \frac{q_m^2}{p_t}$$

By factoring out q_m and examining only cases up to $\frac{q_m}{p_t} = \frac{1}{2}$ (since otherwise *Case 2* will occur) we get:

$$C \leq H + 2 + \sum_{m=0}^{n} (\frac{4}{5} \cdot q_m) \sum_{x=0}^{\infty} \frac{1}{2} \cdot (\frac{4}{5})^x$$

$$C \leq H + 2 + \sum_{m=0}^{n} \frac{4}{5} \cdot q_m \cdot \frac{1}{2} \cdot (\frac{1}{1 - \frac{4}{5}}) \text{ (geometric series)}$$

$$C \leq H + 2 + \sum_{m=0}^{n} q_m$$

$$C \leq H + 4$$

□

Chapter 4

Approximate Binary Search in the Hierarchical Memory Model

In this Chapter We examine the optimum BST problem under the HMM model. We provide two approximate solutions, bound their worst case expected costs via entropy, and show improvement over a previous solution in the related HMM_2 model.

4.1 The Hierarchical Memory Model

The Hierarchical Memory Model (HMM) was proposed in 1987 by Aggarwal et al. as an alternative to the classic RAM model [2]. It was intended to better model the multiple levels of the memory hierarchy. The model has an unlimited number of registers, $R_1, R_2, ...$ each with its own location in memory (a positive integer). In the first version of the model, accessing a register at memory location x_i takes $\lceil \lg(x_i) \rceil$ time. Thus, computing $f(a_1, a_2, ..., a_n)$ takes $\sum_{i=1}^{n} \lceil \lg(location(a_i)) \rceil$ time. The original paper also considered arbitrary cost functions $f(x)$. We will use the cost function as was explained in Thite's thesis [45]. Here, $\mu(a)$ is the cost of accessing memory location a, an integer. We have a set of memory sizes $m_1, m_2, ..., m_l$ which are monotonically increasing. Each memory level has a finite size except m_l which we assume has infinite size. Each memory level has an associated cost of access $c_1, c_2, ..., c_l$. We assume that $c_1 < c_2 < ... < c_l$. The cost of accessing a memory location a is given by

$$\mu(a) = c_i \text{ if } \sum_{j=1}^{i-1} m_j < a \leq \sum_{j=1}^{i} m_j. \tag{4.1}$$

Thite notes that typical memory hierarchies have decreasing sizes for faster memory levels (moving *up* the memory hierarchy). We make the same assumption:

$$m_1 < m_2 < ... < m_l.$$

Unlike Thite, we also explicitly assume that successive memory level sizes divide one another evenly:

$$\forall i \in \{1, 2, ..., l-1\} m_i \mid m_{i+1}.$$

4.2 Thite's Optimum Binary Search Trees on the HMM Model

Thite's thesis provides solutions to several problems in the HMM and the related HMM_2 models [45]. He first provides an optimal solution to the following problem (known as **Problem 5** in the work):

Problem 5 [Optimum BST Under HMM]. [45] Suppose we are given a set of n ordered keys $x_1, x_2, ..., x_n$ with associated probabilities of search $p_1, p_2, ..., p_n$, as well as $n+1$ ranges $(-\infty, x_1), (x_1, x_2), ..., (x_{n-1}, x_n), (x_n, \infty)$ with associated probabilities of search $q_0, q_1, ..., q_n$. The problem is to construct a binary search tree T over the set of keys and compute a memory assignment function $\phi : V(T) \rightarrow 1, 2, ..., n$ that assigns nodes of T to memory locations such that the expected cost of a search is minimized under the HMM model.

In order to remain consistent with the work in other chapters of this thesis, We will maintain the use of $B_1, B_2, ..., B_n$ to represent keys instead of Thite's $x_1, x_2, ..., x_n$ and ranges will be represented by $(B_0, B_1), (B_1, B_2), ..., (B_n, B_{n+1})$ where it is assumed that B_0 represents $-\infty$ and B_{n+1} represents ∞.

Thite provided three separate optimum solutions to the problem described; **Parts**, **Trunks**, and **Split**. These algorithms have various use cases and running of times of $O(\frac{2^{h-1}}{(h-1)!} \cdot n^{2h+1})$, $O(\frac{2^{n-m_h} \cdot (n-m_h+h)^{n-m_h} \cdot n^3}{(h-2)!})$ and $O(2^n)$ respectively. Here, h is the minimum memory level such that all n keys can fit on memories of height at most h. More specifically h is defined as

$$\min(h \in \{1, .., l\}) : n \leq \sum_{i=1}^{h} m_i$$

In the following sections, we provide two approximate solutions to this problem that run in times $O(n \cdot \lg(m_1))$ and $O(n)$ and provide an upper bound on their expected search costs.

Thite also considered the same problem under the related HMM_2 model. This model assumes there are simply two levels of memory of size m_1 and m_2 with costs of access c_1 and c_2 where $c_1 < c_2$. Thite provides an optimal solution to this problem (named **TwoLevel**) which runs in time $O(n^5)$. It runs in $o(n^5)$, if $m_1 \in o(n)$, and in $O(n^4)$ if $m_1 \in O(1)$. He also gives an $O(n \lg n)$ time approximate solution with an upper bounded expected search cost of $c_2(H + 1)$. The solution we provide under the HMM model also gives an improvement over Thite's approximate algorithm in both running time and expected cost under the HMM_2 model.

4.3 Efficient Near-Optimal Multiway Trees of Bose and Douïeb

In order to approximately solve the optimum BST problem under the HMM model, we use the multiway search tree construction algorithm of Bose and Douïeb. In 2009, the duo devised a new method with linear running time (independent of the size of a node in tree) and with the best expected cost to date [13]. The group was able to prove that:

$$\frac{H}{\lg(2k-1)} \leq P_{OPT} \leq P_T \leq \frac{H}{\lg k} + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}.$$

Here, H is the entropy of the probability distribution, P_{OPT} is the average path-length in the optimal tree, P_T is the average path length of the tree built using their algorithm and $m = max(n - 3P, P) - 1 \geq \frac{n}{4} - 1$. P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves. Moreover, $q_{\text{rank}[i]}$ is the i^{th} smallest access probability among all leaves (gaps) except q_0 and q_n. Finally, k is the size of a node in the tree.

As described in the section 2.3, in the multiway search tree problem we are given n ordered keys with weights $p_0, ..., p_n$ as well as $n+1$ weights of unsuccessful searches $q_0, ..., q_n$. We often refer to "keys" representing the gaps as (B_i, B_{i+1}). Specifically, we mean the "key" associated with a search between keys B_{i-1} and B_i. The goal is to build a minimum

cost tree where k keys fit inside a single node.

