Computer Science

New Analytical Methods for Online Binary Search Trees

Wanchote Po Jiamjitrak

Aalto University

DOCTORAL THESES

Aalto University publication series
DOCTORAL THESES 77/2023

New Analytical Methods for Online Binary Search Trees

Wanchote Po Jiamjitrak

A doctoral thesis completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall U3, Undergraduate Centre of the school on 29 May at 13:00.

Aalto University
School of Science
Computer Science
Combinatorics of Efficient Computations

Supervising professor
Asst. Prof. Parinya Chalermsook

Thesis advisor
Asst. Prof. Parinya Chalermsook

Preliminary examiners
Prof. James Ian Munro, University of Waterloo, Canada
Asst. Prof. László Kozma, Freie Universität Berlin, Germany

Opponents
Prof. James Ian Munro, University of Waterloo, Canada
Asst. Prof. László Kozma, Freie Universität Berlin, Germany

Aalto University publication series
DOCTORAL THESES 77/2023

ISBN 978-952-64-1282-5 (printed)
ISBN 978-952-64-1283-2 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-64-1283-2

Unigrafia Oy
Helsinki 2023

Finland

NORDIC SWAN ECOLABEL
Printed matter
4041-0619

Author
Wanchote Po Jiamjitrak

Name of the doctoral thesis
New Analytical Methods for Online Binary Search Trees

Publisher School of Science

Unit Computer Science

Series Aalto University publication series DOCTORAL THESES 77/2023

Field of research Theoretical Computer Science

Manuscript submitted 27 March 2023 **Date of the defence** 29 May 2023

Permission for public defence granted (date) 6 April 2023 **Language** English

☒ **Monograph** ☐ **Article thesis** ☐ **Essay thesis**

Abstract

This thesis aims to analyze the costs of various binary search tree (BST) algorithms. BST is one of the most fundamental data structures. The long-standing dynamic optimality conjecture states that there exists an online self-adjusting BST algorithm such that, starting with any initial tree, its access cost on any sequence X is O(OPT(X)+n) where OPT(X) is the cost of serving X on an offline optimal algorithm and n is the number of keys. Despite attempts from many groups of researchers, we believe that it is still far from being proven, primarily due to the lack of systematic techniques on the analytical side of algorithms. To address this, we explore the power of two analyzing methods in the context of binary search trees: potential functions and extremal combinatorics.

In the first part, we focus on the potential function method, which is widely used but mysterious in how it works. We introduce new charging schemes based on inversion counting, a popular potential function for list updates that has not been used in a BST context. An inversion potential function is arguably the most natural potential function, as it captures the difference between the states of two different algorithms. We illustrate our techniques in the context of both list updates and BSTs: (1) systematically deriving many known list update results and (2) unifying, strengthening, and deriving new bounds for BSTs.

In the second part, we explore and extend the use of extremal combinatorics in an amortized analysis of BSTs. The technique exploits the specific structures of the input of BSTs. This method encodes the execution log of BSTs in the form of a matrix and applies powerful theorems from forbidden matrix literature to upper-bound the cost. We propose an extra preprocessing step that decomposes the matrix into several simpler submatrices. We show the applications of the BST Greedy algorithm, one of the most promising candidates for being dynamically optimal, including deriving improved bounds for long-standing conjectures such as preorder traversal, postorder traversal, and path conjectures.

Keywords Online Algorithms, Data Structures, Binary Search Trees, List Updates, Extremal Combinatorics

ISBN (printed) 978-952-64-1282-5 **ISBN (pdf)** 978-952-64-1283-2

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2023

Pages 106 **urn** http://urn.fi/URN:ISBN:978-952-64-1283-2

Preface

This dissertation is the culmination of the research conducted during my doctoral studies at Aalto University, under the guidance of Professor Parinya Chalermsook. I am grateful to him for his invaluable supervision and the time he devoted to my research. Through my doctoral journey, I have gained immense knowledge, not only in research but also in philosophy. I firmly believe that a relaxed atmosphere and the freedom to explore one's ideas foster creativity for all purposes.

I would like to extend my sincere appreciation to my pre-examiners, Professor J. Ian Munro and Professor László Kozma, for their invaluable feedback and guidance throughout the thesis writing process. Their insightful comments have significantly improved the quality of my work and enhanced my writing skills. I am also grateful to my co-authors, whose contributions have been instrumental in the completion of this thesis. Without their support, this accomplishment would not have been possible.

Furthermore, I would like to express my gratitude to the Saarbrücken Graduate School of Computer Science at Saarland University for providing me with the necessary support and resources during my Master's degree, which have been instrumental in shaping my knowledge and skills in the field. In particular, I would like to extend my special thanks to Thatchaphol Saranurak, who introduced me to theoretical computer science and provided me with software support throughout my research.

Moreover, I acknowledge the support and encouragement of my colleagues and friends worldwide, who helped me maintain my composure during challenging times. Finally, I am grateful to my feline companion, Bubsie-bubsie, for her therapeutic massages and unwavering emotional support, albeit in exchange for her daily ration of food.

Helsinki, May 9, 2023,

Wanchote Po Jiamjitrak

Contents

List of Figures

List of Tables

Abbreviations

BST	Binary Search Tree
OPT	optimal algorithm, cost function: $X \to \mathbb{R}$
MF	Move-to-front algorithm
BIT	BIT algorithm
TS	Timestamp algorithm
MR	Move-to-root algorithm

Symbols

X	access sequence
x_t	access key at time t
$\mathcal{A}$	the algorithm in context
$\mathcal{O}$	optimal algorithm
$\mathcal{F}$	fractional optimal algorithm
S	Splay algorithm
G	Greedy algorithm
LF	lazy finger bound
WS	working set bound
$\mathcal{CP}$	canonical payment function: $[n] \to \{0,1\}$
Φ, Ψ, Γ	potential functions

Part I

Potential Function Analysis

1. Amortized analysis & potential function

1.1 Amortized analysis

Amortized analysis is a method to analyze of *average* cost per operation of the given algorithm. One way to calculate this is to divide the total cost by the number of operations. However, when the number of operations is not finite, things become more complicated. Consider the following example of an incrementing binary counter [1]. The counter starts from 0000 and the only operation on the binary counter is to increase the counter by 1. The cost of each operation is the number of bits that need to be flipped. For example, $01011 \rightarrow 01100$ has a cost of 3 because we flipped 2 bits from 1 to 0 and 1 bit from 0 to 1. See Table 1.1 for some examples of the cost of operations.

operation	cost
$0000 \rightarrow 0001$	1
$0001 \rightarrow 0010$	2
$0010 \rightarrow 0011$	1
$0011 \rightarrow 0100$	3
$0100 \rightarrow 0101$	1
$0101 \rightarrow 0110$	2

Table 1.1. Example of cost for each operation of binary counter

One way to bound the average cost is to add the same amount of coins into each operation, then use these coins to pay for the cost that occurs. The leftover coins can be stored to pay for some future operation. On the other hand, if the newly added coins are not enough, we can pay using the stored coins. Hence, if we can show that there are always enough coins to

[1] Thanks to prof. Raimund Seidel that he showed this example during an Algorithms and Data Structures course at Saarland University

pay for the cost of each operation, the amortized cost is upper-bounded by the number of coins we add to each operation.

Back to our example, first, imagine that each bit has its own *wallet*. We will show that adding 2 coins per operation is enough to pay for the cost. From $0000 \rightarrow 0001$, two coins are added and only one coin is used. So, the other coin is stored. Imagine that we store this coin on the 1-bit (now the rightmost bit) wallet. From $0001 \rightarrow 0010$, when the 1-bit is flipped to 0, we use the stored coin from the previous operation to pay for this cost. Then, for the two newly added coins, one is used to pay for the cost of flipping 0 to 1, and the other coin is stored on the 1-bit (now the second rightmost bit) wallet. By keeping the invariant that each 1-bit wallet always has a coin, we can make sure that we can pay for the cost of flipping 1 to 0. For flipping 0 to 1, notice that it occurs exactly once in each operation. This means we can use the two newly added coins: one pays for the flipping cost and one is stored in its wallet. This process can continue indefinitely. Hence, we can conclude that the amortized cost of the binary counter is at most 2.

In general, there can be some coins in the system initially. Let Φ_0 be the number of coins initially and Φ_i be the total number of coins after the i^{th} operation. We can write this as an inequality:

$$\Phi_i - \Phi_{i-1} + \text{cost}(i) \leq (\#\text{added coins})$$

where $\text{cost}(i)$ is the cost of the i^{th} operation. This means that the added coins are enough to pay for the cost and the difference between coins in wallets. When this holds for all i, we have

$$\sum_i (\Phi_i - \Phi_{i-1} + \text{cost}(i)) \leq \sum_i (\#\text{added coins})$$

$$\sum_i (\text{cost}(i)) - \Phi_0 \leq \sum_i (\#\text{added coins})$$

The second inequality is due to the fact that Φ_i is non-negative. Intuitively, the average cost is roughly the number of coins we added per operation (when the number of operations is sufficiently long, Φ_0 becomes insignificant.)

1.2 Competitive analysis

In list update and binary search tree problems, we often deal with competitive analysis. In competitive analysis, we often compare two algorithms. The idea is similar to amortized analysis, except that the number of added coins in each round may not be the same number. In competitive analysis, the number of added coins depends on the comparison bounds. For

example, there are two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Suppose that we want to show that the amortized cost of $\mathcal{A}_1$ is at most the cost of $\mathcal{A}_2$. Similarly to the amortized analysis, we can show that, given an input sequence $X = (x_1, \dots x_m)$, for all i,

$$\Phi_i - \Phi_{i-1} + \text{cost}_{\mathcal{A}_1}(x_i) \leq \text{cost}_{\mathcal{A}_2}(x_i)$$

where $\text{cost}_{\mathcal{A}}(x_i)$ is the cost of $\mathcal{A}$ to serve input x_i (we will define this formally in the next chapter). This gives

$$\sum_i \text{cost}_{\mathcal{A}_1}(x_i) - \Phi_0 \leq \sum_i \text{cost}_{\mathcal{A}_2}(x_i)$$

and

$$\text{cost}_{\mathcal{A}_1}(X) - \Phi_0 \leq \text{cost}_{\mathcal{A}_2}(X)$$

where $\text{cost}_{\mathcal{A}}(X) = \sum_i \text{cost}_{\mathcal{A}}(x_i)$. Roughly speaking, with some initial coins, $\mathcal{A}_1$ is at least as good as $\mathcal{A}_2$. Instead, if we want to show that $\mathcal{A}_1$ is at least as good as an α factor of $\mathcal{A}_2$, the number of added coins in the i^{th} round is $\alpha \cdot \text{cost}_{\mathcal{A}_2}(x_i)$. This eventually gives us

$$\text{cost}_{\mathcal{A}_1}(X) - \Phi_0 \leq \alpha \cdot \text{cost}_{\mathcal{A}_2}(X)$$

In the context of online algorithms, we usually compare the cost of our algorithm with an offline optimal algorithm. Given an input sequence $X = (x_1, \dots x_m)$, an algorithm $\mathcal{A}$ is online if after each time $t \in [m]$, $\mathcal{A}$ only knows $(x_1, \dots, x_t)$. In contrast to the offline algorithm, the entire input is known beforehand. Let $\mathsf{OPT}(X)$ be the optimal offline algorithm for the given sequence X. We say that an algorithm $\mathcal{A}$ is *α-competitive* if there exists a constant β such that $\text{cost}_{\mathcal{A}}(X) \leq \alpha \cdot \text{cost}_{\mathsf{OPT}}(X) + \beta$ for all X. Notice that this is similar to the previous inequality when $\beta = \Phi_0$. For binary search tree literature, it is common that β can be a function of a number of keys. In the case that the algorithm is randomized, $\text{cost}_{\mathcal{A}}(X)$ is substituted by $\mathbb{E}[\text{cost}_{\mathcal{A}}(X)]$. Note that, in this thesis, we consider only randomized algorithm against the *oblivious* adversary, i.e., an access sequence is specified before $\mathcal{A}$ starts.

1.3 Potential function and accounting method

This section briefly discusses the connection between potential function and explicit accounting method (which we will use in the later sections).

Before doing so, we start by briefly summarizing the potential function method. Roughly speaking, in order to prove the inequality $\text{cost}_{\mathcal{A}}(X) \leq \alpha \cdot \text{cost}_{\mathsf{OPT}}(X) + \beta$, one defines a real-valued function $\Phi(S)$ that depends on the state of execution S of $\mathcal{A}$ and OPT.

Let $X = (x_1, ..., x_m)$ be an input sequence. Suppose that the states of algorithm change from $S_{\text{init}} = S_0 \to S_1 \to S_2 \ldots \to S_m = S_{\text{final}}$ (i.e., S_i is the state after serving the input x_i). The potential function method aims to prove that:

$$\Phi(S_i) \leq \Phi(S_{i-1}) - \text{cost}_{\mathcal{A}}(x_i) + \alpha \cdot \text{cost}_{\text{OPT}}(x_i)$$

Equivalently:

$$\Phi(S_i) - \Phi(S_{i-1}) + \text{cost}_{\mathcal{A}}(x_i) \leq \alpha \cdot \text{cost}_{\text{OPT}}(x_i)$$

This is sufficient because, if we sum from time 1 to m, we have

$$\sum_{1}^{m} (\Phi(S_i) - \Phi(S_{i-1}) + \text{cost}_{\mathcal{A}}(x_i)) \leq \alpha \cdot \sum_{1}^{m} (\text{cost}_{\text{OPT}}(x_i))$$

Expanding the sum:

$$\Phi(S_m) - \Phi(S_{m-1}) + \Phi(S_{m-1}) - \Phi(S_{m-2}) + \ldots + \Phi(S_1) - \Phi(S_0) + \sum_{1}^{m} \text{cost}_{\mathcal{A}}(x_m) \leq \alpha \cdot \sum_{1}^{m} \text{cost}_{\text{OPT}}(x_t)$$

Hence, we obtain

$$\Phi(S_{\text{final}}) - \Phi(S_{\text{init}}) + \sum_{1}^{m} \text{cost}_{\mathcal{A}}(x_m) \leq \alpha \cdot \sum_{1}^{m} \text{cost}_{\text{OPT}}(x_t)$$

which implies α-competitive for $\mathcal{A}$ (given that $\Phi(S_{\text{init}})$ is appropriately bounded).

In this thesis, we view the real-valued function $\Phi(S)$ as a collection of "coins" that are *explicitly* placed somewhere in the system. Those coins are placed at some specific positions in each state of the algorithm. Proving the potential function argument is equivalent to the following: When an operation is executed, we deduct $\text{cost}_{\mathcal{A}}(x_t)$ specific coins from the system (to pay for the access cost of $\mathcal{A}$) and add specific $\alpha \cdot \text{cost}_{\text{OPT}}(x_t)$ coins, therefore having $\Phi(S) + \alpha \cdot \text{cost}_{\text{OPT}}(x_t) - \text{cost}_{\mathcal{A}}(x_t)$ after the operation. It suffices to argue that at least $\Phi(S')$ coins remain.

The accounting method, in contrast to the potential function, better highlights the process of comparing the algorithm's cost with the optimal cost. An accounting method makes things easier to verify since explicit costs (coins) are associated with the objects. Coins are usually placed in the place where they will be used in the future, for example, we put a coin on each 1-bit on the binary counter.

While a potential method is widely used in binary search tree context, the literature on an accounting method is missing. We expected that this could lead to a new understanding of binary search trees. We first start with a

list update problem. The list update problem is considered to be *easier* to achieve a constant competitive ratio and most importantly, an inversion potential function is known. An inversion potential function is arguably the most natural potential function since it measures the distance between two states. Afterward, we later apply the idea to a BST context.

See [49, 25, 47, 11] for more comprehensive survey about the topic.

2. List update

List update is a fundamental online problem. Move-to-front is one of the first algorithms that are introduced for an online list update problem. In 1985, Sleator and Tarjan have shown that the Move-to-front (MF) algorithm is 2-competitive [47], which matches the lower bound for deterministic algorithms. The problem has attracted a lot of attention (see e.g. [7, 3, 33] and references therein) over the past three decades, culminating in the best-known upper bound of 1.6 [4] and the lower bound of 1.5 [50] on the competitive ratios of randomized algorithms in the *oblivious adversary model*; these bounds have so far stood for two decades. Besides research, the simplicity of the model made the problem appealing from the curriculum perspective: Together with the ski rental problem, list update has been used often as the very first examples for introducing the concept of online algorithms, potential function, and amortized analysis.

The factor-2 competitive analysis of MF (as well as many other algorithms for list update) relies on the *potential function method*. Despite admitting a very short proof (with the right wording optimization, the proof can be compressed into less than half a page), several challenges remain, for instance, "why do these potential functions (magically) work?" or "how do I adapt them to analyze related algorithms?"

In this chapter, we make a proof more illustrative by 'accounting' the values of inversion potential function over the keys (i.e. we still analyze the same potential function as done in the literature but in a more illustrative way), whereby direct charging arguments can be applied. Every step of the proof is intuitive. For the purpose of exposition, we present the analysis of 3 different algorithms for the list update problem: MF [9], BIT [44], and deterministic TIMESTAMP (TS) [2]. These three algorithms are sufficient as the ingredients in the state-of-the-art result [4]. Compared to the known proofs, our proof is longer, but it offers more intuitions, allowing for possible adaptation by the algorithm's designers. For instance, we believe that readers who read and understand our proof for MF would be able to guess the potential function for BIT before seeing it. Understanding the idea of the potential function of list update will be beneficial as a

background for analysing BST in the next chapters.

2.1 Model

We are given an initial linked list of n keys, denoted by $[n] = \{1, 2, \ldots, n\}$. An access sequence is described by $X = (x_1, \ldots, x_m) \in [n]^m$ where each x_t is revealed at time t, and the access incurs the cost of $i - 1$ if x_t is the i^{th} key in the list.[1] This cost is called *access cost*; denoted by access-cost$_{\mathcal{A}}(x)$ where x is the access key before the update by algorithm $\mathcal{A}$.

After each access, the list can be modified in the following ways:

1. Move x to an arbitrary position toward the front. This update costs nothing. This is called a *free move*.
2. Swap any two consecutive elements in the list. This update costs 1 per swap. This is called a *paid move.*

These are called *update costs*, denoted by update-cost$_{\mathcal{A}}(x)$ where x is the access key before the update. Note that update-cost$_{\mathcal{A}}(x)$ = #(paid moves).[2] Given the access sequence $X = (x_1, x_2, \ldots, x_m)$, the cost of algorithm $\mathcal{A}$ on this access sequence is $\text{cost}_{\mathcal{A}}(X) = \sum_t \text{cost}_{\mathcal{A}}(x_t) = \sum_t \text{access-cost}_{\mathcal{A}}(x_t) + \sum_t \text{update-cost}_{\mathcal{A}}(x_t)$.

2.2 Potential function

Let $\mathcal{A}$ be a list. For each $a, b \in [n]$, we use notation $a \prec_{\mathcal{A}} b$ to denote "a appears before b on list $\mathcal{A}$". Let $\lhd_{\{a,b\}}$ be the key in $\{a, b\}$ closer to the front of the $\mathcal{O}$ list. In particular, $\lhd_{\{a,b\}} = a$ if $a \prec_{\mathcal{O}} b$, otherwise $\lhd_{\{a,b\}} = b$. For each key $z \in [n]$, let $J_{\mathcal{A}}(z) = \{y \mid y \prec_{\mathcal{A}} z\}$ and $I_{\mathcal{A}}(z) = \{z\} \cup J_{\mathcal{A}}(z)$. See Figure 2.1.

Let $\mathcal{A}$ be an algorithm we want to analyze and $\mathcal{O}$ be an optimal algorithm. Both $\mathcal{A}$ and $\mathcal{O}$ store the keys in $[n]$. We want to have a potential function that captures the "difference" between $\mathcal{A}$ and $\mathcal{O}$. The most natural scheme (which does not always work) often used in the context of online algorithms for this purpose is *inversion*.

Definition 2.1 (Inversion)**.** Let $z, \alpha \in [n]$. We say that z forms an *inversion* with α if $z \in I_{\mathcal{A}}(\alpha)$ and $\alpha \in I_{\mathcal{O}}(z)$.

Note that "z forms an *inversion* with α" is different from "α forms an *inversion* with z".