The algorithm occurs in three steps. First, a new distribution is created by distributing all leaf weights to appropriate internal nodes. We refer readers to the paper [13] for a more detailed explanation, but we give an overview here. First, an examination of the peaks and valleys of the probability distribution of the leaf weights is used to do this weight reassignment. After this, the new probability distribution is made into a k-ary tree using a greedy recursive algorithm. The algorithm recursively chooses the $l \leq k$ elements to go in the root node such that each child's subtree will have probability of access of at most $\frac{1}{k}$. After this *internal key tree* is completed as described, leaf nodes are reattached to the internal key tree. Their algorithm's design allows the authors to bound the depth of keys and leaves by their associated probabilities. This ultimately allows the duo to achieve the bounds they have described.

4.4 Algorithm ApproxMWPaging

In this section We provide an algorithm for creating a BST and subsequently packing it into memory. The algorithm first uses the multiway search tree construction algorithm of Bose and Douïeb as a subroutine to be a mutliway search tree. This multiway search tree is converted into a BST, then packed into the memory hierarchy.

First, we describe how we convert a multiway search tree to a binary search tree. For the sake of clarity, we will call what are typically known as *nodes* of the multiway tree *pages*. This represents how various items of our search tree will fit onto pages of our memory hierarchy. We maintain the notion of calling individual items *keys*.

Lemma 4.4.1. *Given a multiway tree T' with page size k and n keys, where keys are associated with a probability distribution of successful and unsuccessful searches as in Knuth's original optimum binary search tree problem, we can create a BST T where each key in a given page $g \in T'$ forms a connected component in T in $O(n \lg(k))$ time.*

Proof. For each page g, we sort its keys by lexicographic order and create a complete BST B over the keys. We create an ordering over all potential locations where keys could be added to the tree from left to right. All keys in all descendant pages of a page g in a specific subtree rooted at a child of g will lie in a specific range. There are at most $k + 1$ of these ranges (since our page has at most k keys). These ranges precisely correspond to

the at most $k+1$ locations where a new child key can to B. We note that we cannot add new children to keys which correspond to unsuccessful searches. We order these locations from left to right and attach root keys from the newly created BST's of each of the ordered (left to right) children of g. These are all valid connections since each child of g has keys in these correct ranges, and combining BST's in this fashion produces a valid BST. We perform a sort of $O(k)$ items (in time $O(k \lg(k)))$ $O(\frac{n}{k})$ times, and make $O(n)$ new parent child connections, giving us total time $O(n \lg(k))$. □

In order to approximately solve the optimum BST problem under the HMM model, we do the following:

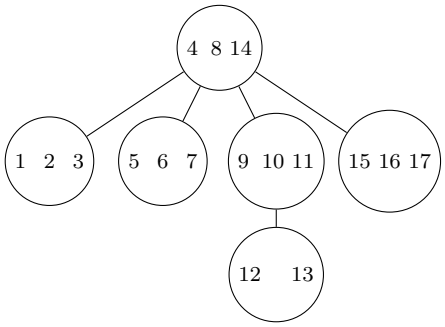
1. First, we create a multiway tree T' using the algorithm of Bose and Douïeb. This takes $O(n)$ time with our node size equal to m_1 (the smallest level of our memory hierarchy) [13].

2. Inside each page (node of the multiway tree T'), we create a balanced binary search tree (ignoring weights). We use a simple greedy approach where we sort the keys, then recursively select the middle key as the root. We call each of these T'_k for $k \in 1, ..., \lceil n/m_1 \rceil$. This takes $O(n \cdot \lg(m_1))$ by Lemma 4.4.1.

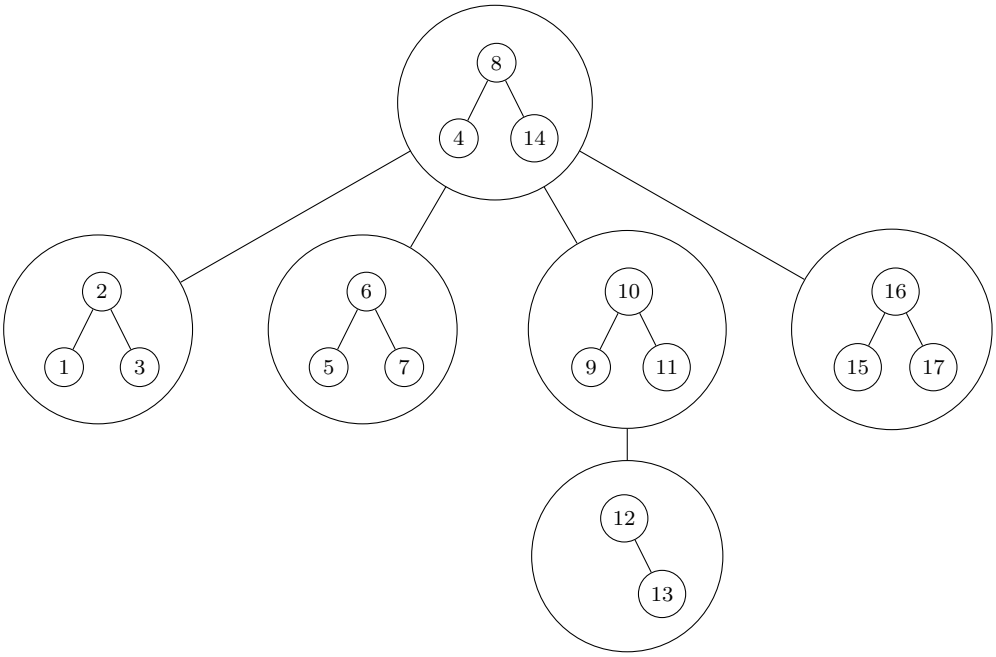
3. In order to make this into a proper binary search tree, we must connect the $O(n/m_1)$ BST's we have made as described in Lemma 4.4.1. From T', we create a BST T. This takes $O(n \cdot \lg(m_1))$ time.

4. We pack keys into memory in a breadth first search order of T starting from the root. This takes $O(n)$ time.

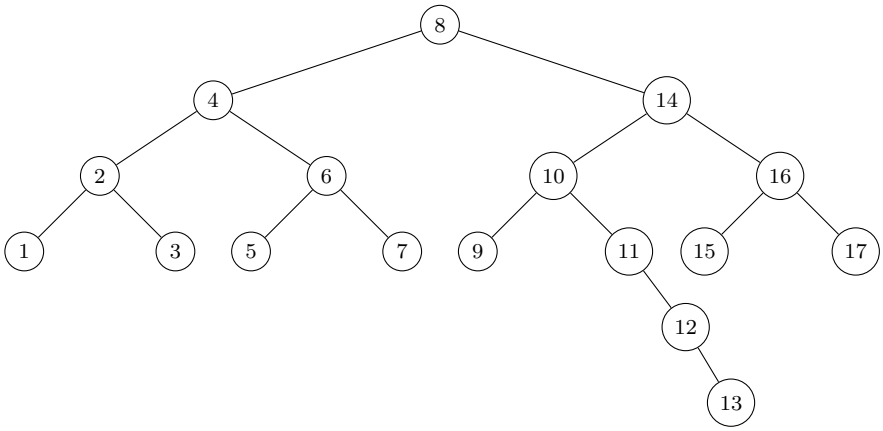
We are left with a binary search tree which has been properly packed into our memory in total time $O(n \cdot \lg(m_1))$.