[1] This is called a *partial* cost model. As opposed to a *full* cost model, where cost is i instead of $i - 1$. The two models have different lower bounds [7, 50, 6].

[2] While this model is commonly used, it is unrealistic. For example, changing the structure of the list from $[a_1, \ldots, a_{\frac{r}{2}}, a_{\frac{r}{2}+1}, \ldots, a_r]$ to $[a_{\frac{r}{2}+1}, \ldots, a_r, a_1, \ldots, a_{\frac{r}{2}}]$ takes a quadratic number of swaps, whereas a linear number of non-adjacent swaps suffices.

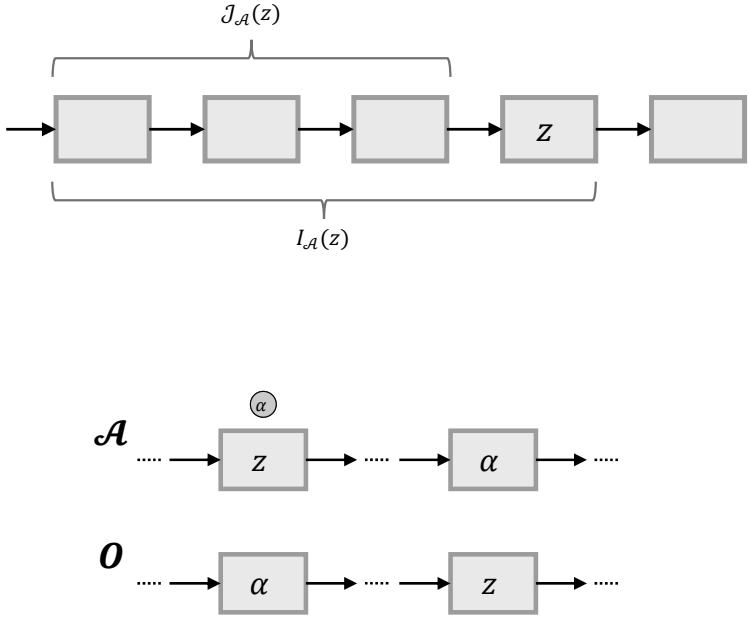


Figure 2.2. An illustration of coin for base potential function

Definition 2.2 (Base Potential function)**.** Define the potential $\Phi = \Phi_{\mathcal{A},\mathcal{O}}$ at each state of execution of our algorithm $\mathcal{A}$ and optimal $\mathcal{O}$ as follows. Let $\Phi(z,\alpha) = 1$ if z forms an inversion with α; otherwise, $\Phi(z,\alpha) = 0$. The potential is defined as $||\Phi|| = \sum_{z,\alpha} \Phi(z,\alpha)$. This is the total number of inversions.

We "visualize" our potential function value as a collection of "coins". Whenever $\Phi(z,\alpha) = 1$, one can imagine there is a coin of label α (or α-coin) placed at key z in $\mathcal{A}$ whenever z forms an inversion with α (see Figure 2.2). By definition, z always forms an inversion with itself, so there is always a z-coin at z. The coin interpretation will be used crucially in our analysis.

Lemma 2.3. *For all $z,\alpha \in [n]$ such that $z \prec_{\mathcal{A}} \alpha$, we have $\Phi(z, \lhd_{\{z,\alpha\}}) = 1$.*

Proof. If $\lhd_{\{z,\alpha\}} = z$, then this is trivial because z always forms an inversion with itself and thus has its own coin. If $\lhd_{\{z,\alpha\}} = \alpha$, this means $\alpha \prec_{\mathcal{O}} z$ and hence z forms an inversion with α. □

Definition 2.4 (Canonical payment function)**.** Consider algorithm $\mathcal{A}$ and optimal $\mathcal{O}$. The *canonical payment function* for accessing x is a function $\mathcal{CP} = \mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ such that for each z such that $z \prec_{\mathcal{A}} x$, we have $\mathcal{CP}(z, \lhd_{\{z,x\}}) = 1$. The function is 0 everywhere else (see Figure 2.3).

We will simply write $\mathcal{CP}$ instead of $\mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ when it is clear from the context. The role of $\mathcal{CP}$ is to specify which coins are deducted to pay for the access cost of x.

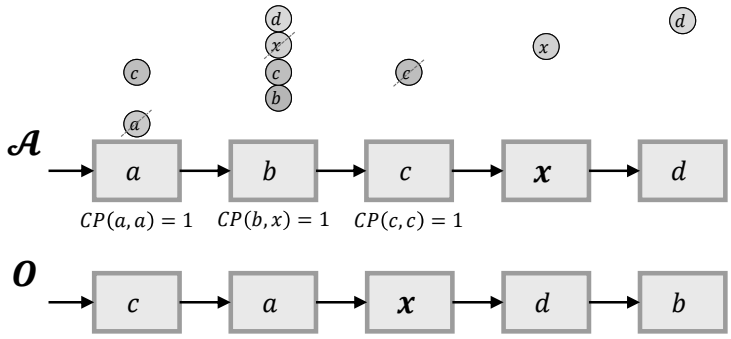


Figure 2.3. An example of $\mathcal{CP}$ function when x is accessed, along with the corresponding three coins which are deducted to pay access-cost$_{\mathcal{A}}(x)$. The supports of $\mathcal{CP}$ are 0 everywhere else.

Observation 2.5. *When accessing x,* $\sum_{z,\alpha} \mathcal{CP}(z,\alpha) = |J_{\mathcal{A}}(x)| = \text{access-cost}_{\mathcal{A}}(x)$.

Observation 2.6. *When accessing x, for all z,* $\Phi(z, \lhd_{\{z,x\}}) \geq \mathcal{CP}(z, \lhd_{\{z,x\}})$.

Proof. Directly follow by Lemma 2.3. □

Next, we introduce some algorithms in the list update literature. Then, we analyze them using our potential functions and coins arguments. We always start with two lists: $\mathcal{A}$ for the algorithm and $\mathcal{O}$ for the optimal list. After x is accessed, $\mathcal{A}$ is updated to $\mathcal{A}'$ and we upper bound the change in potential. Then, $\mathcal{O}$ is updated to $\mathcal{O}'$ and we upper bound the change in potential again.

2.3 Move-to-front

Now we use the inversion potential function to analyze the MF algorithm. The result is known from [47, 9].

2.3.1 Move-to-front algorithm

Move-to-front is the list update algorithm that performs as follows. After accessing each key x, the algorithm will update the list by moving x to the front of the list. All other relative orders are the same. See Figure 2.4 for an example. Notice that Move-to-front only uses free moves, so $\text{cost}_{\mathcal{A}}(X) = \text{access-cost}_{\mathcal{A}}(X)$.

Theorem 2.7. $\forall X, \text{cost}_{\mathcal{A}}(X) \leq 2 \cdot \text{cost}_{\mathcal{O}}(X)$

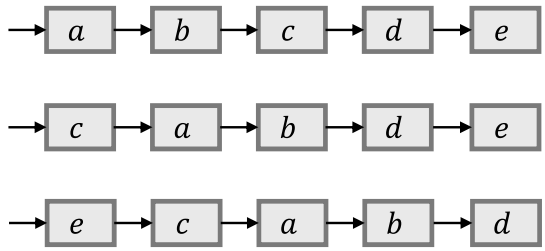


Figure 2.4. An example of Move-to-front algorithm where the top list in the figure is the initial list. Then, the first access is c and the second access is e

2.3.2 Move-to-front analysis

We will upper bound the change of potential function Φ as follows:

$$||\Phi_{\mathcal{A}',\mathcal{O}'}|| \leq ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + 2 \cdot \text{access-cost}_{\mathcal{O}}(x) + \text{update-cost}_{\mathcal{O}}(x)$$

Since $2 \cdot \text{access-cost}_{\mathcal{O}}(x) + \text{update-cost}_{\mathcal{O}}(x) \leq 2 \cdot \text{cost}_{\mathcal{O}}(x)$, this implies Theorem 2.7. Deducting $\mathcal{CP}$ from Φ can be seen as, for each $z \prec_{\mathcal{A}} x$, removing $\lhd_{\{z,x\}}$-coin from z. Let $\widehat{\Phi}$ be the function after removing the coins, so $||\widehat{\Phi}|| = ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$.

Lemma 2.8. $||\Phi_{\mathcal{A}',\mathcal{O}}|| \leq ||\widehat{\Phi}|| + 2 \cdot \text{access-cost}_{\mathcal{O}}(x)$

Proof. Recall that $\Phi_{\mathcal{A}',\mathcal{O}} - \widehat{\Phi} = \sum_{z,\alpha}(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha))$. It suffices to consider only $(z,\alpha) \in S$ where $S = \{(z,\alpha) \mid \Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) = 1, \widehat{\Phi}(z,\alpha) = 0\}$. We divide into two cases:

1. Let $S_1 = S \cap \{(z,\alpha) \mid \Phi_{\mathcal{A},\mathcal{O}} = 1\}$. We will show that $\sum_{(z,\alpha)\in S_1}(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha)) \leq \text{access-cost}_{\mathcal{O}}(x)$. This corresponds to existing inversions that pay their coins out but remains inversions.

   By $\widehat{\Phi}(z,\alpha) = 0, \Phi_{\mathcal{A},\mathcal{O}} = 1$, it means that one $\lhd_{\{z,x\}}$-coin are deducted from each $z \in J_{\mathcal{A}}(x)$. For those z that $\lhd_{\{z,x\}} = z$, it means $z \prec_{\mathcal{O}} x$. There are at most $\text{access-cost}_{\mathcal{O}}(x)$ such keys that possibly need coins. For those z that $\lhd_{\{z,x\}} = x$, it means $x \prec_{\mathcal{O}} z$. Since $x \prec_{\mathcal{A}'} z$ for all z, we have $\Phi_{\mathcal{A}',\mathcal{O}}(z,x) = 0$. Hence, no coins is needed.

2. Let $S_2 = S \cap \{(z,\alpha) \mid \Phi_{\mathcal{A},\mathcal{O}} = 0\}$. We will show that $\sum_{(z,\alpha)\in S_2}(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha)) \leq \text{access-cost}_{\mathcal{O}}(x)$. This corresponds to new inversions.

   Since we only move x, the new inversions must incur by x. Since all keys appear after x in $\mathcal{A}'$, only keys that appear before x in $\mathcal{O}$ could form new inversions. There are at most $\text{access-cost}_{\mathcal{O}}(x)$ such keys.

□

Lemma 2.9. $||\Phi_{\mathcal{A}',\mathcal{O}'}|| \leq ||\Phi_{\mathcal{A}',\mathcal{O}}|| + \text{update-cost}_{\mathcal{O}}(x)$

Proof. There are two ways $\mathcal{O}$ can be updated: free moves and paid moves. We decompose each part into one move at a time.

- Fix one free move. Let $\Phi^{(1)}_{\mathcal{A}',\mathcal{O}'}$ and $\Phi^{(2)}_{\mathcal{A}',\mathcal{O}'}$ be a potential before and after the move. Let y be the key that switches with x. The relative order of y with x in $\mathcal{O}$ becomes the same as $\mathcal{A}$. So, $||\Phi^{(2)}_{\mathcal{A}',\mathcal{O}'}|| \leq ||\Phi^{(1)}_{\mathcal{A}',\mathcal{O}}||$.
- Fix one paid move. Let $\Phi^{(1)}_{\mathcal{A}',\mathcal{O}'}$ and $\Phi^{(2)}_{\mathcal{A}',\mathcal{O}'}$ be a potential before and after the move. Notice that each move can only create at most one inversion. So, $||\Phi^{(2)}_{\mathcal{A}',\mathcal{O}'}|| \leq ||\Phi^{(1)}_{\mathcal{A}',\mathcal{O}}|| + \text{update-cost}_{\mathcal{O}}(x)$.

□

2.3.3 Discussion

In fact, the access sequence is also allowed the *insert* and *delete* operations. For simplicity, we only consider the access operations. Our MF proofs can be modified easily to hold for insert and delete. For insertion, we just add coins to maintain the potential, the number of coins is bounded by the insertion cost of OPT. For deletion, it is similar to access.

2.4 BIT

BIT was introduced in [44] and shown to be 1.75-competitive. In this section, we use the modified inversion potential function to analyze the same result for this algorithm.

2.4.1 BIT algorithm

BIT is a randomized list update algorithm which maintains $bit(z) \in \{0,1\}$ for each key. Initially, these bits are randomly assigned, that is, pick a random vector $\vec{B} = (B_1, B_2, \ldots, B_n) \in \{0,1\}^n$ and define $bit(z) = B_z$ for each $a \in [n]$. At each time, after accessing key x, the algorithm will flip $bit(x)$ from $(0 \Rightarrow 1)$ or $(1 \Rightarrow 0)$. After being flipped, if $bit(x) = 1$, the algorithm updates the list by moving x to the front of the list; otherwise, x remains in the same position.

Notice that BIT only uses free moves, so $\text{cost}_{\mathcal{A}}(X) = \text{access-cost}_{\mathcal{A}}(X)$.

Theorem 2.10. $\forall X, \mathbb{E}_{\vec{B}}[\text{cost}_{\mathcal{A}}(X)] \leq 1.75 \cdot \text{cost}_{\mathcal{O}}(X)$[3]

2.4.2 BIT analysis

Define a potential $\Psi(z,\alpha) = \Phi(z,\alpha) + \Phi_1(z,\alpha)$ for all $z, \alpha \in [n]$ where $\Phi_1(z,\alpha) = 1$ if $z \neq \alpha$ and z forms an inversion with α with $bit(\alpha) = 1$. One can imagine

[3] We remark the optimal here is in the oblivious adversary model.

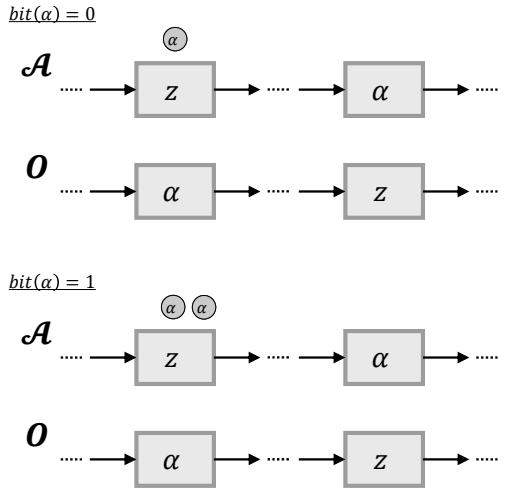


Figure 2.5. An illustration of coins for BIT

that there is *one* (or *two*) α-coin placed at key z in $\mathcal{A}$ for each inversion with $bit(\alpha) = 0$ (or 1). See Figure 2.5.

Remark: This potential function is in fact very intuitively derived from Φ. Notice that $bit(\alpha)$ reflects the "state" of α. When $bit(\alpha) = 0$ (regular state), the algorithm would perform exactly the same as Move-to-front, while $bit(\alpha) = 1$, it would take one access to get back to the regular Move-to-front state. So the potential function is one coin per inversion when $bit(\alpha) = 0$ and two coins per inversion when $bit(\alpha) = 1$.

We will upper bound the change of potential function Ψ as follows:

$$||\Psi_{\mathcal{A}',\mathcal{O}'}|| \leq ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + 1.75 \cdot \text{access-cost}_{\mathcal{O}}(x) + 1.5 \cdot \text{update-cost}_{\mathcal{O}}(x)$$

Since $1.75 \cdot \text{access-cost}_{\mathcal{O}}(x) + 1.5 \cdot \text{update-cost}_{\mathcal{O}}(x) \leq 1.75 \cdot \text{cost}_{\mathcal{O}}(x)$, this implies Theorem 2.10. Deducting $\mathcal{CP}$ from Ψ can be seen as, for each $z \prec_{\mathcal{A}} x$, removing $\lhd_{\{z,x\}}$-coin from z. Let $\widehat{\Psi}$ be the function after removing the coins, so $||\widehat{\Psi}|| = ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$.

Lemma 2.11. $||\Psi_{\mathcal{A}',\mathcal{O}}|| \leq ||\widehat{\Psi}|| + 1.75 \cdot \text{access-cost}_{\mathcal{O}}(x)$

Proof. Recall that $\Psi_{\mathcal{A}',\mathcal{O}} - \widehat{\Psi} = \sum_{z,\alpha}(\Psi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Psi}(z,\alpha))$. It suffices to consider only $(z,\alpha) \in S$ where $S = \{(z,\alpha) \mid \Psi_{\mathcal{A}',\mathcal{O}}(z,\alpha) > \widehat{\Psi}(z,\alpha)\}$. We divide it into three cases:

1. Let $S_1 = S \cap \{(z,\alpha) \mid \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) > 0 \text{ and } \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) > \widehat{\Psi}(z,\alpha)\}$. We will show that $\sum_{(z,\alpha)\in S_1}(\Psi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Psi}(z,\alpha)) \leq \text{access-cost}_{\mathcal{O}}(x)$. This corresponds to existing inversions that pay their coins out but remains inversions.

For each $z \in J_{\mathcal{A}}(x)$, one $\lhd_{\{z,x\}}$-coin are deducted. For those z that $\lhd_{\{z,x\}} = z$, it means $z \prec_{\mathcal{O}} x$. There are at most access-cost$_{\mathcal{O}}(x)$ such keys. For those z that $\lhd_{\{z,x\}} = x$, it means $x \prec_{\mathcal{O}} z$.

- If $bit(x)$ is flipped from 1 to 0, then x remains in the same position. Since $\Psi_{\mathcal{A},\mathcal{O}}(z,x) = 2$ and $\Psi_{\mathcal{A}',\mathcal{O}}(z,x) = \widehat{\Psi}(z,\alpha) = 1$ for each inversion, we do not need any coins for this.
- If $bit(x)$ is flipped from 0 to 1, then x will be moved to the front of the list. So, $x \prec_{\mathcal{A}'} z$ for all z and hence $\Psi_{\mathcal{A}',\mathcal{O}}(z,x) = 0$.

2. Let $S_2 = S \cap \{(z,\alpha) \mid \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) > 0 \text{ and } \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) = \widehat{\Psi}(z,\alpha)\}$. We will show that $\sum_{(z,\alpha)\in S_2}(\Psi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Psi}(z,\alpha)) \le 0$. This corresponds to existing inversions that need more coins from bit flipping.

   This is only possible when $bit(x)$ is flipped from 0 to 1, which could affect $z \prec_{\mathcal{A}'} x$. Since x is moved to the front, there is no z such that $z \prec_{\mathcal{A}'} x$.

3. Let $S_3 = S \cap \{(z,\alpha) \mid \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 0\}$. We will show that $\sum_{(z,\alpha)\in S_3}(\Psi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Psi}(z,\alpha)) \le 0.75 \cdot \text{access-cost}_{\mathcal{O}}(x)$. This corresponds to new inversions.

   When $bit(x)$ is flipped from 0 to 1, then x will be moved to the front of the list. The new inversions must incur by x. Since all keys appear after x in $\mathcal{A}'$, only keys that appear before x in $\mathcal{O}$ form inversions. Let $\mathbf{I}[\cdot]$ denote the indicator function that evaluates a true predicate to 1, and 0 otherwise. We have to add either 1 or 2 coins to z depending on their $bit(z)$. The number of coins added is upper bounded by

$$\sum_{z\in J_{\mathcal{O}}(x)} ((1\mathbf{I}[bit(z)=0] + 2\mathbf{I}[bit(z)=1]) \cdot \mathbf{I}[bit(x)=1]) + (0 \cdot \mathbf{I}[bit(x)=0])$$

   Hence,

$$\begin{aligned}
\mathbf{E}_{\vec{B}}[\#(\text{coins added})] &\le \sum_{z\in J_{\mathcal{O}}(x)} ((1\cdot 0.5 + 2\cdot 0.5)\cdot 0.5) + (0\cdot 0.5) \\
&= \sum_{z\in J_{\mathcal{O}}(x)} 0.75 \\
&\le 0.75 \cdot \text{access-cost}_{\mathcal{O}}(x)
\end{aligned}$$

   Here, we use the fact that $\{bit(z)\}_{z\in[n]}$ are independent random variables.

□

Lemma 2.12. $||\Psi_{\mathcal{A}',\mathcal{O}'}|| \le ||\Psi_{\mathcal{A}',\mathcal{O}}|| + 1.5 \cdot \text{update-cost}_{\mathcal{O}}(x)$

Proof. Here, we use an observation by Reingold and Westbrook [43] that there is an optimal list update algorithm that never uses a free move. So,

we will only analyze our algorithm against the optimal that uses only paid moves. [4] If $\mathcal{O}$ uses paid moves, notice that each move can only create at most one inversion. Assume that z forms a new inversion with α.

$$\#(\text{coins added}) = 1\mathbf{I}[bit(\alpha) = 0] + 2\mathbf{I}[bit(\alpha) = 1]$$

$$\begin{aligned}\mathbf{E}_{\vec{B}}[\#(\text{coins added})] &= 1 \cdot 0.5 + 2 \cdot 0.5 \\ &= 1.5\end{aligned}$$

□

2.5 Deterministic TIMESTAMP

TIMESTAMP algorithms were introduced in [2] and shown to achieve a competitive ratio of $\max\{2-p, 1+p(2-p)\}$ for any real $p \in [0,1]$. When $p = (3-\sqrt{5})/2$, the competitive ratio is optimal at the golden ratio $(1+\sqrt{5})/2 \approx 1.62$. In this section, we will modify the inversion potential function to analyze the TIMESTAMP algorithm when $p = 0$ (deterministic TIMESTAMP algorithm or TS), which gives a competitive ratio of 2. TS and BIT are ingredients for the best-known randomized algorithm [4], which is 1.6-competitive.

2.5.1 Deterministic TIMESTAMP algorithm

Let x be the accessed key. Let y be the key closest to the front of the list that precedes x and that gets accessed *at most once* since the last access of x. The Timestamp algorithm is as follows: after key x is accessed, insert x in front of key y. If there is no such y, x stays in the same place. If x is accessed for the first time, move x to the front of the list. [5]

Notice that $\mathcal{A}$ only use free moves, so $\text{cost}_{\mathcal{A}}(X) = \text{access-cost}_{\mathcal{A}}(X)$.

Theorem 2.13. $\forall X, \text{cost}_{\mathcal{A}}(X) \leq 2 \cdot \text{cost}_{\mathcal{O}}(X)$

2.5.2 Deterministic TIMESTAMP analysis

First, let us introduce a model that will help in the analysis. This model was first introduced by Bentley and McGeoch [9] and is widely used in literature. Consider the model where we deal with a collection of all $\binom{n}{2}$ possible lists of size two. The cost of access x in this model is defined

[4] This is because, instead of using a free move at time t, it can use the equivalent paid moves at the end of time $t-1$ and yield the same cost.

[5] the algorithm is a little different from the original paper where Timestamp does not move the first access of each key. This change still keeps the main idea of the algorithm while making it easier to analyze via potential function as in [8][46]

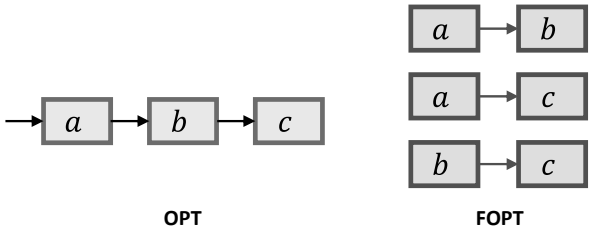


Figure 2.6. An example of $\mathcal{F}$ lists that correspond to $\mathcal{O}$

as #(lists where x is behind). Let *free move* and *paid move* be defined the same as before. Let $\mathcal{F}$ be the optimal algorithm in this model. It is not hard to see that $\forall X, \text{cost}_{\mathcal{F}}(X) \le \text{cost}_{\mathcal{O}}(X)$ since every operation of $\mathcal{O}$ can be simulated by $\mathcal{F}$ with the same cost. See an example in Figure 2.6. Therefore, it suffices for us to show that $\forall X, \text{cost}_{\mathcal{A}}(X) \le 2 \cdot \text{cost}_{\mathcal{F}}(X)$ to prove Theorem 2.13. Note that $\mathcal{F}$ might not be consistent with a full list.

The main benefit of this model is, since each list has size 2, the behavior of the optimal algorithm is simpler to understand.

Observation 2.14 ([30])**.** *For the list of size 2 (i and j), there is an optimal algorithm where i is always in front of j, if i gets accessed twice in a row relative to j (and vice versa).*

Define a potential $\Lambda(z,\alpha) = \Phi(z,\alpha) + \Phi_2(z,\alpha)$ for all $z, \alpha \in [n]$ where $\Phi_2(z,\alpha) = 1$ if all following conditions hold:

1. $z \neq \alpha$
2. z forms an inversion with α
3. z is accessed *at least twice* since the last access of α.

One can imagine that there is *one* (or *two*) α-coin placed at key z in $\mathcal{A}$ for each inversion such that z is accessed *at most once* (or *at least twice*).

We will upper bound the change of potential function Λ as follows:

$$||\Lambda_{\mathcal{A}',\mathcal{F}'}|| \le ||\Lambda_{\mathcal{A},\mathcal{F}} - \mathcal{CP}|| + 2 \cdot \text{access-cost}_{\mathcal{F}}(x) + 2 \cdot \text{update-cost}_{\mathcal{F}}(x)$$

Since $2 \cdot \text{access-cost}_{\mathcal{F}}(x) + 2 \cdot \text{update-cost}_{\mathcal{F}}(x) \le 2 \cdot \text{cost}_{\mathcal{F}}(x) \le 2 \cdot \text{cost}_{\mathcal{O}}(x)$, this implies Theorem 2.13. Deducting $\mathcal{CP}$ from Ψ can be seen as, for each $z \prec_{\mathcal{A}} x$, removing $\lhd_{\{z,x\}}$-coin from z. Let $\widehat{\Lambda}$ be the function after removing the coins, so $||\widehat{\Lambda}|| = ||\Lambda_{\mathcal{A},\mathcal{F}} - \mathcal{CP}||$.

Lemma 2.15. $||\Lambda_{\mathcal{A}',\mathcal{F}}|| \le ||\widehat{\Lambda}|| + 2 \cdot \text{access-cost}_{\mathcal{F}}(x)$

Proof. Recall that $\Lambda_{\mathcal{A}',\mathcal{F}} - \widehat{\Lambda} = \sum_{z,\alpha} (\Lambda_{\mathcal{A}',\mathcal{F}}(z,\alpha) - \widehat{\Lambda}(z,\alpha))$. It suffices to consider only $(z,\alpha) \in S$ where $S = \{(z,\alpha) \mid \Lambda_{\mathcal{A}',\mathcal{F}}(z,\alpha) > \widehat{\Lambda}(z,\alpha)\}$. When x is accessed, let $p(x)$ and $p'(x)$ be positions of x on list $\mathcal{A}$ and $\mathcal{A}'$, respectively. See Figure 2.7. We divide into three cases:

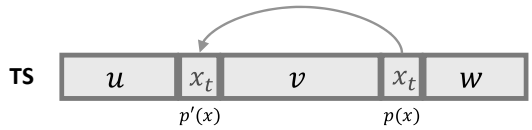


Figure 2.7. TS updates by moving x from position $p(x_t)$ to $p'(x_t)$

1. Let $S_1 = S \cap \{(z,\alpha) \mid \Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) > 0 \text{ and } \Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) > \widehat{\Lambda}(z,\alpha)\}$. We will show that $\sum_{(z,\alpha)\in S_1}(\Lambda_{\mathcal{A}',\mathcal{F}}(z,\alpha) - \widehat{\Lambda}(z,\alpha)) \le \text{access-cost}_{\mathcal{F}}(x)$.

   By $\Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) > \widehat{\Lambda}(z,\alpha)$, it means that one $\lhd_{\{z,x\}}$-coin are deducted from each $z \in J_{\mathcal{A}}(x)$. For those z that $\lhd_{\{z,x\}} = z$, it means $z \prec_{\mathcal{F}} x$. There are at most access-cost$_{\mathcal{F}}(x)$ such keys. Each key need one coin. For those z that $\lhd_{\{z,x\}} = x$, it means $x \prec_{\mathcal{F}} z$.

   - For each $z \prec_{\mathcal{A}'} p'(x)$, $\Lambda_{\mathcal{A},\mathcal{O}}(z,x) = 2$ and $\Lambda_{\mathcal{A}',\mathcal{O}}(z,x) = \widehat{\Lambda}(z,\alpha) = 1$ for each inversion. Since x is the last access compared to z, we do not need to add any coins.
   - For each $z \succ_{\mathcal{A}'} p'(x)$, there is no inversion.

2. Let $S_2 = S \cap \{(z,\alpha) \mid \Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) > 0 \text{ and } \Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) = \widehat{\Lambda}(z,\alpha)\}$. We will show that $\sum_{(z,\alpha)\in S_2}(\Lambda_{\mathcal{A}',\mathcal{F}}(z,\alpha) - \widehat{\Lambda}(z,\alpha)) \le \text{access-cost}_{\mathcal{F}}(x)$.

   - For each $z \prec_{\mathcal{A}'} p'(x)$, has inversion only if $z \succ_{\mathcal{F}} x$. This has already been considered in case S_1.
   - For each $z \succ_{\mathcal{A}'} p(x)$, it has inversion only if $z \prec_{\mathcal{F}} x$. We have that $\Lambda_{\mathcal{A},\mathcal{F}}(x,z) \ge 1$. So, $\Lambda_{\mathcal{A}',\mathcal{F}}(x,z) - \Lambda_{\mathcal{A},\mathcal{F}}(x,z) \le 1$ and we add at most one z-coin.

   Since all keys that need coins in S_2 are disjoint and in front of x in $\mathcal{F}$, we need at most access-cost$_{\mathcal{F}}(x)$ coins.

3. Let $S_3 = S \cap \{(z,\alpha) \mid \Lambda_{\mathcal{A},\mathcal{F}}(z,\alpha) = 0\}$. We will show that $\sum_{(z,\alpha)\in S_3}(\Lambda_{\mathcal{A}',\mathcal{F}}(z,\alpha) - \widehat{\Lambda}(z,\alpha)) \le 0$.

   New inversions only occur when $z \succ_{\mathcal{A}'} p'(x)$ but $z \prec_{\mathcal{A}'} p(x)$, and $z \prec_{\mathcal{F}} x$. We consider whether z or x is the more recent access *before* the current time. If z is more recent, then $\Lambda_{\mathcal{A}',\mathcal{F}}(z,x) = 1$. So, we need one x-coin. Otherwise, x is more recent, by Observation 2.14, we can assume that $x \prec_{\mathcal{F}} z$ and there is no inversion.

□

Lemma 2.16. $||\Lambda_{\mathcal{A}',\mathcal{F}'}|| \le ||\Lambda_{\mathcal{A}',\mathcal{F}}|| + 2 \cdot \text{update-cost}_{\mathcal{F}}(x)$

Proof. As in Lemma 2.12, we will only analyze our algorithm against the optimal that uses only paid moves. If $\mathcal{F}$ uses paid moves, notice that each move can only create at most one inversion. So, we need at most 2 coins.

□

2.6 Discussion

We propose a new viewpoint that assigns meaning to the values of potential functions. This viewpoint has two advantages: (1) It leads to a more illustrative analysis of various online list update algorithms, and (2) It is more suitable for the purpose of (relatively minor) modification of potential function, for instance, in this view, BIT potential function is quite immediate from MF potential function. However, our accounting method has a limitation in dealing with randomized algorithms. Notice that BIT only uses random bits at the beginning and becomes deterministic afterward. For instance, suppose we wish to analyze the list update algorithm such that, at each time, the accessed key move to the front with a probability of 0.5. If the accessed key fails to move to the front several times consecutively, our method may encounter difficulties as we may exhaust our coins to cover its cost. In the next chapter, we will demonstrate how to apply this technique to binary search trees.

3. Binary search trees

A binary search tree (BST) is one of the most fundamental data structures. The long-standing dynamic optimality conjecture states that there exists an online self-adjusting BST algorithm such that, starting with any initial tree, its access cost on any sequence X is $O(\mathsf{OPT}(X)+n)$ where $\mathsf{OPT}(X)$ is the cost of serving X on an offline optimal algorithm and n is the number of keys. Despite attempts from many groups of researchers, we believe the conjecture is still far from being proven, primarily due to the lack of systematic techniques on the analytical side of algorithms.

In this chapter, we first define a BST model and introduce well-known BST algorithms. We then introduce a geometric view of BST model, which gives us insight into how to define an inversion potential function in order to analyze the cost of various BST algorithms. The inversion potential function is a popular potential function for list updates but has never been used in a BST context. Arguably, this is the most natural potential function since it captures the difference between the states of two different algorithms. By extending this potential function, we are able to unify and strengthen known BST bounds. Moreover, we give a first analysis that can compare the costs of two dynamic binary search trees against each other directly and systematically. As a consequence, a broad class of BST algorithms has the simulation embedding property, one of the two property necessary for being dynamically optimal. This chapter is partly taken from [19].

3.1 Model

In this section, we define a model of BST that we use in this thesis.

BST is a pointer data structure in a pointer model that is used to handle the dictionary problem. BST is a rooted binary tree with the following rules.

(i) Each node store a unique key $z \in [n]$. (We called this node z)

(ii) For each node z, each key in the left subtree of z are less than z.

(iii) For each node z, each key in the right subtree of z are more than z.

(iv) When accessing x, the pointer starts at the root. Let p be the current node where the pointer is at. The pointer goes left if $x < p$, otherwise the pointer goes right. This process continues until x is found. (In this thesis, we only consider the case where the key is found.)

We denote the root of BST T by root(T). When access x arrives, the pointer p starts at the root, moves to the left child if the key that p is pointing at is higher than x, or moves to the right otherwise, until x is found. Nodes on the search path of x (including x), denoted by $\mathcal{S}(x)$, are said to be *touched* by T. The access cost of accessing x on BST T, denoted by $\text{cost}(x,T)$, is defined as the number of times we need to compare x. Equivalently, $\text{cost}(x,T) = depth_T(x) + 1$ where $depth_T(x)$ be a depth of node x in T.

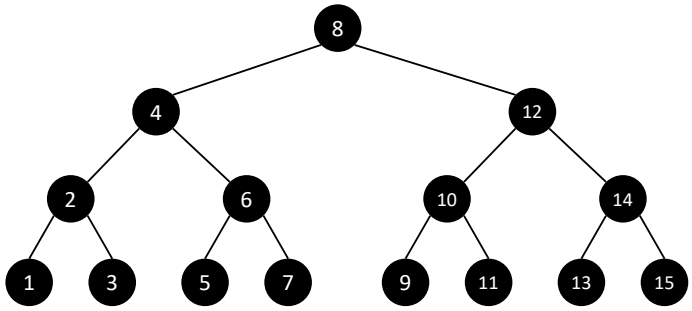


Figure 3.1. A complete balanced BST T with 15 nodes. For any node x, $depth_T(x) \leq \lceil \log_2 15 \rceil = 4$. $\mathcal{S}(5) = \{8,4,6,5\}$. $\text{cost}(5,T) = 4$.

3.1.1 Static trees

For a static tree, the BST does not change throughout the entire access sequence X.

Input:	$X = (x_1, ..., x_m) \in [n]^m$
Output:	A static BST T
Minimize:	$\sum_{i=1}^{m} \text{cost}(x_i, T)$

Throughout this section, we use n as the number of keys and m as the length of the access sequence X. We use notation $\text{cost}(X,T) = \sum_{i=1}^{m} \text{cost}(x_i,T)$. In a balanced BST, each node has depth at most $O(\log n)$.

Observation 3.1. *Given $X \in [n]^m$ and a balanced static BST T,* $\text{cost}(X,T) = O(m \log n)$.

With many insert and delete operations, a tree might become unbalanced. Some examples of classic algorithms that are used to re-balance the tree after insertion and deletion are AVL tree and Red-black tree [1][29].

3.1.2 Dynamic trees

In a dynamic model, after each access x_t, an algorithm is allowed to reshape

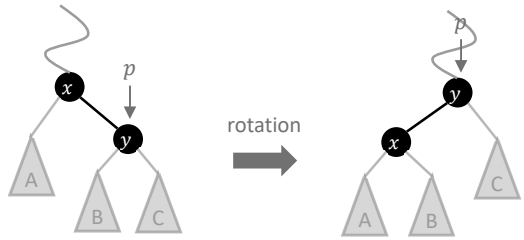


Figure 3.2. An example of a rotation operation

This operation is symmetrical when y is a left child of x. Equivalently, one can consider the following BST model:

Input:	$X = (x_1, ..., x_m) \in [n]^m$
Output:	(i) An initial BST T_0
	(ii) $Q = (Q_1, ..., Q_m)$
	where Q_i is a subtree of T_{i-1} that includes x_i and root(T_{i-1})
Minimize:	$\sum_{i=1}^{m} \text{cost}(x_i, T_{i-1}) + \sum_{i=1}^{m} \lvert Q_i \rvert$
	where T_i is obtained by replacing Q_i on subtree of T_{i-1} that have the same set of keys

See Figure 3.3 for an example of a sequence that dynamic tree. The cost of reshaping the tree $\sum_{i=1}^{m} |Q_i|$ is called update cost. Note that updates cannot change the elements in the tree. In the example, the total cost of the dynamic tree is at most $3n$, while every static tree needs $\Omega(n \log n)$ for this access sequence.

Remark that even an arbitrary initial tree T_0 only affects the total cost by at most an additive factor of n. Throughout the thesis, we assume that

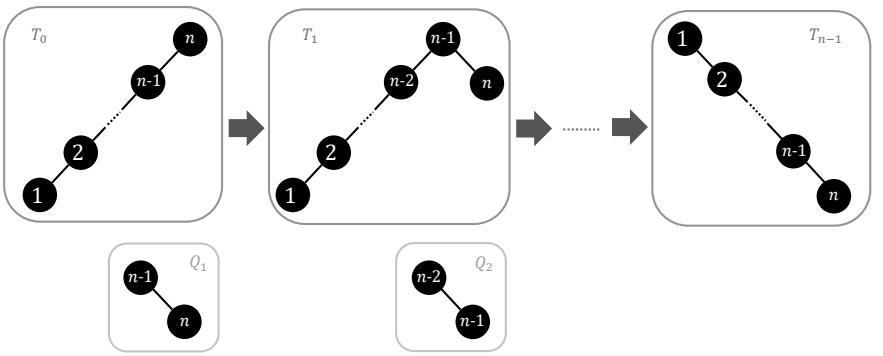


Figure 3.3. An example that shows when a dynamic tree is better than static one. Let $X = (n, n-1, ..., 1)$. This dynamic tree cost $O(n)$, but every static tree cost $\Omega(n \log n)$.

$m > n$. Hence we omit the reference to initial trees for brevity.

For a dynamic tree algorithm $\mathcal{A}$, since $\mathcal{A}$ defines Q, we use notation $\text{cost}_{\mathcal{A}}(x_i) = \text{cost}(x_i, T_{i-1}) + |Q_i|$ and $\text{cost}_{\mathcal{A}}(X) = \sum_{i=1}^{m} \text{cost}_{\mathcal{A}}(x_i)$. In this thesis, we only consider BST algorithms that only change the search path ($\forall i \in [m], Q_i \subseteq \mathcal{S}(x_i)$). So, the total cost is within a constant factor of access cost: $\text{cost}_{\mathcal{A}}(X) = \sum_{i=1}^{m} \text{cost}_{\mathcal{A}}(x_i) = \sum_{i=1}^{m} \text{cost}(x_i, T_{i-1}) + \sum_{i=1}^{m} |Q_i| \leq 2 \sum_{i=1}^{m} \text{cost}(x_i, T_{i-1})$.

For convenience, we will abuse notation and use $\mathcal{A}$ to stand for both the BST algorithm and the state of BST at a certain time.

Given $X \in [n]^m$, we denote $\mathsf{OPT}(X) = \min_{\mathcal{A}} \text{cost}_{\mathcal{A}}(X)$ as cost of an offline optimal algorithm for X. We also use the notation OPT as an offline optimal algorithm.

Definition 3.2 (Dynamic Optimality [48])**.** A BST algorithm $\mathcal{A}$ is *dynamically optimal* if, for all $X \in [n]^m$, $\text{cost}_{\mathcal{A}}(X) = O(\mathsf{OPT}(X) + f(n))$.[1]

The two binary search tree algorithms called *Splay* and *Greedy* are the two most-studied candidates to be dynamically optimal. We will define these in the later section/chapter.