(a) Part 1. of ApproxMWPaging



(b) Part 2. of ApproxMWPaging



(c) Part 3. of ApproxMWPaging

Memory Location	Node	Left Child Location	Right Child Location
1	8	2	3
2	4	4	5
3	14	6	7
4	2	8	9
5	6	10	11
6	10	12	13
7	16	14	15
8	1	—	—
9	3	—	—
10	5	—	—
11	7	—	—
12	9	—	—
13	11	—	16
14	15	—	—
15	17	—	—
16	12	—	17
17	13	—	—

(d) Part 4. of ApproxMWPaging

Figure 4.1: An example of the ApproxMWPaging algorithm.

4.5 Expected Cost ApproxMWPaging

First, we bound the depth of nodes in our BST T. The depth of a key B_i (denoted (B_{i-1}, B_i) for unsuccessful searches) is defined as $d_T(B_i)$ (resp. $d_T(B_{i-1}, B_i)$). Note that depth is the number of edges between a node and the root (i.e. the depth of the root is

0). As in the work of Bose and Douïeb, let $m = \max(n - 3P, P) - 1 \geq \frac{n}{4} - 1$ where P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves [13].

Lemma 4.5.1. *For a key B_i,*

$$d_T(B_i) \leq \lg(\frac{1}{p_i}).$$

For a key (B_{i-1}, B_i),

$$d_T(B_{i-1}, B_i) \leq \lg(\frac{1}{q_i}) + 2$$

for all leaves, and

$$d_T(B_{i-1}, B_i) \leq \lg(\frac{1}{q_i}) + 1$$

for at least m of them (and the two extremal leaves, (B_0, B_1) and (B_n, B_{n+1})).

Proof. First we note that in the tree T' we build using Bose and Douïeb's multiway tree algorithm, the maximum depth of keys (call this $d_{T'}(B_i), d_{T'}(B_{i-1}, B_i)$) for a page size m_1 is [13]:

$$\begin{aligned} d_{T'}(B_i) &\leq \lfloor log_{m_1}(\frac{1}{p_i}) \rfloor \\ d_{T'}(B_{i-1}, B_i) &\leq \lfloor log_{m_1}(\frac{2}{q_i}) \rfloor + 1 \text{ for all leaves, and} \\ d_{T'}(B_{i-1}, B_i) &\leq \lfloor log_{m_1}(\frac{1}{q_i}) \rfloor + 1 \text{ for at least } m \text{ of them (and the two extremal leaves)} \end{aligned}$$

As explained in the paper, these follow from Lemmas 1 and 2 of Bose and Douïeb [13].

Inside a page, we make a balanced (ignoring weight) BST, so each key has a depth within a page of at most $\lfloor \lg(m_1) \rfloor$. Since our algorithm always connects the root of the BST made for a page to a key in the BST made for the page's parent, a key B_i has a *page depth* (the number of unique pages accessed in order to access the key) of at most the bounds on $d_{T'}(B_i)$ and $d_{T'}(B_{i-1}, B_i)$ described. Since we examine at most $\lceil \lg(m_1) \rceil$ keys

within any one page, (and only 1 key in a leaf page) a key's depth is at most

$$\begin{aligned} & d_T(B_i) \leq \lfloor \lg(m_1) \rfloor \lfloor log_{m_1}(\frac{1}{p_i}) \rfloor \\ \implies & d_T(B_i) \leq \lg(m_1) \cdot log_{m_1}(\frac{1}{p_i}) \\ \implies & d_T(B_i) \leq \lg(\frac{1}{p_i}). \end{aligned}$$

For an unsuccessful search

$$\begin{aligned} & d_T(B_{i-1}, B_i) \leq \lfloor \lg(m_1) \rfloor \lfloor log_{m_1}(\frac{2}{q_i}) \rfloor + 1 \\ \implies & d_T(B_{i-1}, B_i) \leq \lg(m_1) \cdot log_{m_1}(\frac{2}{q_i}) + 1 \\ \implies & d_T(B_{i-1}, B_i) \leq \lg(\frac{1}{q_i}) + 2 \text{ for all leaves, and} \\ \implies & d_T(B_{i-1}, B_i) \leq \lg(\frac{1}{q_i}) + 1 \text{ for at least } m \text{ of them (and the two extremal leaves).} \end{aligned}$$

□

We now describe W, the cost of searching for a key located at the deepest node of tree T. Let $m'_j = \sum_{k \leq j} m_j$. We define $m'_0 = 0$. As defined in Thite's work, we let h be the smallest j such that $m'_j \geq n$. Let $D(T)$ be the height of T (the depth of the deepest node in the tree).

Lemma 4.5.2.

$$W \leq \sum_{i=1}^{h-1} \left(\lfloor \lg(m'_i + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1 \rfloor \right) \cdot c_i + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

Proof. Consider the accessing each key along the path from the root to the deepest key in T. We will examine $D(T)$ keys. We access one key at depth 0, one key at depth 1, and so on. Because the tree is packed into memory in BFS order, a key a depth i will be at memory index at most $2^i - 1$. Now, consider how many levels of the binary search tree T will fit in m_1. In order for all keys of depth i (and higher) to be in m_1 we need:

$$2^i - 1 \leq m_1 \implies i \leq \lg(m_1 + 1).$$

Thus, at least $\lfloor \lg(m_1 + 1) \rfloor$ levels of T will completely fit on m_1. Next we examine how many levels of T fit on m_j for $1 < j < l$. The last level to completely fit on m_j or higher memories is the maximum i such that:

$$2^{i+1} - 1 \leq m'_j \implies i \leq \lg(m'_j + 1).$$

Thus, at least $\lfloor \lg(m'_j + 1) \rfloor$ levels of T fit on m_j or higher levels of memory. On our search for the deepest key in the tree, we make at least $\lfloor \lg(m_1 + 1) \rfloor$ checks for elements located at memory m_1. This costs a total of

$$\lfloor \lg(m_1 + 1) \rfloor \cdot c_1.$$

For each memory level m_j for $1 < j < l$, we make at least $\lfloor \lg(m'_j + 1) \rfloor$ checks in m_j or higher memories. Of these checks, at least $\lfloor \lg(m'_{j-1} + 1) \rfloor$ are in memory levels strictly higher up in the memory hierarchy than j. Since $c_j > c_i$ for $j > i$, an upper bound on the cost of searching for all elements in memory level j on the path from the root to the deepest element of the tree is:

$$(\lfloor \lg(m'_j + 1) \rfloor - \lfloor \lg(m'_{j-1} + 1) \rfloor) \cdot c_j.$$

Finally, we can upper bound the cost of search within m_h by assuming that all remaining searches take place at memory level h. The searches at level h will cost at most:

$$(\lg(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor) \cdot c_h.$$

Combining the above three equations (and summing over every memory level) gives us the total cost of searching for a key in the deepest part of the tree:

$$W \leq \lfloor \lg(m_1 + 1) \rfloor \cdot c_1 + \sum_{i=2}^{h-1} \left(\lfloor \lg(m'_i + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1) \rfloor \right) \cdot c_i + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

$$W \leq \sum_{i=1}^{h-1} \left(\lfloor \lg(m'_i + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1) \rfloor \right) \cdot c_i + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

□

This leads us to the following upper bound for W.