We mentioned earlier that a static balanced tree cost $O(m \log n)$ for all $X \in [n]^m$. The following theorem shows that this bound is tight even for dynamic trees.

Theorem 3.3. *There are sequences $X \in [n]^m$ with $\mathsf{OPT}(X) = \Omega(m \log n)$.*

In fact, [10] shows that this lower bound holds for most sequences $X \in [n]^m$. One such sequence is called *bitwise reversal* [51].

[1] In the original conjecture, the additive term is linear. However, it is quite common in the literature (including this thesis) to relax an additive term as a function of n.

3.1.3 Properties

In this subsection, we give some important bounds that have been studied in the BST literature. For more detail, we refer to [16].

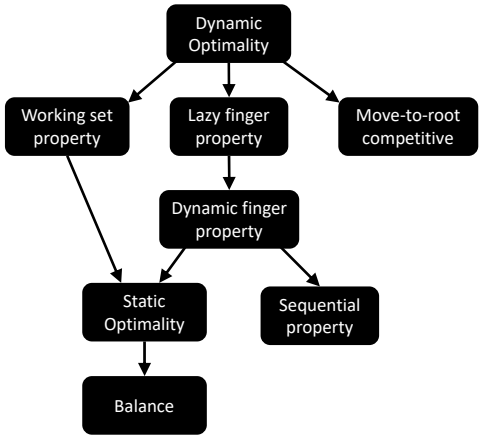


Figure 3.4. The relation between BST bounds.

Definition 3.4 (Balanced)**.** A BST algorithm $\mathcal{A}$ is *balanced* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) = O(m \log n)$.

Definition 3.5 (Static optimality)**.** A BST algorithm $\mathcal{A}$ is *statically optimal* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) = O(\min_R \mathrm{cost}(X,R))$.

Given X, a static tree with minimum cost according to X can be found in polynomial time [34, 38].

Definition 3.6 (Working set property [48])**.** Given $X = (x_1,...,x_m) \in [n]^m$, let $t_X(i)$ be the last access time of x_i in X before time i, i.e., $t_X(i) = \max\{j < i \mid x_j = x_i\}$ ($t_X(i) = 0$ if x_i is accessed for the first time). Define $w_X(i)$ as a number of distinct accessed keys since time $t_X(x)$, i.e., $w_X(i) = \{x_j \mid t_X(i) < j \le i\}$. We denote the working set bound $\mathsf{WS}(X) := \sum_{i=1}^{m} \log |w_X(i)|$. We say that an algorithm $\mathcal{A}$ has the *working set property* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) = O(\mathsf{WS}(X) + f(n))$.

Definition 3.7 (Sequential property)**.** A BST algorithm $\mathcal{A}$ has *sequential property* if, for $X = (1,...,n)$, $\mathrm{cost}_{\mathcal{A}}(X) = O(n)$.

Definition 3.8 (Dynamic finger property [48])**.** We denote a dynamic finger bound for a given sequence $X = (x_1,...,x_m) \in [n]^m$ as $\mathsf{DF}(X) := \sum_{i=2}^{m} \log(|x_i - x_{i-1}| + 1)$. A BST algorithm $\mathcal{A}$ has *dynamic finger property* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) = O(\mathsf{DF}(X) + f(n))$.

While $\mathsf{WS}(X)$ measures the distance of X in time, $\mathsf{DF}(X)$ measures the distance to X in space (between keys). Both Greedy and Splay are statically optimal and have the working set property. This follows from the access lemma [48]. Also, it is known that they satisfy the dynamic finger property [23, 22, 48]. However, only Greedy is known to have the lazy finger property [32].

Definition 3.9 (Lazy finger property [12])**.** For a sequence $X = (x_1, \ldots, x_m) \in [n]^m$ and a fixed BST R, let x_0 be a root of R and $\mathsf{LF}_R(X) = \sum_{i=1}^{m} d_R(x_{i-1}, x_i)$ where $d_R(a,b)$ denotes the number of edges on the unique path in R from a to b. We denote the lazy finger bound $\mathsf{LF}(X) := \min_R \mathsf{LF}_R(X)$. We say that an algorithm $\mathcal{A}$ has the *lazy finger property* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) = O(\mathsf{LF}(X) + f(n))$.

For more details on BST bounds, we refer to [16].

3.2 Move-to-root algorithm

In this section, we discuss one of the simplest online BST algorithms called Move-to-root. Move-to-root was introduced in [5] and was shown that it is statically optimal in expectation if the accesses are drawn independently at random from the frequency distribution. In the later section/chapter, we will compare Move-to-root with Splay and Greedy.

3.2.1 Definition

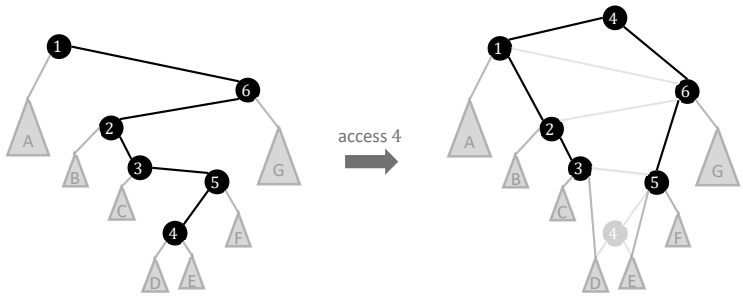


Figure 3.5. An example of Move-to-root algorithm

Definition 3.10 (MR-Competiveness)**.** A BST algorithm $\mathcal{A}$ is MR-*competitive* if, for all $X \in [n]^m$, $\mathrm{cost}_{\mathcal{A}}(X) \le O(\mathrm{cost}_{\mathsf{MR}}(X) + f(n))$.

3.2.2 Properties

In this subsection, we will show that, while MR is known not to be dynamically optimal, MR-competitive is not subsumed by other properties.

Proposition 3.11. *Let an initial tree be a path where 1 is the root and $i+1$ is the right child of i for all $i \in [n-1]$. Let $X = (n, n-1, ..., 1)$ be an access sequence. Then,* $\mathrm{cost}_{\mathsf{MR}}(X) = \Omega(n^2)$.

Proof. After key i is accessed, it becomes a new root and key $i-1$ becomes a leaf. Since all keys less than $i-1$ are ancestors of $i-1$, this gives the cost of at least $i-1$ when $i-1$ is accessed. So, the total cost is at least $n + (n-1) + ... + 1 = \Omega(n^2)$. □

Since X is a permutation sequence, $\mathsf{OPT}(X) = O(n \log n)$ by a static balanced tree. Hence, MR is not dynamically optimal. Next, we will show that MR-competitive is not subsumed by the working set bound and the lazy finger bound.

Consider a set of points in X in a plane $\mathbb{R}^2$, i.e. a point set $\{(x_t, t)\}_{t \in [m]}$. For $\mathbf{p},\mathbf{q} \in \mathbb{R}^2$, define $\square_{\mathbf{p},\mathbf{q}}$ as the set of integral points in the minimally closed rectangular area defined by $\mathbf{p}$ and $\mathbf{q}$. We say that such the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is *empty* if $\square_{\mathbf{p},\mathbf{q}}$ contains no other point than $\mathbf{p}$ or $\mathbf{q}$. Let $\gamma(a,t)$ denote the last access time of a before time t. Let $Z_t := \{a \mid \square_{(a,\gamma(a,t)),(x_t,t)} \text{ is empty in } X_{\le t}\}$ (See Figure 3.6).

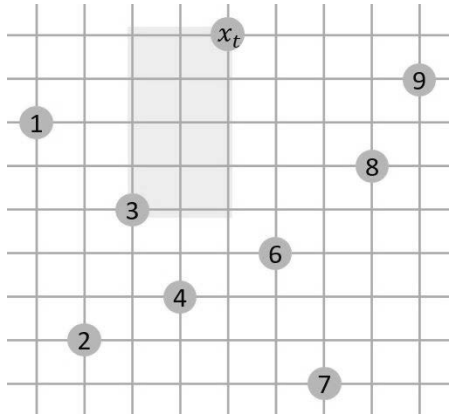


Figure 3.6. Each point is an access point where the vertical axis is time and the horizontal axis is keys. $Z_t = \{1,3,4,6,8,9\}$

Lemma 3.12. *For* MR*, when x_t is accessed, a is touched if and only if $a \in Z_t$.*

Proof. ($\Leftarrow$) Assume without loss of generality that $a < x_t$. After time $\gamma(a,t)$, node x_t is in the subtree rooted at a. $\square_{(a,\gamma(a,t)),(x_t,t)}$ is empty in $X_{\le t}$ implies that there is no key in $[a, x_t]$ is accessed after time $\gamma(a,t)$. This means x_t remains being a descendant of a until time t. So, a is touched at time t.

($\Rightarrow$) We will prove by contraposition. If $a \notin Z_t$, this means there is a key $b \in (a, x_t]$ that is accessed after time $\gamma(a,t)$. Right after b is accessed and the tree is updated, either (1) a and x_t are in subtrees of different children of b (if $b \neq x_t$) or (2) x_t becomes an ancestor of a (if $b = x_t$). Key a can never be an ancestor of x_t again unless a is accessed. So, a will not be touched at time t. □

Corollary 3.13. $\text{cost}_{\mathsf{MR}}(X) = \sum_{t\in[m]} \text{cost}_{\mathsf{MR}}(x_t) = \sum_{t\in[m]}(|Z_t|+1)$.

The key message from Corollary 3.13 is that we can easily infer $\text{cost}_{\mathsf{MR}}(x_t)$ by looking at the geometry of X.

Proposition 3.14. *There exists a sequence $X \in [n]^m$ such that $\mathsf{LF}(X) = \Omega(m \log n)$, $\mathsf{WS}(X) = \Omega(m \log n)$, but $\text{cost}_{\mathsf{MR}}(X) = O(m)$.*

Proof. Let Y be a *tilted 2-by-$\frac{n}{2}$ grid* search sequence, i.e.

$$Y = 1, \frac{n}{2}+1, 2, \frac{n}{2}+2, \ldots, \frac{n}{2}-1, n-1.$$

In [17], it is shown that $\mathsf{LF}(Y) = \Omega(n \log n)$. Let Z be the reversed sequence of Y, i.e.

$$Z = n-1, \frac{n}{2}-1, n-2, \frac{n}{2}-2, \ldots, \frac{n}{2}+1, 1.$$

Let $Y \| Z$ be a concatenation of Y and Z. Let X be $Y \| Z$ with $\frac{n}{2}$ inserted between each access (see Figure 3.7). Note that $|X| = \Theta(n)$. Notice that $\mathsf{LF}_{\mathcal{O}}(X) \geq \mathsf{LF}_{\mathcal{O}}(Y)$ for every fixed tree $\mathcal{O}$ because Y is a sub-sequence of X. Hence, we have that $\mathsf{LF}(X) \geq \mathsf{LF}(Y) = \Omega(n \log n)$.

Consider the working set bound, $\mathsf{WS}(X)$. Let ρ_t be the number of distinct keys that were accessed since the last access of x_t (if x_t is the first access of such key, then $\rho_t = 0$).

$$\begin{aligned}
\mathsf{WS}(X) &:= \sum_t \log(1+\rho_t)) \\
&\geq \begin{cases} \sum_t \log(1+\rho_t)) & \text{if } x_t \in \{1,2,\ldots,\frac{n}{2}-1\} \\ 0 & \text{otherwise} \end{cases} \\
&\geq \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \log(1+n-2i) \\
&= \Omega(\log(\frac{n}{2}!)) \\
&= \Omega(n \log n)
\end{aligned}$$

For MR, it is easy to verify from Figure 3.7 that each access touches at

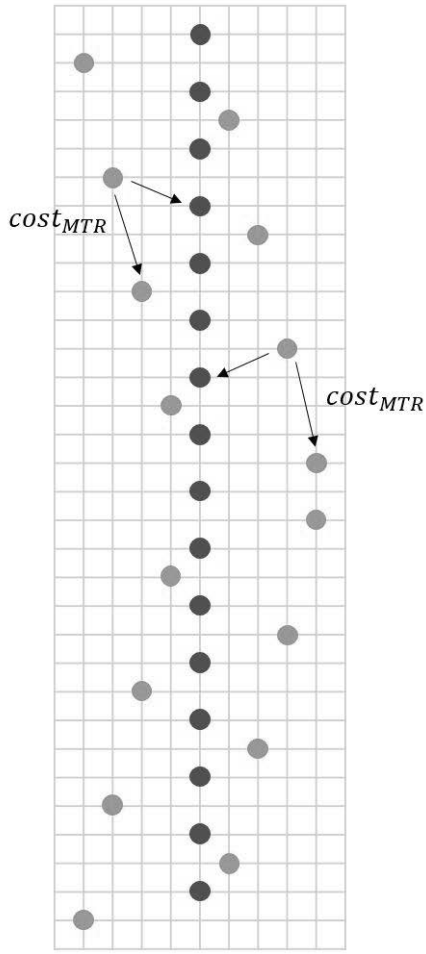


Figure 3.7. Sequence X

most 3 nodes (informally: itself, a node in the middle line, and a node in the tilted grid). Hence, $\text{cost}_{\mathsf{MR}}(X) = O(n)$. □

As a by-product from above, Move-to-root could help us find the upper bound of $\mathsf{OPT}(X)$. This is beneficial as it is usually difficult to upper bound $\mathsf{OPT}(X)$ because we do not know how OPT operates. The example above shows us that the optimal cost for the tilted grid sequence is linear.

Corollary 3.15. *Define X and Y sequences as above. Then,* $\mathsf{OPT}(Y) \leq \mathsf{OPT}(X) \leq O(n)$.

3.3 Interval geometry & potential functions

In this section, propose a new potential function, that we call *extended (geometric) inversion*. Inversion is arguably the most natural potential

function principle that has been used in competitive analysis but has never been used in the context of BSTs. We use our potential function to derive new results, as well as streamline/strengthen existing results.

3.3.1 Interval geometry

Our potential function is defined based on geometry that requires the correct drawing of BSTs. When drawing a BST, one should always use two rules: (1) imagine placing a node containing key z on the plane at point p where $p.x = z$ (the x-coordinate is z). (2) If u is a parent of v, always draw u higher than v (See Figure 3.8)

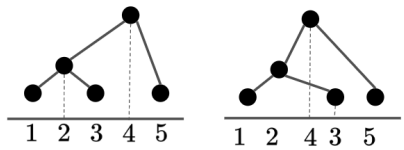


Figure 3.8. A drawing of a BST that complies with the rules (left) and does not comply with the rules (right).

Let $\mathcal{A}$ be a BST. For each key $z \in [n]$, let $I_{\mathcal{A}}(z)$ be the largest open interval in $(0, n+1)$ containing only all keys in the subtree of $\mathcal{A}$ rooted at z. More formally, $I_{\mathcal{A}}(z) = (\mathsf{left}_{\mathcal{A}}(z), \mathsf{right}_{\mathcal{A}}(z))$ where $\mathsf{left}_{\mathcal{A}}(z)$ is the nearest left ancestor of z (0 if not exist) and $\mathsf{right}_{\mathcal{A}}(z)$ is the nearest right ancestor of z ($n+1$ if not exist). When drawing the BST correctly, we have that the intervals $\{I_{\mathcal{A}}(z)\}_{z\in[n]}$ form a laminar family (that is, each pair of intervals is either disjoint or nested); see Figure 3.9.

Observation 3.16. *The following statements are true:*

(i) If z' is a left child of z, then $I_{\mathcal{A}}(z') = (\mathsf{left}_{\mathcal{A}}(z'), \mathsf{right}_{\mathcal{A}}(z')) = (\mathsf{left}_{\mathcal{A}}(z), z)$.

(ii) If z' is a right child of z, then $I_{\mathcal{A}}(z') = (\mathsf{left}_{\mathcal{A}}(z'), \mathsf{right}_{\mathcal{A}}(z')) = (z, \mathsf{right}_{\mathcal{A}}(z))$.

(iii) For any $z, z' \in [n]$, we have $z \in I_{\mathcal{A}}(z')$ iff z is in the subtree rooted at z'.

(iv) When accessing x, $z \in \mathcal{S}_{\mathcal{A}}(x)$ iff $x \in I_{\mathcal{A}}(z)$.

(v) In dynamic tree, let $\mathcal{A}'$ be a BST obtained by replacing $\mathcal{S}_{\mathcal{A}}(x)$ in $\mathcal{A}$ with a subtree of the same set of keys. Then, $\forall z \notin \mathcal{S}_{\mathcal{A}}(x)$, $I_{\mathcal{A}'}(z) = I_{\mathcal{A}}(z)$ (because the nearest left / right ancestor of z does not change).

Let $\mathcal{A}$ be an algorithm we want to analyze and $\mathcal{O}$ be an optimal algorithm. In this thesis, we only consider $\mathcal{A}$ that changes only the search path (this includes Greedy and Splay). Both $\mathcal{A}$ and $\mathcal{O}$ store the keys in $[n]$. We want to have a potential function that captures the "difference" between $\mathcal{A}$ and $\mathcal{O}$. The most natural scheme (which does not always work) often used in the context of online algorithms for this purpose is *inversion*.

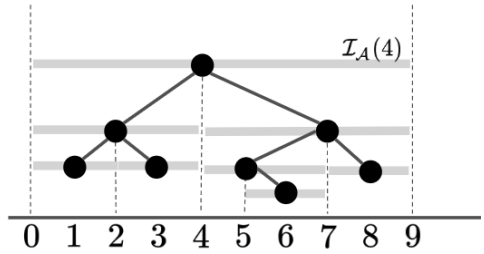


Figure 3.9. Example of intervals (in grey) for all keys. In this figure, $I_{\mathcal{A}}(4) = (0,9)$ and $I_{\mathcal{A}}(5) = (4,7)$

Definition 3.17 (Inversion)**.** Let $z, \alpha \in [n]$. We say that z forms an *inversion* with α if $z \in I_{\mathcal{O}}(\alpha)$ and $\alpha \in I_{\mathcal{A}}(z)$.

Notice that an inversion is an indicator of α being in the subtree of z in one BST, but z is in the subtree of α in the other. See Figure 3.10. We note that this is the same definition of inversion as in the List Update model. Unfortunately, we are unable to use this natural and intuitive scheme to prove any meaningful result. We will use *extended inversion* instead.

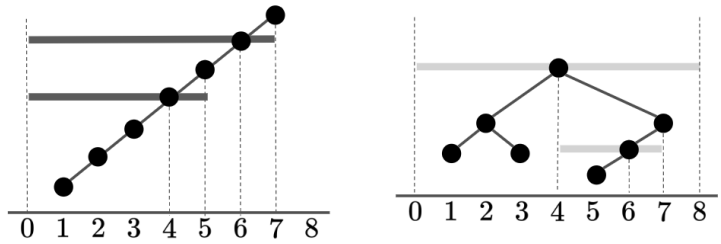


Figure 3.10. An example of inversion between the keys 4 and 6. The intervals of $\mathcal{A}$ are shown in dark grey and intervals of $\mathcal{O}$ in light grey. The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively.

For each interval in BST $\mathcal{O}$, $I_{\mathcal{O}}(\alpha)$, we define three **important points** of α in $\mathcal{O}$ as $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{O}}(\alpha), \mathsf{right}_{\mathcal{O}}(\alpha)\}$.

Definition 3.18 (Extended Inversion)**.** Let $z, \alpha \in [n]$. We say that z forms an *extended inversion* with α if and only if $z \in I_{\mathcal{O}}(\alpha)$ and $\exists \beta \in \mathcal{P}_{\mathcal{O}}(\alpha)(\beta \in I_{\mathcal{A}}(z))$. See Figure 3.11.

Unlike inversion, the notion of extended inversion is not symmetric. Clearly, if z forms an inversion with α, then z also forms an extended inversion with α.

3.3.2 Potential functions

All proofs in this paper build upon the following base function.

Definition 3.19 (Base Potential function)**.** Define the potential $\Phi = \Phi_{\mathcal{A},\mathcal{O}}$ at each state of execution of our algorithm $\mathcal{A}$ and optimal $\mathcal{O}$ as follows. Let

Figure 3.11. The darker and the lighter intervals are $I_{\mathcal{A}}(z)$ and $I_{\mathcal{O}}(z)$, respectively. In the first figure from left, z forms an inversion with α. In the second, z forms an extended inversion with α but not an inversion. In the third figure, there is no extended inversion, since $I_{\mathcal{A}}(z)$ does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. In the fourth figure, there is no extended inversion because $z \notin I_{\mathcal{O}}(\alpha)$.

$\Phi(z,\alpha) = 1$ if z forms an extended inversion with α; otherwise, $\Phi(z,\alpha) = 0$. The potential is defined as $||\Phi|| = \sum_{z\in[n]} \sum_{\alpha\in[n]} \Phi(z,\alpha)$. This is the total number of extended inversions.

We "visualize" our potential function value as a collection of "coins". Whenever $\Phi(z,\alpha) = 1$, one can imagine there is a coin of label z (or z-coin) placed at node α in $\mathcal{O}$ whenever z forms an extended inversion with α (see Figure 3.12). By definition, z always forms an extended inversion with itself, so there is always a z-coin at z. We use $\Phi(\bullet,\alpha) = \sum_{z\in[n]} \Phi(z,\alpha)$ to denote the total number of coins at α, and $\Phi(z,\bullet) = \sum_{\alpha\in[n]} \Phi(z,\alpha)$ the total number of z-coins. The coin interpretation will be used crucially in our analysis.

The main properties we would need are the following:

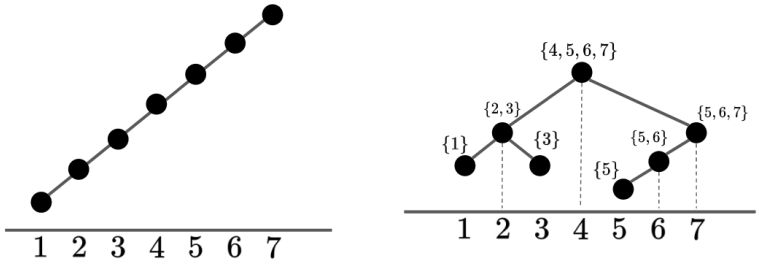


Figure 3.12. The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively. The set notation shown at each node is a collection of coins placed at that node. For instance, there are 5- and 6-coins at node 6. The path of 3-coins (according to Lemma 3.20) is a path containing nodes 3 and 2.

Lemma 3.20. *[Upward path property] For each $z \in [n]$, the set of nodes having z-coin (that is, $Q_z = \{\alpha : \Phi(z,\alpha) = 1\}$) is a contiguous subpath of the path from z to the root of $\mathcal{O}$.*

Proof. Let $\alpha_1, \alpha_2 \in Q_z$. Since $z \in I_{\mathcal{O}}(\alpha_1) \cap I_{\mathcal{O}}(\alpha_2)$, we must have that $I_{\mathcal{O}}(\alpha_1) \subseteq I_{\mathcal{O}}(\alpha_2)$ (from laminar property). Since $z \in Q_z$, we have that Q_z is a subset of the path from z to the root. Next, we argue that Q_z is connected. Suppose α' is on the path from z to the root such that $\alpha' \notin Q_z$. See Figure 3.13. We argue that the parent of α' (say α, where α' is the right child of α) is also not in Q_z: From Observation 3.16, $I_{\mathcal{O}}(\alpha') = (\alpha, \mathsf{right}_{\mathcal{A}}(\alpha))$. Recall that $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{A}}(\alpha), \mathsf{right}_{\mathcal{A}}(\alpha)\}$. Since $\alpha' \notin Q_z$, we have that $I_{\mathcal{A}}(z)$ is completely

contained in $(\alpha, \text{right}_{\mathcal{O}}(\alpha))$. This means that it does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. □

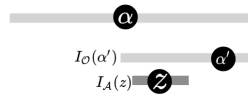


Figure 3.13. Illustration of the proof of Lemma 3.20.

By the above lemma, we view the potential function $\Phi(z, \bullet)$ as the coins on an upward path in $\mathcal{O}$ and Φ as a collection of upward paths.

Lemma 3.21. *Consider an access key $x \in [n]$. Let z be a key on the search path $\mathcal{S}_{\mathcal{A}}(x)$ of our algorithm. Then z forms an extended inversion with $LCA_{\mathcal{O}}(x,z)$. Hence, there is a z-coin at node $LCA_{\mathcal{O}}(x,z)$.*

Proof. Fix $z \in \mathcal{S}_{\mathcal{A}}(x)$. Recall that $x \in I_{\mathcal{A}}(z)$. Denote by $\ell = LCA_{\mathcal{O}}(x,z)$. Since ℓ is between x and z, $\ell \in I_{\mathcal{A}}(z)$. Since ℓ is an ancestor of z in $\mathcal{O}$, $z \in I_{\mathcal{O}}(\ell)$, and thus an inversion occurs between z and ℓ. □

The above lemma is the main advantage of the inversion principle: Each $z \in \mathcal{S}_{\mathcal{A}}(x)$ contributes +1 to the cost of $\mathcal{A}$ when accessing x. The lemma shows that such cost can be "deducted" from the z-coin at $LCA_{\mathcal{O}}(x,z)$.

Definition 3.22 (Canonical payment function)**.** Consider algorithm $\mathcal{A}$ and optimal $\mathcal{O}$. The canonical payment function for accessing x is a function $\mathcal{CP} = \mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ such that for each $z \in \mathcal{S}_{\mathcal{A}}(x)$, we have $\mathcal{CP}(z, LCA_{\mathcal{O}}(z,x)) = 1$. The function is 0 everywhere else (Figure 3.14).

We will simply write $\mathcal{CP}$ instead of $\mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ when it is clear from the context. We use $\mathcal{CP}$ as a payment function where coins that are used to pay for the accessed cost are specified. Notice that, for each $z \in \mathcal{S}_{\mathcal{A}}(x), \mathcal{CP}(z,\alpha) = 1$ only if α is on the search path $\mathcal{S}_{\mathcal{O}}(x)$ of the optimal. From Lemma 3.21, we have that $\mathcal{CP}(z,\alpha) \le \Phi(z,\alpha)$ for all $z, \alpha \in [n]$.

Observation 3.23. $\sum_{z,\alpha} \mathcal{CP}(z,\alpha) = |\mathcal{S}_{\mathcal{A}}(x)|$

Proposition 3.24. *The potential change $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\hat{\Phi}||$ is upper bounded by the "change on search paths", that is,*

$$||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Phi}|| \le \sum_{z \in \mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha \in \mathcal{S}_{\mathcal{O}}(x)} \left(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha)\right)$$

Proof. We show two statements:

1. For each $z \notin \mathcal{S}_{\mathcal{A}}(x)$ and $\alpha \in [n]$, we have that $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \le \widehat{\Phi}(z,\alpha)$.

   For each $z \notin \mathcal{S}_{\mathcal{A}}(x)$, the intervals $I_{\mathcal{A}}(z) = I_{\mathcal{A}'}(z)$, so the number of extended inversions for them remains the same. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \le \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$.

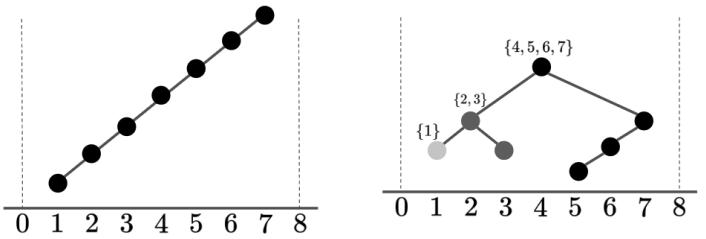


Figure 3.14. An example of coins in the support of $\mathcal{CP}$. In this example, when access 1, we touch every key in $\mathcal{A}$, but we touch only {1,2,4} in $\mathcal{O}$. Each $z \in \{4,5,6,7\}$ has its coin at node 4 in the support of $\mathcal{CP}$. Each z in {2,3} has its coin at node 2. Finally, 1 has a coin at node 1.

2. For each $z \in \mathcal{S}_{\mathcal{A}}(x)$ and $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$, we have that $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leq \widehat{\Phi}(z,\alpha)$.

   Fix $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$. Due to the previous observation, only $z \in \mathcal{S}_{\mathcal{A}}(x)$ may create a new inversion. By Lemma 3.21, $\Phi_{\mathcal{A},\mathcal{O}}(z, LCA_{\mathcal{O}}(x,z)) = 1$. Since $LCA_{\mathcal{O}}(x,z)$ is an ancestor of α, by Lemma 3.20, $\Phi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 1$. This means that there was already a z-coin at α. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leq \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$.

□

Remark. If we use inversion (without extended) potential function, Proposition 3.24 fails due to the fact, for $z \in \mathcal{S}_{\mathcal{A}}(x)$ and $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$, Lemma 3.21 and Lemma 3.20 do not imply that there was already a z-coin at α.

3.3.3 Move-to-root competitiveness

Recall our potential function framework. Let $\mathcal{A}$ be an algorithm and $\mathcal{O}$ be the MR algorithm. Suppose, after accessing key $x \in [n]$, the algorithm changes to $\mathcal{A}'$ and MR changes to $\mathcal{O}'$. We are interested in proving the upper bound

$$||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leq O(|\mathcal{S}_{\mathcal{O}}(x)|)$$

As outlined previously, it suffices to upper bound the following:

- The change of $\mathcal{A}$ to $\mathcal{A}'$ after subtracting the cost:
  $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leq O(|\mathcal{S}_{\mathcal{O}}(x)|)$
- And then the change of $\mathcal{O}$ to $\mathcal{O}'$:
  $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}|| \leq O(|\mathcal{S}_{\mathcal{O}}(x)|)$

The first item actually corresponds to the static optimality property of BSTs.

Definition 3.25. A BST algorithm $\mathcal{A}$ satisfies static optimality via potential function Φ if and only if, there exists a universal constant C such that, for every static tree $\mathcal{O}$ and access $x \in [n]$, we have $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leq C \cdot |\mathcal{S}_{\mathcal{O}}(x)|$.

Our potential function is particularly suitable for proving MR-competitiveness. In fact, to show that $\mathcal{A}$ is $O(1)$-competitive to MR, it suffices to only prove static optimality for $\mathcal{A}$ using extended inversions (together with a function that depends only on n.) Our analysis is the first where the costs of two dynamic binary search trees are compared against each other directly and systematically.

Lemma 3.26. *A BST algorithm that (1) satisfies static optimality via potential function Ψ that depends linearly on extended inversions (i.e. $||\Psi|| = c_1||\Phi|| + c_2 n$), and (2) moves the accessed key to the root, must be $O(1)$-competitive to* MR.

Proof. From Definition 3.25, if $\mathcal{A}$ satisfies static optimality via Ψ, we have that:

$$||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leq C \cdot |\mathcal{S}_{\mathcal{O}}(x)|$$

Now, since the second term of potential is $c_2 n$ (only depending on n), to upper bound $||\Psi_{\mathcal{A}',\mathcal{O}'}|| - ||\Psi_{\mathcal{A}',\mathcal{O}}||$, it suffices to show that, we have $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}|| \leq 0$.

For key $z = x$, $||\Phi_{\mathcal{A}',\mathcal{O}}(z,x)|| \geq ||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = c_1$ because $|\mathcal{S}_{\mathcal{O}'}(x)| = 1$. Now, we consider keys $z \neq x$. Observe that intervals in $\mathcal{O}'$ change as follows:

(i) $I_{\mathcal{O}'}(x) = (0, n+1)$.

(ii) $I_{\mathcal{O}'}(y) = (\mathsf{left}_{\mathcal{O}}(y), x)$ for all y such that $y < x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.

(iii) $I_{\mathcal{O}'}(y) = (x, \mathsf{right}_{\mathcal{O}}(y))$ for all y such that $y > x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.

(iv) $I_{\mathcal{O}'}(y) = I_{\mathcal{O}}(y)$ for all $y \notin S_{\mathcal{O}}(x)$.

In other words, the only new important point in $\mathcal{O}'$ is x. Since $\mathcal{A}'$ also has x as the root, no intervals $I'_{\mathcal{A}}(z)$ can contain x. Since x is the only new ancestor for z, this means $||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = 0$. □

3.4 Splay algorithm

Splay trees were introduced in [48]. It has been conjectured to be dynamically optimal. The following observation about BSTs is crucial for the proof in this section.

Observation 3.27. *For every three keys $z_1 < z_2 < z_3$ on a BST T, either*

- $LCA_T(z_1, z_3) = LCA_T(z_1, z_2)$ *or*
- $LCA_T(z_1, z_3) = LCA_T(z_2, z_3)$.

3.4.1 Definition

Splay trees do as follows: when x_i is accessed, p is pointing at x_i, we recursively perform the following operations until x_i becomes a new root.

- *if x_i has no grandparent, use zig operation*
- *if x_i's parent and grandparent are on the same side of x_i, use zig-zig operation*
- *if x_i's parent and grandparent are on the different side of x_i, use zig-zag operation*

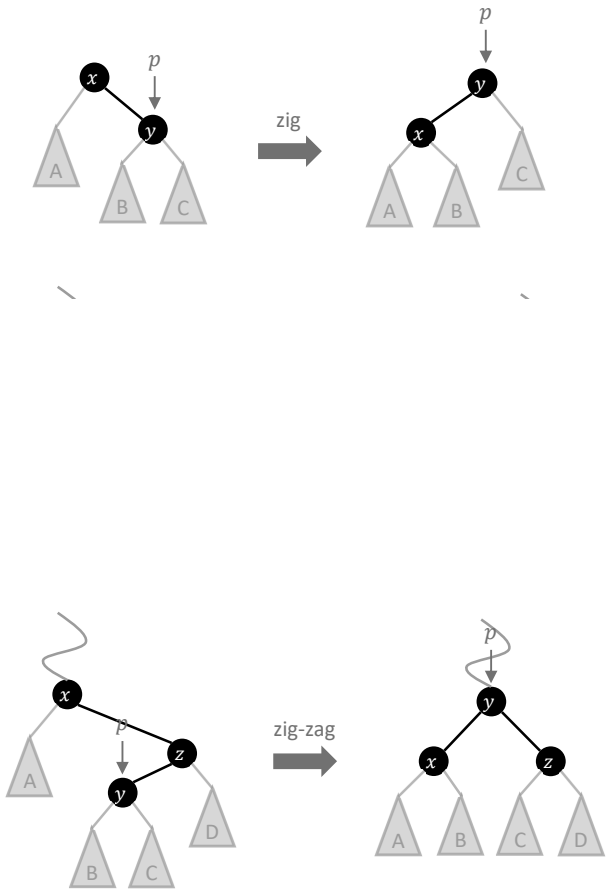


Figure 3.15. zig, zig-zig, and zig-zag operations

See Figure 3.16.

Observation 3.28. *Let Z be a set of touched keys that belong to zig-zig pairs (excluding x). let $Z_1 = \{z \mid I_{\mathcal{A}'}(z) \subset I_{\mathcal{A}}(z)\}$ and $Z_2 = Z \setminus Z_1$.*

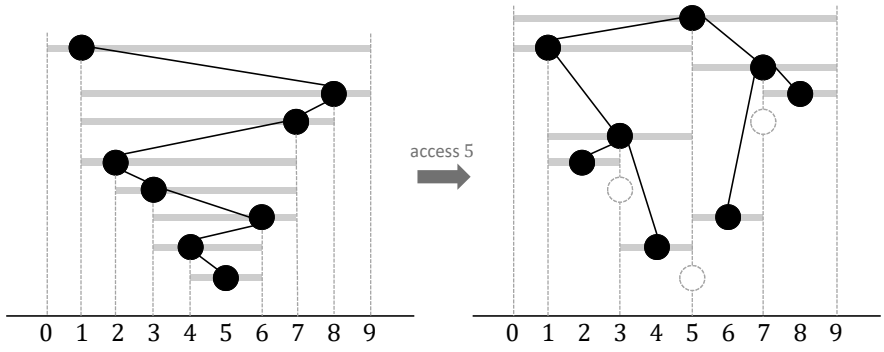


Figure 3.16. An example of Splay

- $\{I_{\mathcal{A}'}(z)\}_{z\in Z_1}$ *are disjoint. See Figure 3.17 (blue).*
- $\{I_{\mathcal{A}'}(z) \setminus I_{\mathcal{A}}(z)\}_{z\in Z_2}$ *are disjoint. See Figure 3.17 (red).*

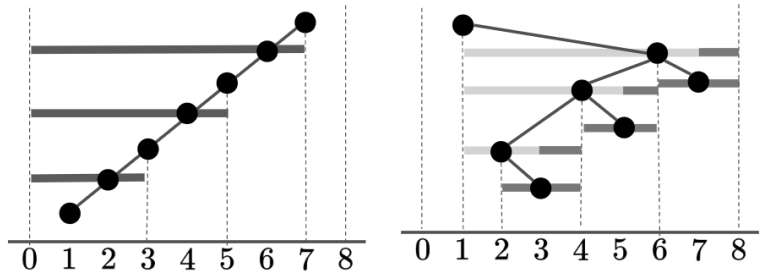


Figure 3.17. A BST before (left) and after (right) splaying $x = 1$.

3.4.2 Static optimality

The main purpose of this subsection is to show that Splay is statically optimal. Consider algorithm $\mathcal{A}$ that, after access x arrives, updates the trees into $\mathcal{A}'$, while $\mathcal{O}$ does not change.

Define the potential $\Psi(z,\alpha) = 2\Phi(z,\alpha)$ for all $z,\alpha \in [n]$ (there are 2 coins for each extended inversion). For convenience, we use notation $\Psi = \Psi_{\mathcal{A},\mathcal{O}}$ and $\Psi' = \Psi_{\mathcal{A}',\mathcal{O}}$.

Theorem 3.29. *Let $\mathcal{A}$ be the Splay tree. For each $X \in [n]^m$ and each static tree $\mathcal{O}$,* $\text{cost}_{\mathcal{A}}(X) \le 16 \cdot \text{cost}_{\mathcal{O}}(X) + m + n^2$.

We will show that for every static tree $\mathcal{O}$, we have $||\Psi'|| \le ||\Psi - \mathcal{CP}|| + 16|\mathcal{S}_{\mathcal{O}}(x)| + 1$.

Deducting $\mathcal{CP}$ from Ψ can be seen as removing some coins from the search path $\mathcal{S}_{\mathcal{O}}(x)$. Recall that in $\Psi - \mathcal{CP}$, for each z on the search path of $\mathcal{A}$, we

had removed one z-coin at $LCA(z,x)$. After removing coins, we move coins around as follows:

- *(zig-zig)*: For each zig-zig pair $z_1, z_2 \in \mathcal{S}_{\mathcal{A}}(x)$, let z_1 be a key such that z_1 is a child of z_2 in $\mathcal{A}'$. We move one z_1-coin at $LCA_{\mathcal{O}}(z_1,x)$ to be a z_2-coin of $LCA_{\mathcal{O}}(z_2,x)$. See Figure 3.18.
- *(zig-zag)*: For each zig-zag pair $z_1, z_2 \in \mathcal{S}_{\mathcal{A}}(x)$, let z_1 be a key such that $LCA_{\mathcal{O}}(z_1,x) = LCA_{\mathcal{O}}(z_1,z_2)$.
  (a) If $\Psi(z_2, LCA_{\mathcal{O}}(z_1,z_2)) = 2$, this means we have two z_2-coins at $LCA_{\mathcal{O}}(z_1,z_2)$. We change one of them to z_1-coin. Then, we move

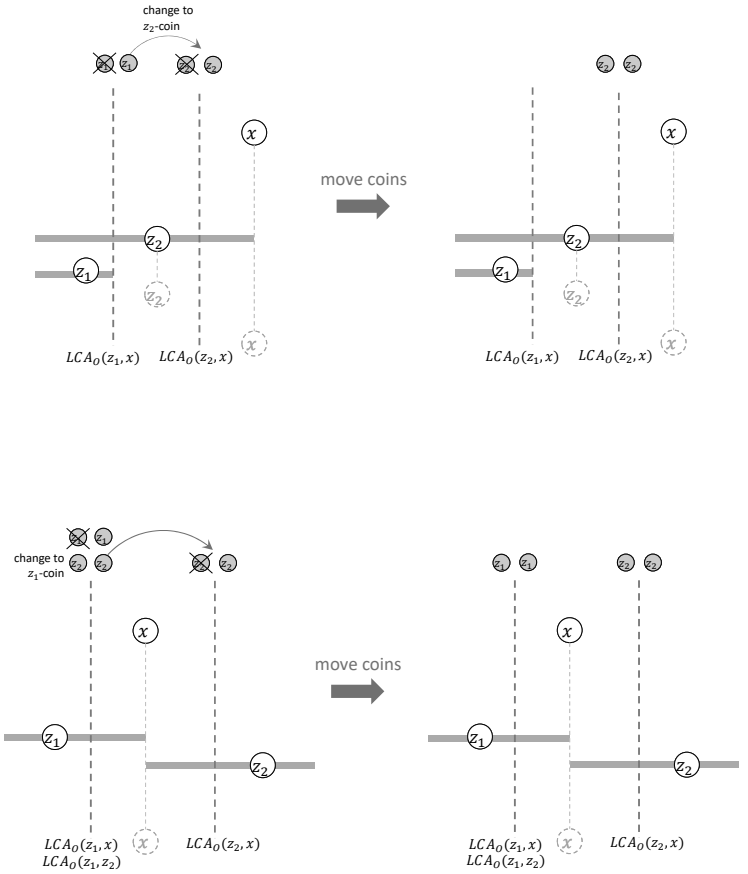


Figure 3.19. An illustration of proof in zig-zag case (a)

Let $\widehat{\Psi}$ be the function after moving the coins, so $||\widehat{\Psi}|| = ||\Psi - \mathcal{CP}||$.