Lemma 4.5.3. *Assuming that $l \neq 1$:*

$$W < D(T) \cdot c_h$$

Proof. We can rearrange Lemma 4.5.2 as follows:

$$W \leq D(T) \cdot c_h - \sum_{i=1}^{h-1} \lfloor \lg(m'_i + 1) \rfloor \cdot (c_{i+1} - c_i)$$

Since we have that $c_i > c_{i-1}$ for all i, $\sum_{i=1}^{h-1} \lfloor \lg(m'_i + 1) \rfloor \cdot (c_{i+1} - c_i)$ is strictly positive. This instantly gives the result desired. □

Note that $\frac{W}{D(T)}$ represents the average cost per memory access when accessing the deepest (and most costly) element of our tree. Since our costs of access are monotonically increasing as we move deeper in the tree, we will see that $\frac{W}{D(T)}$ can be used as an upper bound for the average cost per memory access when searching for any element of T.

Lemma 4.5.4. *The cost of searching for a keys B_i and (B_{i-1}, B_i) ($C(B_i)$ and $C(B_{i-1}, B_i)$ respectively) can be bounded as follows:*

$$C(B_i) \leq (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{p_i}) + 1) \cdot c_h$$

$$C(B_{i-1}, B_i) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{q_i}) + 2) \cdot c_h \text{ for all leaves, and}$$

$$C(B_{i-1}, B_i) \leq (\lg(\frac{1}{q_i}) + 1) \cdot \frac{W}{D(T)} < (\lg(\frac{1}{q_i}) + 1) \cdot c_h \text{ for at least } m \text{ (and the two extremal) leaves.}$$

Proof. From Lemma 4.5.1 we have a bound on the depth of keys B_i and (B_{i-1}, B_i). We must do $d_T(B_i) + 1$ accesses to find a key and $d_T(B_{i-1}, B_i)$ accesses to find a leaf. Note that since our tree is stored in BFS order in memory, whenever we examine a key's child, it will be at a memory location of at least the same, if not higher cost (by being in the same or a deeper page). Thus, the cost of accessing the entire path from the root to a specific key can be upper bounded by multiplying the length of this path by the average cost per memory access of the most expensive (and deepest) key of the tree (this is exactly $\frac{W}{D(T)}$). Note that when searching for keys, we must search along the entire path from root to the key in question, while we need only examine the path from the root to the parent of a key

for unsuccessful (B_{i-1}, B_i) searches. Combining 4.5.1, 4.5.2 and 4.5.3 gives

$$\begin{aligned} & C(B_i) \leq (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} \\ \implies & C(B_i) < (\lg(\frac{1}{p_i}) + 1) \cdot \frac{D(T) \cdot c_h}{D(T)} \\ \implies & C(B_i) < (\lg(\frac{1}{p_i}) + 1) \cdot c_h \end{aligned}$$

For leaves we have that

$$\begin{aligned} & C(B_{i-1}, B_i) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{W}{D(T)} \\ \implies & C(B_{i-1}, B_i) \leq (\lg(\frac{1}{q_i}) + 2) \cdot \frac{D(T) \cdot c_h}{D(T)} \\ \implies & C(B_{i-1}, B_i) < (\lg(\frac{1}{q_i}) + 2) \cdot c_h \text{ for all leaves, and} \\ \implies & C(B_{i-1}, B_i) < (\lg(\frac{1}{q_i}) + 1) \cdot c_h \text{ for at least } m \text{ (and the two extremal) leaves.} \end{aligned}$$

□

We can now bound the expected cost of search using the bounds for the cost each key.

Theorem 4.5.5.

$$C \leq \frac{W}{D(T)} \cdot \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} pq_{rank[i]} \right) \text{ and}$$

$$C < \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} pq_{rank[i]} \right) \cdot c_h$$

where $pq_{rank[i]}$ is the i^{th} smallest access probability among all keys and leaves except q_0 and q_n.

Proof. The total expected cost of search is simply the sum of the weighted cost of search for all keys multiplied by the probability of searching for each key. Given our last lemma,

we have that:

$$C \leq \sum_{i=1}^{n} p_i \cdot C(B_i) + \sum_{j=1}^{n+1} q_j \cdot C(B_{i-1}, B_i)$$

$$\implies C \leq \sum_{i=1}^{n} p_i \cdot (\lg(\frac{1}{p_i}) + 1) \cdot \frac{W}{D(T)} + \left(\sum_{i=0}^{n} q_i(\lg(\frac{1}{q_i}) + 2) - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right) \cdot \frac{W}{D(T)}$$

$$\implies C \leq \frac{W}{D(T)} \left(\sum_{i=1}^{n} p_i \lg(\frac{1}{p_i}) + \sum_{i=0}^{n} q_i \lg(\frac{1}{q_i}) + \sum_{i=1}^{n} p_i + 2\sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right)$$

$$\implies C \leq \frac{W}{D(T)} \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right)$$

By Lemma 4.5.3 this gives:

$$C < \left(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]} \right) \cdot c_h$$

□

4.6 Approximate Binary Search Trees of De Prisco and De Santis with Extensions by Bose and Douïeb

We provide another approach to building the approximately optimal BST under the HMM model. This approach uses the approximate BST solution (in the simple RAM model) of De Prisco and De Santos (modified by Bose and Douïeb) [16, 13]. we explain the method here.

As in the classic Knuth problem, we are given a set of n probabilities of searching for keys $(p_1, p_2, ..., p_n)$, as well as $n+1$ probabilities of unsuccessful searches $(q_0, q_1, ..., q_n)$. De Prisco and De Santos give an algorithm which constructs a binary search tree in $O(n)$ time with an expected cost of at most [16]

$$H + 1 - q_0 - q_n + q_{max}$$

where q_{max} is the maximum probability of an unsuccessful search. This was later modified by Bose and Douïeb (the same paper described in section 4.3) to have an improved bound

of [13]

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}.$$

Here, P is the number of increasing or decreasing sequences in a left-to-right read of the access probabilities of the leaves and $m' = \max(2n - 3P, P) - 1 \geq \frac{n}{2} - 1$. Moreover, $pq_{\text{rank}[i]}$ is the i^{th} smallest access probability among all keys and leaves except q_0 and q_n.

First, we explain the algorithm of De Prisco and De Santis and then explain the extensions of Bose and Douïeb. De Prisco and De Santis' algorithm occurs in three phases.

Phase 1 An auxiliary probability distribution is created using $2n$ zero probabilities, along with the $2n + 1$ successful and unsuccessful search probabilities. Yeung's linear time alphabetic search tree algorithm is used with the $2n + 1$ successful and unsuccessful search probabilities used as leaves of the new tree created [51]. This is referred to as the *starting tree*.

Phase 2 What's known as the *redundant tree* is created by moving p_i keys up the *starting tree* to the lowest common ancestor of keys q_{i-1} and q_i. The keys which used to be called p_i are relabelled to $old.p_i$.

Phase 3 The *derived tree* is constructed from the *redundant tree* by removing redundant edges. Edges to and from nodes which represented zero probability keys are deleted. This *derived tree* is a binary search tree with the expected search cost described.

In Bose and Douïeb's work, they explain how they can substitute their algorithm for Yeung's linear time alphabetic search tree algorithm which results in a better bound (as described above). We use the updated version (by Bose and Douïeb) of De Prisco and De Santis' algorithm as a subroutine in the sections to follow.

4.7 Algorithm ApproxBSTPaging

Our second solution to create an approximately optimal BST under the HMM model works as follows:

1. First, we create a BST T using the algorithm of De Prisco and De Santis [16] (as updated by Bose and Douïeb [13]). This takes $O(n)$ time.

2. In a similar fashion to step 4) of *ApproxMWPaging*, we pack keys from T into memory in a breadth-first search order starting from the root. This relatively simple traversal also takes $O(n)$ time.

We are left with a binary search tree which is properly packed into memory in total time $O(n)$.

4.8 Expected Cost ApproxBSTPaging

As explained in the Bose and Douïeb paper, the average path length search cost of the tree created by their algorithm is at most: [13]

$$H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} p q_{\text{rank}[i]}$$

We call this value P_T (the average search cost of tree T). Similar to the proof in section 4.5, if we can bound the cost of search for a given path length, then we can form a bound on the average cost of search in the HMM model. As before, we can describe the cost of searching for a key located at deepest node of tree T: W. Recall, $m'_j = \sum_{k \leq j} m_k$, $m'_0 = 0$ and let h be the smallest j such that $m'_j \geq n$.

Lemma 4.8.1. *When using the ApproxBSTPaging, the cost of searching for a node deepest in the tree is at most:*

$$W \leq \sum_{k=1}^{h-1} \left(\lfloor \lg(m'_i + 1) \rfloor - \lfloor \lg(m'_{i-1} + 1) \rfloor \right) \cdot c_i + \left(D(T) - \lfloor \lg(m'_{h-1} + 1) \rfloor \right) \cdot c_h$$

Proof. Since we are simply putting keys into memory in BFS order, and all we use is the height of the tree and the memory hierarchy, the proof is identical to that of Lemma 4.5.2. □

Since we have the same result as Lemma 4.5.2, this immediately implies Lemma 4.5.3 is true as well. Assuming that $l \neq 1$, we have that $W < D(T) \cdot c_h$. As in the proof of the expected cost ApproxMWPaging, $\frac{W}{D(T)}$ represents the average cost per memory access when accessing the deepest (and most costly) element of our tree. Thus, $\frac{W}{D(T)}$ upper bounds the average cost per memory access when searching for any element of T.

Theorem 4.8.2.

$$C \leq (\frac{W}{D(T)}) \cdot (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \text{ and}$$

$$C < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \cdot c_h$$

Proof. Bose and Douïeb show that after using their algorithm for Phase 1 of De Prisco and De Santis algorithm algorithm, every leaf of the *starting tree* (all keys representing successful and unsuccessful searches) are at depth at most $\lfloor \lg(\frac{1}{p}) \rfloor + 1$ for at least $\max(2n - 3P, P) - 1$ of $p \in (\{p_1, p_2, ..., p_n\} \cup \{q_0, q_1, ..., q_n\})$ and $\lfloor \lg(\frac{1}{p}) \rfloor + 2$ for all others. Recall that P is the number of peaks in the probability distribution $q_0, p_1, q_1, ..., p_n, q_n$. After phases 2 and 3 of the algorithm, each key has its depth decrease by 2, and all leaves (except one) move up the tree by one.

$$C \leq \sum_{i=1}^{n} \left(p_i \cdot \frac{depth(p_i) + 1}{D(T)} \cdot W \right) + \sum_{i=0}^{n} \left(q_i \cdot \frac{(depth(q_i))}{D(T)} \cdot W \right)$$

$$\implies C \leq \frac{W}{D(T)} \left(\sum_{i=1}^{n} (p_i \cdot (depth(p_i) + 1)) + \sum_{i=0}^{n} (q_i \cdot (depth(q_i))) \right)$$

$$\implies C \leq \frac{W}{D(T)} (\text{WeightedAveragePathLength}(T))$$

$$\implies C \leq \left(\frac{W}{D(T)} \right) \cdot \left(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]} \right) \text{ and by Lemma 4.5.3}$$

$$\implies C < \left(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]} \right) \cdot c_h$$

□

4.9 Improvements over Thite in the HMM_2 Model

The HMM_2 model is the same as the general HMM model with the added constraint that there are only two types of memory (slow and fast). In Thite's thesis, he proposed both an optimal solution to the problem, as well as an approximate solution (*Algorithm Approx-BST*) that runs in time $O(n \lg(n))$ [45]. we first show that:

Lemma 4.9.1. *The proof of quality of approximation of Thite's approximate algorithm has a small mistake which raises its cost from at most:*

$$c_2(H+1) \text{ to}$$
$$c_2(H+1+\sum_{i=0}^{n} q_i)$$

Proof. Specifically, in Lemma 14 in 3.4.2.3 Quality of approximation in Thite's thesis, he proves that $\delta(z_k) = l + 2$. Here, $\delta(z_k)$ represents the depth of a leaf node z_k. Note that Thite considers the depth of the root to be 1 instead of 0 which updates how the cost of search is calculated accordingly. l represents the depth of recursion of the *Approx-BST* algorithm. In Lemma 15, Thite goes on to prove that $q_k \leq 2^{-\delta(z_k)+2}$. In this proof, Thite shows that $q_k \leq 2^{-l+1}$, but makes a mistake when substituting in $l = \delta(z_k) - 2$ and gets $q_k \leq 2^{-\delta(z_k)+2}$ while the correct bound is $q_k \leq 2^{-\delta(z_k)+3}$. This updated bound would change his depth bound in Lemma 16 from $\delta(z_k) \leq \lfloor \lg(\frac{1}{q_k}) \rfloor + 2$ to $\delta(z_k) \leq \lfloor \lg(\frac{1}{q_k}) \rfloor + 3$. Finally, substituting into his final equation for the upper bound on the expected cost of search for the tree would give:

$$\begin{aligned}
&\sum_{i=1}^{n} \left(c_2 p_i \delta(B_i) + \sum_{i=0}^{n} c_2 q_i (\delta(q_i) - 1 \right) \\
\leq &\sum_{i=1}^{n} \left(c_2 p_i (\lg(\frac{1}{p_i}) + 1) + \sum_{i=0}^{n} c_2 q_i (\frac{1}{q_i} - 1 + 3 \right) \\
\leq &c_2 \cdot \left(H + 1 + \sum_{i=0}^{n} q_i \right)
\end{aligned}$$

□

This is of particular interest because if Thite's bound on *Algorithm Approx-BST* had been correct, then in the case where $c_2 = c_1$ (typical RAM model), Thite's method would

have provided a strict improvement over the work of Bose and Douïeb [13] which seems unlikely since Thite used the BST approximation algorithm of Mehlhorn from 1984 [40] (much before the work of Bose and Douïeb).