Lemma 3.30. *We have* $||\Psi'|| - ||\widehat{\Psi}|| \leq O(|\mathcal{S}_{\mathcal{O}}(x)|)$. *(In other words, we can add at most* $O(|\mathcal{S}_{\mathcal{O}}(x)|)$ *to achieve the coin level of* Ψ'*.)*

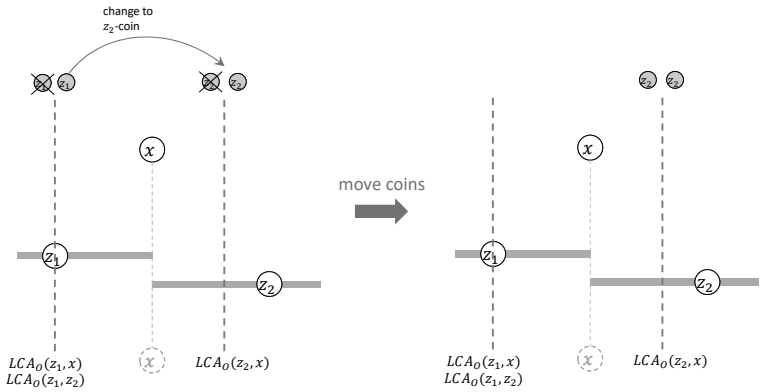


Figure 3.20. An illustration of proof in zig-zag case (b)

The rest of this section is devoted to proving the lemma. Notice that we only move coins of touched keys from those nodes in $\mathcal{S}_{\mathcal{O}}(x)$. So, Proposition 3.24 holds and we have $||\Psi'||-||\widehat{\Psi}|| \leq \sum_{z\in\mathcal{S}_{\mathcal{A}}(x)}\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)}\left(\Psi'(z,\alpha)-\widehat{\Psi}(z,\alpha)\right)$. The following claim, therefore, finishes the proof.

Lemma 3.31. *We have* $\sum_{z\in\mathcal{S}_{\mathcal{A}}(x)}\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)}\left(\Psi'(z,\alpha)-\widehat{\Psi}(z,\alpha)\right) \leq 16|\mathcal{S}_{\mathcal{O}}(x)|+1$.

Proof. We divide touched keys into 4 cases: zig-zig keys, zig-zag keys, zig key, and x itself.

(i) For x, we use the crude bound of $\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)}\left(\Psi'(x,\alpha)-\widehat{\Psi}(x,\alpha)\right) \leq 2|\mathcal{S}_{\mathcal{O}}(x)|$.

(ii) For the zig key z, since $I_{\mathcal{A}'}(z) \subset I_{\mathcal{A}}(z)$, there is no new extended inversion. Since we deducted one z-coin, we add only one z-coin. We have $\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)}\left(\Psi'(z,\alpha)-\widehat{\Psi}(z,\alpha)\right) \leq 1$.

(iii) For zig-zig key z, we divide the analysis into two cases:

(a) New extended inversions, i.e. $\Psi'(z,\alpha)=2$ and $\widehat{\Psi}(z,\alpha)=\Psi(z,\alpha)=0$. We upper bound the potential increase by the notion of *witness intervals*. For the first case, a new extended inversion can appear in $\Psi'(z,\alpha)$ due to the fact that the interval $I_{\mathcal{A}'}(z)$ becomes larger than $I_{\mathcal{A}}(z)$. In this case, we define $J(z)=I_{\mathcal{A}'}(z)\setminus I_{\mathcal{A}}(z)$ as a **witness** interval of z. From Observation 3.28, these witness intervals are disjoint. Since there are 3 important points for each $\alpha\in\mathcal{S}_{\mathcal{O}}(x)$ and each point can be in at most one witness interval, there are at most $3|\mathcal{S}_{\mathcal{O}}(x)|$ witness intervals. Each witness interval requires 2 coins, so we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total.

(b) Existing extended inversions that pay their coins out, i.e. $\Psi'(z,\alpha)=2, \widehat{\Psi}(z,\alpha)=0$ and $\Psi(z,\alpha)=2$.

For the second case, this happened due to z pays its coin out of canonical payment and give the other coin to its pair. In

this case, we say that $J(z) = I_{\mathcal{A}'}(z)$ is a witness interval. From Observation 3.28, the witness intervals of this type are disjoint. Therefore, with the same argument as in the new extended inversion case, we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total.

(iv) For zig-zag pair z_1, z_2. Let z_1 be a key such that $LCA_{\mathcal{O}}(z_1, x) = LCA_{\mathcal{O}}(z_1, z_2)$. Assume that $z_1 < x < z_2$ (the other case is symmetric). We divide the analysis into two cases:

(a) If $\Psi(z_2, LCA_{\mathcal{O}}(z_1, z_2)) = 2$, after moving coins, there is no z_2-coin on $LCA_{\mathcal{O}}(z_1, z_2)$. Since $\mathsf{left}_{\mathcal{A}'}(z_2) = x$, we have $\Psi'(z_2, LCA_{\mathcal{O}}(z_1, z_2)) = 0$. So, we do need to add any coin.

(b) Otherwise, $LCA_{\mathcal{O}}(z_1, x) \notin I_{\mathcal{A}}(z_2)$. Since $(z_1, x) \subseteq I_{\mathcal{A}}(z_2)$ and $LCA_{\mathcal{O}}(z_1, x) \in [z_1, x]$ (by definition of LCA), this means $LCA_{\mathcal{O}}(z_1, x) = z_1$. So, $z_1 \in \mathcal{S}_{\mathcal{O}}(x)$. After moving coins, there are no z_1-coins at z_1. Since there are at most $|\mathcal{S}_{\mathcal{O}}(x)|$ such z_1, we need at most $2|\mathcal{S}_{\mathcal{O}}(x)|$ coins.

□

From Lemma 3.26, we obtain the following corollary.

Corollary 3.32. *Splay is* MR*-competitive.*

3.5 Family

In this section, we show that a broad class of BST algorithms are $O(1)$-competitive to the Move-to-root algorithm. It is shown in [15] that this family of BST is statically optimal. Here, using the inversion potential function, we show that this family of BST is statically optimal and hence MR-competitive as a consequence.

3.5.1 Definition

We follow the definitions in [15]. Let $\mathcal{A}$ be a BST algorithm such that $\mathcal{A}$ rearranges only the search path of x, $\mathcal{S}_{\mathcal{A}}(x)$. Let $\mathcal{S}'_{\mathcal{A}}(x)$ denote the tree that is formed by all keys in $\mathcal{S}_{\mathcal{A}}(x)$ after $\mathcal{A}$ updates. For each key z, the left-depth of z is the number of left-going edges on the path from the root to z. Likewise for the right-depth of z. For a search path $\mathcal{S}_{\mathcal{A}}(x)$, #(crossing edges in $\mathcal{S}_{\mathcal{A}}(x)$) is the number of edges $\{z_1, z_2\}$ on $\mathcal{S}_{\mathcal{A}}(x)$ such that $z_1 < x < z_2$ or $z_2 < x < z_1$.

We say that the algorithm $\mathcal{A}$ is (μ, η_1, η_2)-good if, for an access x, $\mathcal{A}$ satisfies the following properties:

(i) Moves x to root.

(ii) #(leaves in $\mathcal{S}'_{\mathcal{A}}(x)$) + #(crossing edges in $\mathcal{S}_{\mathcal{A}}(x)$) $\geq \frac{|\mathcal{S}_{\mathcal{A}}(x)|}{\mu}$.

(iii) For each key $z < x$, left-depth of z in $\mathcal{S}'_{\mathcal{A}}(x) \le \eta_1$.
For each key $z > x$, right-depth of z in $\mathcal{S}'_{\mathcal{A}}(x) \le \eta_2$.

When all parameters μ, η_1, η_2 are constant, we say that $\mathcal{A}$ is *awesome*. Note that Splay trees satisfy this property with $\mu = 2$ (assuming no rotation) and $\eta_1 = \eta_2 = 2$.

Observation 3.33. *Let x be an access key and $z \in \mathcal{S}_{\mathcal{A}}(x)$. Let $J(z) = I_{\mathcal{A}'}(z) \setminus I_{\mathcal{A}}(z)$. The following statements are true:*

- *If $z < x$, then*
  - $\mathsf{right}_{\mathcal{A}}(z) > x$ *because $z \in \mathcal{S}_{\mathcal{A}}(x)$*
  - $\mathsf{right}_{\mathcal{A}'}(z) \le x$ *because x is the new root*
  - $J(z) \cap (z, \mathsf{right}_{\mathcal{A}}(z)) = \emptyset$ *directly follows*
- *If $z > x$, then*
  - $\mathsf{left}_{\mathcal{A}}(z) < x$ *because $z \in \mathcal{S}_{\mathcal{A}}(x)$*
  - $\mathsf{left}_{\mathcal{A}'}(z) \ge x$ *because x is the new root*
  - $J(z) \cap (\mathsf{left}_{\mathcal{A}}(z), z) = \emptyset$ *directly follows*

The intuition of the observation is that, after $\mathcal{A}$ updates, the interval of each touched key z does not stretch in the direction toward x (but possibly stretch in the direction away from x).

3.5.2 Static optimality

In this subsection, using our potential function, we show that every awesome algorithm satisfies static optimality. The main novelty here is the inversion-based proof technique, as a different proof was known before in [15] using a variation of the sum-of-logs potential function defined in [48].

For every (μ, η_1, η_2)-good BST, we use a potential function $||\Psi|| = 4\mu||\Phi||$ where Φ is an extended inversion potential (i.e. we have 4μ coins for each extended inversion). Again, we abbreviate $\Psi_{\mathcal{A},\mathcal{O}}$ and $\Psi_{\mathcal{A}',\mathcal{O}}$ (after accessing x, turning $\mathcal{A}$ into $\mathcal{A}'$) as Ψ and Ψ' respectively.

Theorem 3.34. *Let $\mathcal{A}$ be a (μ, η_1, η_2)-good BST algorithm. For each X and each static tree $\mathcal{O}$, $\mathrm{cost}_{\mathcal{A}}(X) \le 4\mu(2\eta_1 + 2\eta_2 + 6) \cdot \mathrm{cost}_{\mathcal{O}}(X) + n^2$.*

Corollary 3.35. *Let $\mathcal{A}$ be an awesome BST. Then $\mathcal{A}$ is statically optimal.*

The rest of this section is devoted to proving the theorem. We will show that for every static tree $\mathcal{O}$, we have

$$||\Psi'|| - ||\Psi - \mathcal{CP}|| \le 4\mu(2\eta_1 + 2\eta_2 + 6)|\mathcal{S}_{\mathcal{O}}(x)|.$$

There are two steps. In the first step, after removing one z-coin at $LCA(z, x)$ for each touched key z, we move coins around. Let $\widehat{\Psi}$ be a function after

moving the coins. Note that $||\widehat{\Psi}|| = ||\Psi - \mathcal{CP}||$. In the second step, we upper bound $||\Psi'|| - ||\widehat{\Psi}|| = ||\Psi'|| - ||\Psi - \mathcal{CP}||$. The purpose of the intermediate function is only to facilitate the analysis.

After removing one z-coin at $LCA(z,x)$ for each touched key z, we move coins around as follows:

1. If #(leaves in $\mathcal{S}'_{\mathcal{A}}(x)$) $\geq \frac{|\mathcal{S}_{\mathcal{A}}(x)|}{2\mu}$, we collect 2μ coins from each leaf $b \in \mathcal{S}'_{\mathcal{A}}(x)$ at $LCA(b,x)$ and give one coin to each $z \in \mathcal{S}_{\mathcal{A}}(x)$ at $LCA(z,x)$.
2. If #(crossing edges in $\mathcal{S}_{\mathcal{A}}(x)$) $\geq \frac{|\mathcal{S}_{\mathcal{A}}(x)|}{2\mu}$, there are at least $\frac{|\mathcal{S}_{\mathcal{A}}(x)|}{4\mu}$ disjoint edges $\{z_1, z_2\}$ in $\mathcal{S}_{\mathcal{A}}(x)$. For each such edge $\{z_1, z_2\}$, let z_1 be a key such that $LCA_{\mathcal{O}}(z_1, x) = LCA_{\mathcal{O}}(z_1, z_2)$.
   (a) If $\Phi(z_1, LCA_{\mathcal{O}}(z_2, x)) = 1$, we collect 4μ z_1-coins from $LCA_{\mathcal{O}}(z_2, x)$ and give one coin to each $z \in \mathcal{S}_{\mathcal{A}}(x)$ at $LCA(z,x)$.
   (b) Else we collect 4μ z_2-coins at z_2 and give one coin to each $z \in \mathcal{S}_{\mathcal{A}}(x)$ at $LCA_{\mathcal{O}}(z,x)$.

Observation 3.36. *In all cases, we collect enough coins to pay to all* $z \in \mathcal{S}_{\mathcal{A}}(x)$.

In the second step, we upper bound $||\Psi'|| - ||\widehat{\Psi}||$. Recall that, by Proposition 3.24, we only need to consider (z,a) such that $z \in \mathcal{S}_{\mathcal{A}}(x)$ and $a \in \mathcal{S}_{\mathcal{O}}(x)$. There are two possible scenarios: refill the coins that are paid out ($\Phi(z,a) = 1, \hat{\Phi}(z,a) = 0$), and new extended inversion ($\Phi'(z,a) = 1, \Phi(z,a) = 0$).

Lemma 3.37. *For the paid coins, we add at most* $12\mu|\mathcal{S}_{\mathcal{O}}(x)|$.

Proof. We do case analysis according to how we move coins in the first step.

(1) In case 1, coins are moved from keys that become leaves of $\mathcal{A}'$. We define the witness interval as $J(z) = I_{\mathcal{A}'}(z)$. Notice that these witness intervals are disjoint and each such interval must contain an important point of some $a \in \mathcal{S}_{\mathcal{O}}(x)$. Therefore, we have at most $3|\mathcal{S}_{\mathcal{O}}(x)|$ such intervals and $||\Phi'(z,a)|| - ||\hat{\Phi}(z,a)|| \leq 12\mu|\mathcal{S}_{\mathcal{O}}(x)|$.

(2) There are at least $\frac{|\mathcal{S}_{\mathcal{A}}(x)|}{4\mu}$ disjoint edges $\{z_1, z_2\}$ in $\mathcal{S}_{\mathcal{A}}(x)$. For each such edge $\{z_1, z_2\}$, let z_1 be a key such that $LCA_{\mathcal{O}}(z_1, x) = LCA_{\mathcal{O}}(z_1, z_2)$.
   (a) In case 2a, z_1-coins are moved from $LCA_{\mathcal{O}}(z_2, x)$. We claim that $\Phi'(z_1, LCA_{\mathcal{O}}(z_2, x)) = 0$, so we do not need to add any coins. This is because $LCA_{\mathcal{O}}(z_2, x) \notin I_{\mathcal{A}'}(z_1)$ after $\mathcal{A}$ is updated.
   (b) In case 2b, $LCA_{\mathcal{O}}(z_2, x) \notin I_{\mathcal{A}}(z_1)$ before the update. Since $[x, z_2) \subseteq I_{\mathcal{A}}(z_1)$ and $LCA_{\mathcal{O}}(z_2, x) \in [x, z_2]$, this means $LCA_{\mathcal{O}}(z_2, x) = z_2$. So, $z_2 \in \mathcal{S}_{\mathcal{O}}(x)$. Since we only moved coins from key in $\mathcal{S}_{\mathcal{O}}(x)$, we add at most $4\mu|\mathcal{S}_{\mathcal{O}}(x)|$.

Hence, we add at most $\max\{12\mu|\mathcal{S}_{\mathcal{O}}(x)|, 4\mu|\mathcal{S}_{\mathcal{O}}(x)|\} = 12\mu|\mathcal{S}_{\mathcal{O}}(x)|$ coins. □

Lemma 3.38. *For new extended inversions, we add at most* $4\mu(2\eta_1 + 2\eta_2 + 3)|\mathcal{S}_{\mathcal{O}}(x)|$ *coins.*

Proof. We again use the notion of witness intervals $J(z) = I_{\mathcal{A}'}(z) \setminus I_{\mathcal{A}}(z)$. Now, it suffices to upper bound the number of witness intervals for each $y \in \mathcal{S}_{\mathcal{O}}(x)$. Fix $y \in \mathcal{S}_{\mathcal{O}}(x)$ such that $y < x$. We consider when $J(z)$ each important point of y.

(i) If $\mathsf{left}_{\mathcal{O}}(y) \in J(z)$, by Observation 3.33, it must be that $z < x$ and $\mathsf{left}_{\mathcal{O}}(y) \in (\mathsf{left}_{\mathcal{A}}(z), z)$. Let Q be the collection of such keys z. Observe that keys in Q must be on the same root-to-leaf path in $\mathcal{A}$. Consider two keys $a, b \in Q$. If $a < b$, then $a \preceq b$ (otherwise, $\mathsf{left}_{\mathcal{O}}(y) < a \preceq \mathsf{left}_{\mathcal{A}}(b)$, so $\mathsf{left}_{\mathcal{O}}(y) \notin (\mathsf{left}_{\mathcal{A}}(b), b)$). Now, write $Q = \{z_1, z_2, \ldots, z_q\}$ where $z_1 < z_2 < \ldots < z_q$, so we must have that $z_1 \preceq z_2 \preceq \ldots \preceq z_q$. Going from z_i to z_{i-1} in the tree must use one left-going edge, but overall we have at most η_1 left-going edges on each path. Hence, $|Q| \leq \eta_1 + 1$.

(ii) If $\mathsf{right}_{\mathcal{O}}(y) \in J(z)$, the argument is symmetrical. Hence, there are at most $\eta_2 + 1$ witness intervals.

(iii) If $y \in J(z)$, we first consider the case that $z \geq y$. By Observation 3.33, y must be in $(\mathsf{left}_{\mathcal{A}}(z), z]$. The number of such keys z is at most $\eta_1 + 1$ by the same argument as the $\mathsf{left}_{\mathcal{O}}(z)$ case. There are at most $\eta_1 + 1$ intervals. For the case that $z < y$, if $y \in J(z)$, then y must be in $(z, \mathsf{right}_{\mathcal{A}}(z))$. By Observation 3.33, $(z, \mathsf{right}_{\mathcal{A}}(z)) \cup J(z) = \emptyset$. So, there is no witness interval.

Hence, there are at most $2\eta_1 + \eta_2 + 3$ witness intervals when $y < x$. With symmetric argument, when $y > x$, there are at most $\eta_1 + 2\eta_2 + 3$ witness intervals. When $y = x$, there are at most $\eta_1 + \eta_2 + 2$ witness intervals in total.

□

From Lemma 3.26, we have the following corollary.