By simply substituting in for $l = 2$ we immediately get that, under this HMM_2 model, both ApproxMWPaging and ApproxBSTPaging provide strict improvements over Thite's *Algorithm Approx-BST*.

Theorem 4.9.2. *In the HMM_2 model, ApproxMWPaging has an expected cost of at most*

$$C_{ApproxMWPaging} < (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{rank[i]}) \cdot c_2$$

and ApproxBSTPaging has an expected cost of at most

$$C_{ApproxBSTPaging} < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \cdot c_2.$$

ApproxMWPaging does so in $O(n \cdot \lg(m_1))$ while ApproxBSTPaging runs in $O(n)$.

Proof. We can directly sub $l = 2$ into Theorems 4.5.5 and 4.8.2 to get the desired result (the running times are as explained in sections 4.4 and 4.7). □

Since both ApproxMWPaging and ApproxBSTPaging run in time $o(n \lg(n))$ (the time of Thite's *Approx-BST* algorithm) and we can see that:

$$\begin{aligned} C_{ApproxMWPaging} &< (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}) \cdot c_2 \\ &< c_2(H + 1 + \sum_{i=0}^{n} q_i) \\ C_{ApproxBSTPaging} &< (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}) \cdot c_2 \\ &< c_2(H + 1 + \sum_{i=0}^{n} q_i) \end{aligned}$$

both methods provide a strictly better approximation and run faster than the *Algorithm Approx-BST* of Thite.

Chapter 5

BST over Multisets

In this chapter we examine a problem related to the initial optimal binary search tree problem of Knuth [36]. Our keys will no longer have a strict ordering and can be rearranged as we please before constructing our binary search tree.

5.1 The Multiset Binary Search Tree Problem

Consider a multiset (a set with possible duplicate values) of n probabilities: $M = \wr p_1, p_2, ..., p_n \int$ such that $\sum_{i=1}^{n} p_i = 1$. Our goal is to create a tree T which minimizes the expected path length, P_T, of nodes (the expected cost of our search in comparisons):

$$P_T = \sum_{i=1}^{n} p_i(b_i + 1) - \sum_{i \in L} p_i \tag{5.1}$$

Here, b_i is the *depth* of the i'th key of M and L is the set of leaves of the tree. We subtract the weight of the leaves of the tree since we need one less comparison to return a pointer a leaf node (as in the original optimal BST problem). In order to simplify our proof later we define f_T (the working *depth* of a node in tree T) as follows:

$$f_T(p_i) = \begin{cases} b_i + 1, & \text{if the } i\text{'th key is an internal node} \\ b_i, & \text{otherwise.} \end{cases}$$

Our expected path length can then be re-written as:

$$P_T = \sum_{i=1}^{n} p_i \cdot f_T(p_i).$$

5.2 The OPT-MSBST Algorithm

We propose the following simple algorithm entitled *OPT-MSBST* for solving this multiset binary search tree problem and subsequently show that it is optimal.

1. First, we create a vector R which is equal to the sorted (from largest to smallest) multiset M.
2. We create a BST T as follows. The root of our tree will be $R[0]$, its two children will be $R[1]$ and $R[2]$, and so on. Formally, $R[i]$ will be placed at location $i+1$ in the BFS order of T.

The complete proof of the optimality of this algorithm follows a similar form to the proof that Huffman codes are optimal [33]. First, we introduce a useful lemma.

Lemma 5.2.1. *Let T be a BST created for a multiset of probabilities as described in the problem definition in 5.1. Let T' be the tree created by swapping the node locations of p_j and p_k in T. Then,*

$$P_{T'} - P_T = (p_j - p_k) \cdot (f_T(p_k) - f_T(p_j)).$$

Proof. We let $f_{T'}$ represent the working height of a node in T'.

$$\begin{aligned}
P_{T'} - P_T &= \sum_{i=1}^{n} p_i \cdot f_{T'}(p_i) - \sum_{i=1}^{n} p_i \cdot f_T(p_i) \\
&= p_j \cdot f_{T'}(p_j) + p_k \cdot f_{T'}(p_k) - p_j \cdot f_T(p_j) - p_k \cdot f_T(p_k) \\
&= p_j \cdot f_T(p_k) + p_k \cdot f_T(p_j) - p_j \cdot f_T(p_j) - p_k \cdot f_T(p_k) \\
&= p_j \cdot (f_T(p_k) - f_T(p_j)) + p_k \cdot (f_T(p_j) - f_T(p_k)) \\
&= p_j \cdot (f_T(p_k) - f_T(p_j)) - p_k \cdot (f_T(p_k) - f_T(p_j)) \\
&= (p_j - p_k) \cdot (f_T(p_k) - f_T(p_j))
\end{aligned}$$

□

Next, we prove the optimality of this algorithm.

Lemma 5.2.2. *The tree T created by OPT-MSBST for the multiset of probabilities p solves the multiset binary search tree problem optimally.*

Proof. We prove this by inducting on $n = |M|$, the size of our multiset of probabilities. Consider the case where $n = 1$. In this case, we create the only tree possible which is obviously optimal. Suppose all trees created by *OPT-MSBST* with $k-1$ nodes are optimal. Consider the case where $n = k$. Let $p_{min} \in M$ be the probability of the node placed in the final position in BFS order in T. By the definition of *OPT-MSBST*, $p_{min} = \min_{p_i \in M}$.

We define M', a multiset of probabilities of size $k-1$ with probabilities $\{p'_1, p'_2, ..., p'_{k-1}\}$ such that

$$p'_i = \frac{p_i}{1 - p_{min}} \implies p_i = p'_i \cdot (1 - p_{min}).$$

Note that

$$\begin{aligned}
\sum_{p'_i \in M'} &= \sum_{i:p_i \in (M - \{p_{min}\})} p'_i \\
&= \sum_{i:p_i \in (M - \{p_{min}\})} \frac{p_i}{1 - p_{min}} \\
&= \frac{1}{1 - p_{min}} \cdot \sum_{i:p_i \in (M - \{p_{min}\})} p_i \\
&= \frac{1}{1 - p_{min}} \cdot (1 - p_{min}) \\
&= 1.
\end{aligned}$$

Thus, let T' be the tree created for M' using *OPT-MSBST*. By our induction hypothesis, it is optimal. Consider using *OPT-MSBST* to create T for M. During the running of the algorithm, we assume we use an arbitrary but fixed tie-breaking rule during sorting. Because of this rule, and by the definition of *OPT-MSBST*, for all $p'_i \in M'$, $f_{T'}(p'_i) = f_T(p_i)$.