Corollary 3.39. *Let* $\mathcal{A}$ *be an awesome BST, then* $\mathcal{A}$ *is* MR*-competitive.*

3.6 Simulation embedding

In this section, we prove that every MR-competitiveness BST algorithm satisfies the simulation embedding property. Note that the terms we use here are rephrased from [36].

Definition 3.40 (Simulation embedding)**.** A BST algorithm $\mathcal{A}$ has the simulation embedding property if, for all access sequence X and algorithm $\mathcal{O}$, there exists a supersequence Y of X such that $\text{cost}_{\mathcal{A}}(Y) \leq O(\text{cost}_{\mathcal{O}}(X))$.

Definition 3.41 (Approximate monotonicity)**.** A BST algorithm $\mathcal{A}$ has the approximately monotone property if, for all access sequence X and subsequence Y of X, we have $\text{cost}_{\mathcal{A}}(Y) \leq O(\text{cost}_{\mathcal{A}}(X))$.

These two properties (simulation embedding and approximate monotonicity) are known to be equivalent to dynamic optimality.

Theorem 3.42 ([36])**.** *A BST algorithm $\mathcal{A}$ is dynamically optimal if and only if it has the simulation embedding property and is approximately monotone.*

Previous proofs of the simulation embedding property [45, 36] are done combinatorially and do not rely on potential function.

Levy and Tarjan [36] show that Splay has thesimulation embedding property by constructing a transition graph G_4 of Splay. Transition graph $G_k = (V, E)$ is a directed graph where each node $v \in V$ represents an instance of k-node BST. There is an edge $(u, v) \in E$ if a tree instance u can be changed to a tree instance v using one access. Also, they show that, in order for BST $\mathcal{A}$ to have the simulation embedding property, it suffices to show that G_k of $\mathcal{A}$ is strongly connected for some constant $k \geq 3$. Also, they mentioned that G_3 is possible to construct for MR.

Proposition 3.43. *The transition graph G_3 of* MR *is strongly connected. Hence,* MR *has the simulation embedding property. (See Figure 3.21)*

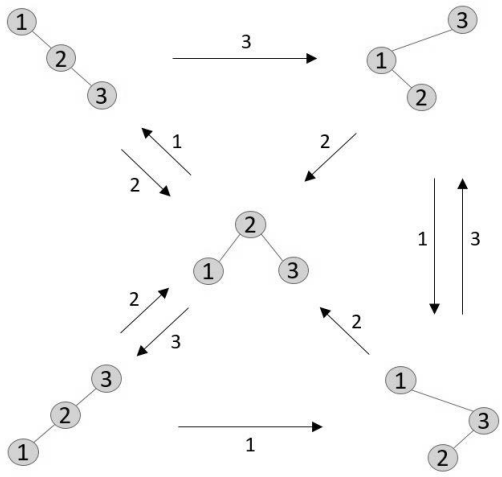


Figure 3.21. G_3 of MR. The number on each arrow represents the key that MR has to access in order to change its tree to the specific structure.

This implies the following theorem.

Theorem 3.44. *If a BST algorithm $\mathcal{A}$ is* MR*-competitive, then $\mathcal{A}$ has the simulation embedding property.*

Proof. Suppose $\mathcal{A}$ is c-competitive to MR. Now fix an input sequence X and a BST algorithm $\mathcal{O}$. From Proposition 3.43, let d be the factor in the simulation embedding property of MR, so we know that there exists a super-sequence Y such that $\text{cost}_{\text{MR}}(Y) \leq d \cdot \text{cost}_{\mathcal{O}}(X)$. Since algorithm $\mathcal{A}$ is c-competitive to MR, we have

$$\text{cost}_{\mathcal{A}}(Y) \leq c \cdot \text{cost}_{\text{MR}}(Y) \leq (cd) \cdot \text{cost}_{\mathcal{O}}(X)$$

This implies that $\mathcal{A}$ has the simulation embedding property. □

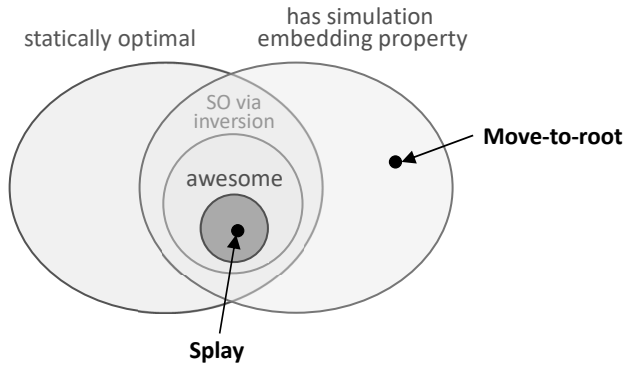


Figure 3.22. A Venn diagram shows the relationship between different sets of BST algorithms

4. Geometric BST

In this chapter, we study a geometric BST algorithm, in particular the Greedy algorithm. We show that the inversion potential function can also be defined for geometric BSTs, and hence, obtain a similar result as the previous chapter for Greedy. In addition, we give another version of the analysis of the lazy finger property using our potential function. This chapter is partly taken from [19].

4.1 Model and definition

In the previous chapter, we already explained how BST algorithms $\mathcal{A}$ in the tree view can be described as intervals $I_{\mathcal{A}}(z)$. In this section, we explain how to do the same for BST algorithms in the geometric view. As discussed in [24], the costs in the tree and geometric views are equivalent up to a constant factor in the competitive ratios.

Let $[n]$ be the set of keys. An access sequence of binary search trees (BST) is specified as searching for key $x_t \in [n]$ at time $t = 1, \ldots, m$. Given an access sequence $X = (x_1, \ldots, x_m) \in [n]^m$, we view X as points $\mathcal{Q}_X = \{(x_t, t) : t = 1, \ldots, m\}$ in the plane where t-th access appears on row t (starting from bottom). We abuse the notation and simply write $\mathcal{Q}_X$ as X.

For $\mathbf{p},\mathbf{q} \in \mathbb{R}^2$, define $\square_{\mathbf{p},\mathbf{q}}$ as the set of integral points in the minimally closed rectangular area defined by $\mathbf{p}$ and $\mathbf{q}$. Let $Z \subseteq \mathbb{R}^2$, we say that such the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is *empty* in Z (or Z-empty) if $Z \cap \square_{\mathbf{p},\mathbf{q}}$ contains no other point than $\mathbf{p}$ or $\mathbf{q}$. Also, the point set Z is *arborally satisfied* if for every pair $\mathbf{p}, \mathbf{q} \in Z$, the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is not Z-empty.

4.1.1 Geometric BST problem

Let $X_{\le t}$ denote the set of points in X in rows t and below. Given an input X of points, we are interested in computing a superset $Y \supseteq X$ such that Y is arborally satisfied, while minimizing $|Y|$. This problem is called MINASS (minimum arborally satisfied superset) problem. In the online version,

at each time $t = 1,2,\ldots,m$, a point in X arrives on the t-th row, and the algorithm must produce a feasible solution $Y_{\leq t}$ at each time t by adding points on the t^{th} row. Points in Y are said to be *touched* by the algorithm $\mathcal{A}$. We denote a set of touched keys at time t by $\mathcal{S}_{\mathcal{A}}(x_t)$ (same notation as search path of x_t in the tree view).

Theorem 4.1 ([24])**.** *Let $\mathcal{A}$ be an algorithm for an online* MINASS *problem. Then for each sequence X, there is a BST algorithm $\mathcal{A}'$ such that $\mathrm{cost}_{\mathcal{A}'}(X)$ is at most a constant times of the cost of $\mathcal{A}$. Also, given a BST algorithm $\mathcal{A}''$, one can construct an arborally satisfied superset Y such that $|Y| \leq \mathrm{cost}_{\mathcal{A}''}(X)$.*

4.1.2 Greedy

This algorithm is first introduced as an offline algorithm called Greedy-Future [37, 39]. Later, [24] shows this algorithm can be made online with cost asymptotically the same.

For each time t and key $z \in [n]$, let $\tau(z,t)$ denote the last touched time of z strictly before time t. At time t, Greedy touches key $z \in [n]$ if and only if $(z,\tau(z,t))$ forms an empty closed rectangle with (x_t,t). The final output of Greedy on input X is defined as $G(X)$; denote the first t rows of such output by $G_{\leq t}(X)$. Note that $G(X)$ contains both X and touched points. See Figure 4.1 for an illustration. The cost of Greedy $|G(X)|$ is defined to be the number of touched keys in total.

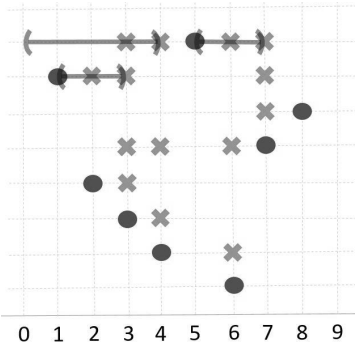


Figure 4.1. An example of intervals of Greedy. Here, circles represent access keys X, and crosses represent touched points $Y \setminus X$. The Interval of each key is defined by the top point (i.e. the last touched point) of such key. Intervals of 2,3, and 6 are shown in this figure.

Greedy is known to have the working set property [26], the lazy finger property [32], and the simulation property [31]. In this chapter, we will show that Greedy is Move-to-root competitive (and hence simulation property) using an extended inversion potential function.

The following observations will be useful for the later proofs. These observations have been used (implicitly or explicitly) in [14, 32].

Observation 4.2. *Let $a, b \in [n]$ with $a < b$. Let b be a touched key at time t. Key a will not be touched again unless some key less than b is accessed after time t.*

Observation 4.3. *For every key $a, b \in [n]$.*

(i) If b is the smallest key larger than a with $\tau(a,t) < \tau(b,t)$, then there exists $t' > \tau(a,t)$ such that $x_{t'} \geq b$.

(ii) If b is the largest key smaller than a with $\tau(a,t) < \tau(b,t)$, then there exists $t' > \tau(a,t)$ such that $x_{t'} \leq b$.

In words, if we know that the last touched time of b is later than a, then there must be an accessed key from the b-side after the last touched time of a.

4.2 Static optimality

4.2.1 Intervals

In order to use our scheme, all satisfied sets must be described in terms of *intervals*. Since the intervals of BST algorithms in a tree view depend on sub-trees, we need an analog of sub-trees in a geometric view. Given a satisfied set S. Let $z \in [n]$ be a key. The interval $I_S(z)$ is defined as $(\mathsf{left}_S(z), \mathsf{right}_S(z))$ where $\mathsf{left}_S(z)$ is the maximum key less than z that is touched at the last touched time of z (0 if it does not exist). Similarly, we define $\mathsf{right}_S(z)$ as the minimum key greater than z that is touched at the last touched time of z ($n+1$ if it does not exist). So the interval $I_S(z)$ only change whenever z is touched. When a satisfied set is obtained by Greedy, we use a notation $I_G(z)$ as an interval of z. See Figure 4.1.

Observation 4.4. *Let x_t be an accessed key on Greedy. The following statements are equivalent:*

(i) z is touched at time t.

(ii) $(z, \tau(z,t))$ forms an empty rectangle with (x_t, t).

(iii) $x_t \in I_G(z)$ at time t.

Proof. For, *(i)* $\iff$ *(ii)*, it holds by definition. Without loss of generality, assume that $x_t < z$. For *(ii)* $\implies$ *(iii)*, the empty rectangle implies that there is no touched key in $[x_t, z]$. For *(iii)* $\implies$ *(ii)*, assume that the rectangle is not empty. Let point $p = (p.x, p.y)$ be the lowest point in the rectangle. Notice that $p.y > t$ because $x_t \in I_G(z)$ at time t. Consider an access at time $p.y$, called $x_{p.y}$. It must be that $x_{p.y} < z$. Since p is the lowest point in the

rectangle formed by $(z,\tau(z,t))$ and (x_t,t), the rectangle formed by $(z,\tau(z,t))$ and $(x_{p.y},p.y)$ must be empty. This means z is touched at time $p.y$ and hence contradicts the fact that $\tau(z,t)$ is the last touched time of z.

□

We also use $\mathcal{S}_G(x)$ as a notation for a set of keys touched by Greedy (same notation as search path).

Given an optimal tree $\mathcal{O}$, Lemma 3.20, Lemma 3.21 and Proposition 3.24 still hold for these geometric intervals. The following observations are important.

Observation 4.5. *Suppose key x_t is accessed. Let $\{I_G(b)\}_{b\in[n]}$ and $\{I_{G'}(b)\}_{b\in[n]}$ be the intervals before and after Greedy updates. Let $b_1 < b_2 < \ldots < b_k$ be the touched keys, $b_0 = 0$ and $b_{k+1} = n+1$. Then we have that $I_{G'}(b_i) = (b_{i-1}, b_{i+1})$ for all $i = 1,\ldots,k$. (See Figure 4.2a)*

Corollary 4.6. *For each point $p \in \mathbb{R}$, there are at most two touched keys z for which $p \in I_{G'}(z)$. In other words, the clique size of this interval graph is 2, and therefore, the touched intervals $I_{G'}(z)$ can be decomposed into two disjoint sets of intervals.*

We show the following static optimality bound for Greedy using our potential.

Theorem 4.7. *For each X and each static tree $\mathcal{O}$, $|G(X)| \le 4\cdot\mathrm{cost}_{\mathcal{O}}(X) - m + n^2$.*

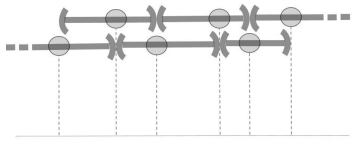
The potential function we use is simply the extended inversion Φ. Let $\mathcal{O}$ be a static tree. After accessing x, the Greedy intervals change from $\{I_G(z)\}_{z\in[n]}$ to $\{I_{G'}(z)\}_{z\in[n]}$, and the potential changes from Φ to Φ'. We will prove that $||\Phi'|| \le ||\Phi - \mathcal{CP}|| + 4|\mathcal{S}_{\mathcal{O}}(x)| - 1$.

With at most n^2 initial coins, summing over all accesses over a sequence X of length m will give us the desired bound. Denote $\Phi - \mathcal{CP}$ by $\widehat{\Phi}$. Due to Proposition 3.24, the change of potential function is only due to the inversions (z,α) where $z \in \mathcal{S}_G(x)$ and $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$.

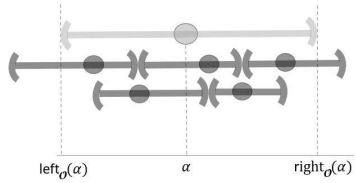
Lemma 4.8. *For each $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$, we have $||\Phi'(\bullet,\alpha)|| - ||\widehat{\Phi}(\bullet,\alpha)|| \le 4$. Moreover, $||\Phi'(\bullet,x)|| - ||\widehat{\Phi}(\bullet,x)|| \le 3$.*

Proof. Fix $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$. Notice that the term $\Phi'(z,\alpha) - \widehat{\Phi}(z,\alpha) = 1$ only if $z \in \mathcal{S}_G(x)$ and $\Phi'(z,\alpha) = 1$. To have $\Phi'(z,\alpha) = 1$, the interval $I_{G'}(z)$ must contain a point in $\mathcal{P}_{\mathcal{O}}(\alpha) = \{l,\alpha,r\}$ (where $l = \mathsf{left}_{\mathcal{O}}(\alpha)$ and $r = \mathsf{right}_{\mathcal{O}}(\alpha)$) and point z must be inside $I_{\mathcal{O}}(\alpha)$. From the fact that z is inside $I_O(\alpha)$ and the geometry of Greedy (Observation 4.5), only one interval can contain l, only one interval can contain r, and at most two intervals can contain α (this is all illustrated in Figure 4.2b); moreover, when $\alpha = x$, only one interval may contain α ($I_G(x)$ itself). This concludes the proof. □

Corollary 4.9. *Greedy is* MR*-competitive and has simulation property.*



(a) Intervals of keys that are touched at the same time in Greedy. Think of all of them as in the same height, we perturb them in order to show their intervals clearly.



(b) Intervals in G are represented in black, while interval in $\mathcal{O}$ is represented in grey

Figure 4.2

4.3 Lazy finger bound

In this section, we will show the following theorem.

Theorem 4.10. *For each access sequence X and each static tree $\mathcal{O}$, we have*

$$|G(X)| \leq 5 \cdot \mathsf{LF}_{\mathcal{O}}(X) + O(m + n^2).$$

As mentioned, compared to [32], our analysis is done by directly comparing two BSTs while their analysis defines a potential function based on leaf-oriented BSTs. Our constant is also more optimized. However, we are not claiming our proof as a better proof, but rather we view it as a "proof of concept" that our new potential function is strong enough to derive the state-of-the-art results in a systematic way.

Recall that $\mathsf{LF}(X) = \min_R \sum_{t=1}^{m} d_R(x_{t-1}, x_t)$ where x_0 is a root of R and $d_R(a,b)$ denote the number of edges on the unique path in R from a to b. One way to view this is that, in each access x_t, there is a finger f at x_{t-1} that has to move to x_t. Finger f can only move to the adjacent node in BST with unit cost, so f has to move $d_{\mathcal{O}}(x_{t-1}, x_t)$ steps in total. Due to the BST structure, f will move a number of steps upwards (potentially zero), before moving a number of steps downwards (potentially zero). We will sometimes abuse the notation of f to be the key that f is pointing at. We imagine the process of moving finger f from x_t to x_{t+1} in $\mathcal{O}$ as being divided into several **atomic time steps**. The atomic time steps are discrete and are identified by (t, f) where t is the time step and f is the location of the finger between time $t-1$ and t. There are exactly $d_{\mathcal{O}}(x_{t-1}, x_t)$ atomic time steps between $t-1$ and t. These atomic time steps are divided into three phases:

(i) (Up phase:) Finger f moves from x_{t-1} to $\ell = LCA_{\mathcal{O}}(x_{t-1}, x_t)$.

(ii) (Down phase:) Finger f moves from ℓ to x_t.

(iii) (Access phase:) Key x_t is accessed.

We call these atomic time steps altogether the time step t. To prove

Theorem 4.10, we bound the change of potential function for each atomic step.

4.3.1 Status

First, we define a *status* for each key $z \in [n]$ on Greedy. The status of z at time t can be identified by the execution log of Greedy at time t (see Figure 4.3), and it may change at each time (even when the key is not touched). The status of key z then will be used to define $I_G(z)$ and potential for z. Here, the intervals are designed a bit differently (but still in a rather systematic way– as we will see). Each key z must be one of these three statuses:

(i) A key z is *top-external* at time t iff z is the leftmost/rightmost touched key at time t.

(ii) A key z is *external* at time t iff z is not touched at time t and z is the leftmost/rightmost touched key at time $\tau(z,t)$.

(iii) A key z is *internal* at time t otherwise.

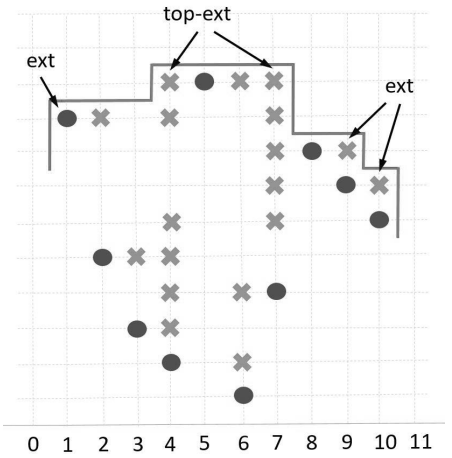


Figure 4.3. Status of each key in Greedy execution log. At the current time, $\{4,7\}$ are top-external, while $\{1,9,10\}$ and $\{2,3,5,6,8\}$ are external and internal respectively. Examples of intervals $I_G(3) = (2,4)$, $I_G(4) = (0,11)$, $I_G(9) = (7,10)$.

Lemma 4.11. *For each time t, let Z_{ext} and $Z_{top\text{-}ext}$ be a set external and top-external keys, respectively. Let*

- $Z_1 = \{(z, \tau(z,t)) : z \in Z_{ext}$ *and* $z \le x_t\} \cup \{(z,t) : z \in Z_{top\text{-}ext}$ *and* $z \le x_t\}$
- $Z_2 = \{(z, \tau(z,t)) : z \in Z_{ext}$ *and* $z \ge x_t\} \cup \{(z,t) : z \in Z_{top\text{-}ext}$ *and* $z \ge x_t\}$

Then Z_1 is monotone increasing, and Z_2 is monotone decreasing[1].