Now, we consider the cost of our tree T:

$$\begin{aligned}
P_T &= \sum_{i:p_i \in (M - \{p_{min}\})} p_i \cdot f_T(p_i) + p_{min} \cdot f_T(p_{min}) \\
&= \sum_{i:p_i \in (M - \{p_{min}\})} p'_i \cdot (1 - p_{min}) \cdot f_{T'}(p_i) + p_{min} \cdot f_T(p_{min}) \\
&= (1 - p_{min}) \cdot \sum_{p'_i \in M'} p'_i \cdot f_{T'}(p_i) + p_{min} \cdot f_T(p_{min}) \\
&= (1 - p_{min}) \cdot P_{T'} + p_{min} \cdot f_T(p_{min})
\end{aligned}$$

Suppose for contradiction that T is not optimal, and there exists a better tree Z (which is optimal). We let f_Z represent the working depth of a node in Z. Note that there must exist an optimal tree Z with a $p_i \in M$ such that $f_Z(p_i)$ is maximized over all $p_i \in M$, and $p_i = min_{p_j \in M}$. Otherwise, by Lemma 5.2.1, we can swap the locations of the nodes p_i (the smallest probability) and p_j (the deepest node in the tree) and get a tree at least as good (if not better). We consider such a tree Z and let p_{min} be its minimum probability (which has maximum f_Z value).

We let Z' be tree made from Z by removing p_{min} from the tree over the probability distribution M' as in the proof of Lemma 5.2.2. As before,

$$p'_i = \frac{p_i}{1 - p_{min}} \implies p_i = p'_i \cdot (1 - p_{min}).$$

We now describe how we remove p_{min} to make Z'. If p_{min} is a leaf, we simply take out the leaf. If it is an internal node (it must be the parent of a leaf since it has maximum f value) we simply move one of its children (call it l), put it in p_{min}'s place, and make its possible other child (call it r) the child of l. Consider the cost of Z:

$$\begin{aligned}
P_Z &= \sum_{p_i \in M} p_i \cdot f_Z(p_i) \\
&= \sum_{i:p_i \in (M - \{p_{min}\})} p_i \cdot f_Z(p_i) + p_{min} \cdot f_z(p_{min}) \text{ as in 5.2.2, we have that:} \\
&= \sum_{p_i \in M'} (1 - p_{min}) \cdot p'_i \cdot f_Z(p_i) + p_{min} \cdot f_Z(p_{min}) \\
&= (1 - p_{min}) \cdot P_{Z'} + p_{min} \cdot f_Z(p_{min})
\end{aligned}$$

Note that our algorithm creates a balanced binary search tree, so the maximum value of f_T over M for T cannot be greater than the maximum value of f_Z for Z. Since we know that p_{min} has has the maximum value for f_T and for f_Z, we have that $f_T(p_{min}) \leq f_Z(p_{min})$. Thus, we have

$$\begin{gathered} P_Z < P_T \\ \implies (1 - p_{min}) \cdot P_{Z'} + p_{min} \cdot f_Z(p_{min}) < (1 - p_{min}) \cdot P_{T'} + p_{min} \cdot f_T(p_{min}) \\ \implies P_{Z'} < P_{T'} \end{gathered}$$

Which is a contradiction to the optimality of T' over $k+1$ nodes. Thus, T must be optimal as desired.

□

We are now ready to prove our main theorem.

Theorem 5.2.3. *The tree T created by OPT-MSBST for the multiset of probabilities p solves the multiset binary search tree problem optimally and is unique up to permutation of the assignment of nodes which have the same f_T value and permutation of the assignment of nodes which correspond to the same probability.*

Proof. Since we already know that *OPT-MSBST* provides an optimal solution, all that remains is to show that the tree T (created for probability multiset M with working node level function f_T) is unique up to the permutations described. Consider any optimal tree Z for probability multiset M. Suppose their exists $p_i \in M$ and $p_j \in M$ such that $p_i < p_j$ and $f_Z(p_i) < f_Z(p_j)$. By Lemma 5.2.1 swapping p_i and p_j gives a strictly better tree, a contradiction. Thus, no such p_i and p_j must exist. This exactly means that Z is identical to T up to permutations between identical probabilities and within the same values of f_T as required. □

Chapter 6

Conclusion and Open Problems

6.1 Conclusion

In this work, we examined several problems related to the optimum BST problem originally proposed (and solved) by Knuth in 1971 [36]. In Chapter 3 we showed that the Modified Entropy Rule first proposed by Güttler, Mehlhorn and Schneider in 1980 had a worst case expected cost of $H + 4$. This improved upon the previous best bound of $c \cdot H + 2$ where $c \approx 1.08$ [26].

In the next two chapters, we examined the problem under different models for external memory. In Chapter 4 we showed that under the Hierarchical Memory Model (HMM) by Aggarwal et al. Our two algorithms ApproxMWPaging and ApproxBSTPaging solved the problem in time $O(n \cdot \lg(m_1))$ and $O(n)$ respectively [2]. Moreover in sections 4.5 and 4.8, we showed the two solutions had worst case expected costs strictly less than

$$(H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{\text{rank}[i]}) \cdot c_h \text{ and}$$

$$(H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{\text{rank}[i]}) \cdot c_h \text{ respectively (theorems 4.5.5 and 4.8.2).}$$

We concluded by showing a mistake in the Master's thesis of Thite, and subsequently proving our solutions provided an improvement over Thite's in the related HMM_2 model.

In Chapter 5, we considered the optimum BST problem without explicit ordering on the keys. This essentially left us with a multiset of probabilities over which we attempted

to build an optimum BST. In section 5.2, we described an algorithm, OPT-MSBST, and proved that it solved the problem optimally. We also showed the solution was unique up to certain permutations.

We have tabulated these results with novel contributions in bold.

Algorithm	Model	Running Time	Worst case expected cost
Modified Minimum Entropy	RAM	$O(n^2)$	$\mathbf{C \leq H + 4}$
ApproxMWPaging	HMM	$\mathbf{O(n \cdot lg(m_1))}$	$\mathbf{C < (H + 1 + \sum_{i=0}^{n} q_i - q_0 - q_n - \sum_{i=0}^{m} q_{rank[i]}) \cdot c_h}$
ApproxBSTPaging	HMM	$\mathbf{O(n)}$	$\mathbf{C < (H + 1 - q_0 - q_n + q_{max} - \sum_{i=0}^{m'} pq_{rank[i]}) \cdot c_h}$
OPT-MSBST	RAM	$\mathbf{O(n \cdot lg(n))}$	—

Table 6.1: Models, running rimes, and worst case expected costs for algorithms discussed in this thesis.

Several interested and related problems still remain open. Firstly, is $H + 4$ a tight bound for the Modified Entropy Rule? We conjecture that this is not the case, and we believe the bound could be lowered to $H + 2$. Moreover, while the metric used to search for the root in this heuristic is good, it is definitely not perfect. The root chosen (up to the modifications of the rule) has the maximum 3-way entropy split. However, this is not necessarily the best root, as selecting larger probability keys as the root can decrease the cost of the tree. It would be interesting to consider what the correct metric would be for correctly selecting the root, possibly in a greedy manner. Chapter 5 is ultimately an introduction into considerations for this problem. Finally, the work in Chapter 4 can likely be extended to more recent models for external memory.

References

[1] M AdelsonVelskii and Evgenii Mikhailovich Landis. An algorithm for the organization of information. Technical report, DTIC Document, 1963.

[2] Alok Aggarwal, Bowen Alpern, Ashok Chandra, and Marc Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.

[3] Alok Aggarwal, Ashok K Chandra, and Marc Snir. Hierarchical memory with block transfer. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 204–216. IEEE, 1987.

[4] Bowen Alpern, Larry Carter, Ephraim Feig, and Ted Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12(2-3):72–109, 1994.

[5] Paul Joseph Bayer. *Improved bounds on the costs of optimal and balanced binary search trees*. Massachusetts Institute of Technology, Project MAC, 1975.

[6] R Bayer and E McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*, pages 107–141. ACM, 1970.

[7] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4):290–306, 1972.

[8] Peter Becker. A new algorithm for the construction of optimal b-trees. *Algorithm TheorySWAT'94*, pages 49–60, 1994.

[9] Peter Becker. Construction of nearly optimal multiway trees. In *Computing and Combinatorics*, pages 294–303. Springer, 1997.

[10] Michael A Bender, Erik D Demaine, and Martin Farach-Colton. Cache-oblivious b-trees. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 399–409. IEEE, 2000.

[11] Michael A Bender, Ziyang Duan, John Iacono, and Jing Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 29–38. Society for Industrial and Applied Mathematics, 2002.

[12] Andrew Donald Booth and Andrew JT Colin. On the efficiency of a new method of dictionary construction. *Information and Control*, 3(4):327–334, 1960.

[13] Prosenjit Bose and Karim Douïeb. Efficient construction of near-optimal binary and multiway search trees. In *Algorithms and Data Structures*, pages 230–241. Springer, 2009.

[14] Gerth Stølting Brodai and Rolf Fagerberg. Funnel heap-a cache oblivious priority queue. In *Algorithms and Computation*, pages 219–228. Springer, 2002.

[15] cppreference.com. std::map. http://en.cppreference.com/w/cpp/container/map. Accessed: 2016-02-02.

[16] Roberto De Prisco and Alfredo De Santis. On binary search trees. *Information Processing Letters*, 45(5):249–253, 1993.

[17] Erik D Demaine, Dion Harmon, John Iacono, and Mihai P a trascu. Dynamic optimality-almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

[18] V Thomas Dock. *FORTRAN IV programming*. [s.n.], Reston, VA, 1972.

[19] E Knuth Donald. The art of computer programming. *Sorting and searching*, 3:426–458, 1999.

[20] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 285–297. IEEE, 1999.

[21] Robert G Gallager. *Information theory and reliable communication*, volume 2. Springer, 1968.

[22] Adriano M Garsia and Michelle L Wachs. A new algorithm for minimum cost binary trees. *SIAM Journal on Computing*, 6(4):622–642, 1977.

[23] Edgar N Gilbert and Edward F Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

[24] Leo Gotlieb. Optimal multi-way search trees. *SIAM Journal on Computing*, 10(3):422–433, 1981.

[25] David Graves and Chris Hogue. *Fortran 77 Language Reference Manual*. Silicon Graphics, Inc.

[26] Reiner Güttler, Kurt Mehlhorn, and Wolfgang Schneider. Binary search trees: Average and worst case behavior. *Elektronische Informationsverarbeitung und Kybernetik*, 16:41–61, 1980.

[27] Thomas N Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM (JACM)*, 9(1):13–28, 1962.

[28] Yasuichi Horibe. An improved bound for weight-balanced tree. *Information and Control*, 34(2):148–151, 1977.

[29] TC Hu. A new proof of the tc algorithm. *SIAM Journal on Applied Mathematics*, 25(1):83–94, 1973.

[30] T.C. Hu. *Combinatorial Algorithms*. Addison-Wesley, MA, 1982.

[31] TC Hu, Daniel J Kleitman, and Jeanne K Tamaki. Binary trees optimum under various criteria. *SIAM Journal on Applied Mathematics*, 37(2):246–256, 1979.

[32] Te C Hu and Alan C Tucker. Optimal computer search trees and variable-length alphabetical codes. *SIAM Journal on Applied Mathematics*, 21(4):514–532, 1971.

[33] David A Huffman et al. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[34] Feffrey H Kingston. A new proof of the garsia-wachs algorithm. *Journal of Algorithms*, 9(1):129–136, 1988.

[35] DE Knuth. Sorting and searching.(the art of computer programming, vol. 3) addison-wesley. *Reading, MA*, pages 551–575, 1973.

[36] Donald E. Knuth. Optimum binary search trees. *Acta informatica*, 1(1):14–25, 1971.

[37] James F Korsh. Greedy binary search trees are nearly optimal. *Information Processing Letters*, 13(1):16–19, 1981.

[38] James F Korsh. Growing nearly optimal binary search trees. *Information Processing Letters*, 14(3):139–143, 1982.

[39] Kenneth C Louden. Compiler construction. *Cengage Learning*, 1997.

[40] Kurt Mehlhorn. Sorting and searching, volume 1 of data structures and algorithms, 1984.

[41] Michael S Paterson and F Frances Yao. Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, 13(1):99–113, 1992.

[42] Robert A Schumacker, Brigitta Brand, Maurice G Gilliland, and Werner H Sharp. Study for applying computer-generated images to visual simulation. Technical report, DTIC Document, 1969.

[43] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

[44] Haoyu Song, Murali Kodialam, Fang Hao, and TV Lakshman. Building scalable virtual routers with trie braiding. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[45] Shripad Thite. Optimum binary search trees on the hierarchical memory model. *arXiv preprint arXiv:0804.0940*, 2008.

[46] Vijay K. Vaishnavi, Hans-Peter Kriegel, and Derick Wood. Optimum multiway search trees. *Acta Informatica*, 14(2):119–133, 1980.

[47] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001.

[48] Jeffrey Scott Vitter and Mark H Nodine. Large-scale sorting in uniform memory hierarchies. *Journal of Parallel and Distributed Computing*, 17(1):107–114, 1993.

[49] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory, ii: Hierarchical multilevel memories. *Algorithmica*, 12(2-3):148–169, 1994.

[50] Peter F Windley. Trees, forests and rearranging. *The Computer Journal*, 3(2):84–88, 1960.

[51] Raymond W Yeung. Alphabetic codes revisited. *Information Theory, IEEE Transactions on*, 37(3):564–572, 1991.