[1]We say that a point set Z is monotone increasing if for all $(z,t),(z',t') \in Z$, we have $z < z'$ iff $t < t'$. We can define a monotone decreasing set analogously.

Proof. This follows immediately from the following statement. For a fixed time t, z is external at time t iff

(i) $\square_{(z,\tau(z,t)),(0,t)}$ is empty for $z < x_t$.

(ii) $\square_{(z,\tau(z,t)),(n+1,t)}$ is empty for $z > x_t$.

We will prove for the case that $z < x_t$. The other case is symmetrical.

($\Rightarrow$) First, observe that z is the leftmost touched key at time $\tau(z,t)$, otherwise the first access key larger than z must touch z (it exists because x_t is one). Let z' be the smallest key larger than z that is touched at time $\tau(z,t)$ ($\tau(z,t) \neq t$ by the definition of external). First, note that such key must exist, otherwise z would get touched again at time $\tau(z,t)+1$, which contradicts the fact that $\tau(z,t)$ is the last time that z is touched. For the sake of contradiction, let t' be the first time that $\square_{(z,\tau(z,t)),(0,t')}$ is non-empty.

If $x_{t'} < z'$, then z must be touched because $\square_{(z,\tau(z,t)),(x_{t'},t')}$ is empty. This contradicts the fact that $\tau(z,t)$ is the last time that z is touched.

If $x_{t'} \geq z'$, then no key $z'' \leq z$ can be touched because $\square_{(z'',\tau(z'',t)),(x_{t'},t')}$ contains z'. This contradicts the fact that $\square_{(z,\tau(z,t)),(0,t')}$ is fist non-empty at time t'.

($\Leftarrow$) Since $\square_{(z,\tau(z,t)),(0,t)}$ is empty, z is the leftmost touched key at time $\tau(z,t)$. So, z is external.

□

This allows us to easily infer the status of each key from the Greedy execution log and to use geometric insight to argue about correctness. See Figure 4.3. Intuitively, external and top-external keys define a "mountain" where x_t is on the top, the left part is the monotone increasing set and the right part is monotone decreasing. The rest of the keys are internal.

We make some useful observations in order for the readers to be familiar with the concept of status. All of them can be easily verified by looking at the Greedy execution log.

Observation 4.12. *Regarding status, the following statements hold:*

- *If z is touched when accessing x_t, then z becomes either internal of top-external.*
- *If z is internal or external, it will remain the same status as long as it is not touched.*
- *If z is top-external, it will still be top-external if it is touched. However, it will be external if it is not touched.*

4.3.2 Intervals

In this subsection, we define intervals for a static tree $\mathcal{O}$ and a satisfied set of Greedy G.

Given a static lazy finger tree $\mathcal{O}$, for each key y, $I_{\mathcal{O}}(y) = (\mathsf{left}_{\mathcal{O}}(y), \mathsf{right}_{\mathcal{O}}(y))$ where $\mathsf{left}_{\mathcal{O}}(y)$ is the nearest left ancestor of y (0 if not exist) and $\mathsf{right}_{\mathcal{O}}(y)$ is the nearest right ancestor of y ($n+1$ if not exist). Note that $I_{\mathcal{O}}(y)$ is defined exactly the same way as in §3.3.

For Greedy, the intervals will now be defined based on the status of the keys. For internal keys, we will use the same definition, but it will be different for other status. See Figure 4.3.

(i) For a *top-external* key z, $I_G(z) = (0, n+1)$

(ii) For an *external* key z, $I_G(z) = (\mathsf{left}_G(z), \mathsf{right}_G(z))$ where $\mathsf{left}_G(z)$ is the next external or top-external on the left of z (0 if not exist) and $\mathsf{right}_G(z)$ is the next external or top-external on the right of z ($n+1$ if not exist)

(iii) For an *internal* key z, $I_G(z) = (\mathsf{left}_G(z), \mathsf{right}_G(z))$ where $\mathsf{left}_G(z)$/$\mathsf{right}_G(z)$ is the next key on the left/right that is touched at the last time z is touched.

Lemma 4.13. *When accessing x_t, if z is touched and $x_t \notin I_G(z)$, then z is external. Moreover, $I_{G'}(z) \subseteq I_G(z)$ where $I_{G'}(z)$ denote an interval of z after Greedy updates.*

Proof. We prove by contraposition. For a touched top-external key z, $I_G(z) = (0, n+1)$, so every key must be in $I_G(z)$. For a touched internal key z, assume that $z < x$ (symmetric argument for $z > x$). Since z is touched, this means $\square_{(z,\tau(z,t)),(x_t,t)}$ is empty. Consider $I_G(z) = (\mathsf{left}_G(z), \mathsf{right}_G(z))$. We have that $\mathsf{right}_G(z) > x_t$. So, $x_t \in I_G(z)$.

Next, we show that $I_{G'}(z) \subseteq I_G(z)$. Assume without loss of generality that $z < x_{t-1}$, i.e. z is external key on the left side of the mountain before time t. Notice that the keys that define $\mathsf{left}_G(z)$ and $\mathsf{right}_G(z)$ are external or top-external. Since $x_t \notin I_G(z)$, there are two cases: (1) $x_t \le \mathsf{left}_G(z)$ (2) $x_t \ge \mathsf{right}_G(z)$. In the first case, according to Lemma 4.11, $\square_{(\mathsf{left}_G(z),\tau(\mathsf{left}_G(z),t)),(x_t,t)}$ and $\square_{(\mathsf{right}_G(z),\tau(\mathsf{right}_G(z),t)),(x_t,t)}$ are empty. Hence, both $\mathsf{left}_G(z)$ and $\mathsf{right}_G(z)$ are touched, and z becomes internal with $I_{G'}(z) = I_G(z)$. In the second case, z cannot be touched because $\square_{(z,\tau(z,t)),(x_t,t)}$ contains $\mathsf{right}_G(z)$.

□

4.3.3 Scope

For each key z, let H_z (the history of z) denote the set of nodes in $\mathcal{O}$ that has been visited by the finger f after the last touch time of z (including all nodes in the paths traversed by the finger). Define f_z as the highest node in H_z.

$$\text{scope}(z) := LCA_{\mathcal{O}}(z, f_z)$$

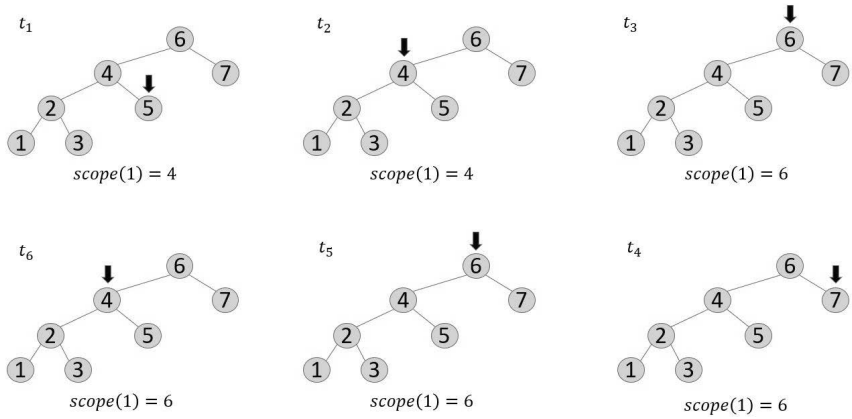


Figure 4.4. Example of of scope(1). This figure shows tree $\mathcal{O}$ at each consecutive atomic time step. Assume that key 1 is touched by Greedy at t_1 and never again later, then scope(1) = $LCA_{\mathcal{O}}(1,5) = 4$. After that, scope(1) may change in each atomic time t_i.

By this definition, as long as z is not touched again, scope(z) can only become higher up in the tree. While the intervals $I_G(z)$ only change once per access, the scopes may change at every atomic time step. To give an example, let $x_{\tau(z,t)}$ be the last access during which z is touched. Right after $x_{\tau(z,t)}$ is accessed, scope(z) = $LCA_{\mathcal{O}}(z, x_{\tau(z,t)})$. After that, if f moves down, scope(z) does not change. But if f moves up to an ancestor v of $LCA_{\mathcal{O}}(z, x_{\tau(z,t)})$, then scope($z$) becomes node v. See Figure 4.4.

Observation 4.14. *Consider an atomic time step where finger f moves from v to v' in $\mathcal{O}$. Let* scope *and* scope$'$ *denote the scope before and after the move of the finger, respectively. For each $z \in [n]$,*

(i) if v' is a child of v, then scope$'(z)$ = scope(z)

(ii) if v' is a parent of v,

- *if* scope(z) = v, *then* scope$'(z) = v'$
- *otherwise,* scope$'(z)$ = scope(z).

4.3.4 Range

Let us say that the finger f is at key a. Consider the search path $\mathcal{S}_{\mathcal{O}}(a)$ of this finger. We order $\mathcal{S}_{\mathcal{O}}(a) = \{a_{-L}, a_{-(L-1)}, \ldots, a_0 = a, a_1, \ldots, a_R\}$ where $a_{-L} < a_{-(L-1)} < \ldots < a_0 = a < a_1 < \ldots < a_R$. Define additionally $a_{-(L+1)} = 0$ and $a_{R+1} = n + 1$ (for handling the corner cases).

(i) For each $i \in \{-L, \ldots, -1\}$, define range(a_i) = $(a_{i-1}, a_i]$.

(ii) For each $i \in \{1,\ldots,R\}$, define $\text{range}(a_i) = [a_i, a_{i+1})$.

See Figure 4.5. Notice that the ranges are completely determined by the location of finger f. We call $\mathcal{R} = \{\text{range}(a_i) \mid i \in \{-L,\ldots,R\}\backslash\{0\}\}$ the set of *active* ranges.

Observation 4.15. *Consider an atomic step where finger f moves from v to v'. Let $\mathcal{R}$ and $\mathcal{R}'$ be the ranges before and after the move.*

(i) If v' is a parent of v, then $\mathcal{R}' = \mathcal{R}\backslash\{\text{range}(v)\}$.

(ii) If v' is a child of v, then $\mathcal{R}' = \mathcal{R} \cup \{\text{range}(v)\}$.

4.3.5 Potential function

We are ready to define the potential function Ψ which consists of three parts $||\Psi|| = ||\widetilde{\Phi}|| + ||E|| + ||V||$, where $\widetilde{\Phi}$ is a modified geometric inversion function.

We will use $x \preceq_T y$ to denote the fact that key x is in the subtree rooted at y in T.

(i) **Extended Inversion Potential ($\widetilde{\Phi}$)**:
$\widetilde{\Phi}(z,a) = 1 \iff z$ forms an extended inversion with a and $a \preceq_{\mathcal{O}} \text{scope}(z)$. We imagine placing a z-coin at a.

(ii) **External Potential (E)**:
If z is external or top-external, then $E(z) = 1$.

Notice that the potential function $\widetilde{\Phi}$ depends on $\text{scope}(z)$ and $I_G(z)$. We remark that the potential may sound ad-hoc at first but it in fact comes from a clean principle of inversion. For the lazy finger tree $\mathcal{O}$, an access to x_t, knowing the location of finger f, costs us (on average) the distance $d_{\mathcal{O}}(LCA_{\mathcal{O}}(x_t, f), x_t)$. In other words, the coins of z only need to be placed at the keys on the search path from z to $LCA_{\mathcal{O}}(z, f)$, and this is exactly what the scope captures.

Finally, to define the last potential V, we translate the concept of virgin subtrees in [32] into our setting. We say that a range $r \in \mathcal{R}$ is **virgin** if no key in r has been touched since the atomic time step at which r has become an active range in $\mathcal{R}$.

Virgin Potential (V):
For each virgin active range $r \in \mathcal{R}$, we have $V(r) = 4$.

Let $D^{\uparrow}(t)$ and $D_{\downarrow}(t)$ be the number of times that f moves up and moves down, respectively, when finger f moves from x_{t-1} to x_t in $\mathcal{O}$. Now, we are ready to prove the main lemma, which keeps track of how the potential function changes in one "shifted cycle" (which covers parts of time t and parts of time $t+1$.)

Lemma 4.16 (Potential Change)**.** *Let Ψ be the current potential function. Then in the following phases,*

(i) (Down phase of t:) Finger f moves down from $LCA_{\mathcal{O}}(x_{t-1}, x_t)$ to x_t. Let Ψ' be the potential function at the end of this phase.

(ii) (Access of t:) Key x_t is accessed and Greedy updates the intervals. Let Ψ'' be the potential at the end of this phase.

(iii) (Up phase of $t+1$:) Finger f moves up from x_t to $LCA_{\mathcal{O}}(x_t, x_{t+1})$. Let Ψ''' be the potential at the end.

Then the potential function changes as follows:

(i) $||\Psi'|| - ||\Psi|| \leq 4D_{\downarrow}(t)$,

(ii) $|\mathcal{S}_G(x_t)| + ||\Psi''|| - ||\Psi'|| \leq 16$,

(iii) $||\Psi'''|| - ||\Psi''|| \leq 6D^{\uparrow}(t+1)$.

Corollary 4.17. *The total access cost $\sum_t |\mathcal{S}_G(x_t)|$ by Greedy is at most $5\mathsf{LF}_{\mathcal{O}}(X) + 16m$.*

Proof. Since x_0 is root, we start at the down phase of 1. In the end, finger stops at x_m, so we finish with the down phase of m and access of m. From Lemma 4.16, we sum over $t = 1, 2, \ldots, m$, we get (by telescoping)

$$\sum_{t=1}^{m} |\mathcal{S}_G(x_t)| + ||\Psi^{\text{last}}|| - ||\Psi^{\text{init}}|| \leq 4D_{\downarrow} + 6D^{\uparrow} + 16m$$

where $D_{\downarrow} = \sum_{t \in [m]} D_{\downarrow}(t)$ and similarly for $D^{\uparrow}$. After accessing x_m, observe that the finger stays there at x_m. Since we start at the root of $\mathcal{O}$, we have that $D_{\downarrow} \geq D^{\uparrow}$. Since $||\Psi^{\text{init}}|| \leq 2n^2$, the total access cost by Greedy is upper bounded by:

$$\begin{aligned} 4D_{\downarrow} + 6D^{\uparrow} + 16m + 2n^2 &\leq 5D_{\downarrow} + 5D^{\uparrow} + 16m + 2n^2 \\ &\leq 5\mathsf{LF}_{\mathcal{O}}(X) + 16m + 2n^2. \end{aligned}$$

□

4.3.6 Analysis of the main lemma

In this subsection, we show the proof of Lemma 4.16. We use an extended inversion potential function notations as follows:

- $\widetilde{\Phi}$: before the down phase of t
- $\widetilde{\Phi}'$: after the down phase of t (or before the access phase of t)

- $\widetilde{\Phi}''$: after the access phase of t (or before the up phase of $t+1$)
- $\widetilde{\Phi}'''$: after the up phase of $t+1$.

We denote external potential function E, E', E'', E''' and virgin potential function V, V', V'', V''' similarly.

[Down phase of t] In this phase, for each key z, scope(z) and $I_G(z)$ is unchanged by Observation 4.14. So the difference $||\widetilde{\Phi}'|| - ||\widetilde{\Phi}|| = 0$ and $||E'|| - ||E|| = 0$. For each atomic step where finger f moves down from v to its child v', a virgin range(v) is created, therefore increasing the virgin potential function by exactly 4 per step. In total, $||V'|| - ||V|| = 4D_{\downarrow}$.

[Access phase of t] Similarly to before, since $||\mathcal{CP}|| = |\mathcal{S}_G(x_t)|$, it suffices to show that $||\Psi''|| - ||\Psi' - \mathcal{CP}|| \le 16$. We create an intermediate function (which is not a valid potential function) $\widehat{\Psi}$ such that $||\widehat{\Psi}|| = ||\Psi' - \mathcal{CP}||$; according to the definition $\mathcal{CP}$ is defined to have coins at the LCAs between touched keys and the searched key, but our function $\widehat{\Psi}$ instead remove (the same number of) coins differently.

Define $\widehat{\Psi}$ as follows. Each key $z \in \mathcal{S}_G(x_t)$ has to pay for its cost. We handle this differently for keys z of different status. If z is an external or top-external, deduct the function $E'(z) \leftarrow E'(z) - 1$. If z is an internal key, then deduct the z-coin at $LCA_{\mathcal{O}}(z, x_t)$.

For each key $z \in [n]$, scope(z) could change to $LCA_{\mathcal{O}}(z, x_t)$ if z is touched, while $I_G(z)$ could increase or decrease. The change of scope(z) to $LCA_{\mathcal{O}}(z, x_t)$ does not increase the extended inversion potential function because the new scope is lower down. So we only need to worry about the change from I_G to $I_{G'}$.

Once the intervals get updated from I_G to $I_{G'}$. Notice that there are two keys that become top-external keys (leftmost and rightmost touched keys), while the rest become internal.

Observation 4.18. $||E''|| - ||E' - \mathcal{CP}|| \le 2$.

Lemma 4.19. *If $z \notin \mathcal{S}_G(x_t)$, then* $||\widetilde{\Phi}''(z,\cdot)|| - ||\widetilde{\Phi}'(z,\cdot) - \mathcal{CP}(z,\cdot)|| \le 0$.

Proof. Note that $||\mathcal{CP}(z,\cdot)|| = 0$ in this case. We will show that $I_{G'}(z) \subseteq I_G(z)$ for each untouched key z. This implies the lemma since no new extended inversion can be created.

(i) If z was internal, it can only remain internal. $I_{G'}(z)$ is defined by the last touched time of z, so $I_{G'}(z) = I_G(z)$.

(ii) If z was top-external, it becomes external. Since $I_G(z) = (0, n+1)$, $I_{G'}(z) \subseteq I_G(z)$.

(iii) If z was external, it can only remain external. Assume without loss of generality that $z < x_t$. We know that z must be external on the left of the mountain both before the access (otherwise z will be touched).

Let $I_G(z) = (l, r)$. Recall that the interval for external is defined by the next left and right external or top-external. We know that l (if it existed) is not touched because, when x_t is accessed, $\square_{(l,\tau(l,t)),(x_t,t)}$ contains z. So, l remains external. For r (if it existed), if r is touched, we consider 2 cases: (1) $x_t \geq r$ and (2) $z < x_t < r$. In this first case, since no key less than r is touched, r becomes top-external. So, $I_{G'}(z) \subseteq I_G(z)$. For the second case, since no key $z' \leq z$ is touched ($\square_{(z',\tau(z',t)),(x_t,t)}$ contains z), the leftmost touched key at time t must be in $(z, x_t]$. This key define the new r, so $I_{G'}(z) \subseteq I_G(z)$.

□

Lemma 4.20. *If $z \in \mathcal{S}_G(x_t)$ and $y \notin \mathcal{S}_{\mathcal{O}}(x_t)$, then $||\widetilde{\Phi}''(z,y)|| - ||\widetilde{\Phi}'(z,y) - \mathcal{CP}(z,y)|| \leq 0$.*

Proof. Note that $||\mathcal{CP}(z,y)|| = 0$ in this case. Consider key $z \in [n]$. We check whether we need a z-coin at y in two cases. When $x_t \in I_G(z)$, we have $LCA_{\mathcal{O}}(x_t, z) \in I_G(z)$. Since z-coins form a subpath, there must be a z-coin at each node in $\mathcal{S}_{\mathcal{O}}(z) \setminus \mathcal{S}_{\mathcal{O}}(x)$ already (which includes y). Otherwise, if $x_t \notin I_G(z)$, from Lemma 4.13, z must be external and $I_{G'}(z) \subseteq I_G(z)$. □

Lemma 4.21. $(||\widetilde{\Phi}''|| + ||V''||) - (||\widetilde{\Phi}'|| + ||V'|| - ||\mathcal{CP}||) \leq 14$.

Proof. From Lemma 4.19 and Lemma 4.20, the extended inversion potential function can only increase for $z \in \mathcal{S}_G(x_t)$ and $y \in \mathcal{S}_{\mathcal{O}}(x_t)$. Recall that touched keys can only be internal or top-external. When $y = x_t$ and $z \in (\mathsf{left}_{\mathcal{O}}(y), \mathsf{right}_{\mathcal{O}}(y))$, the new value of scope$(z)$ is $LCA_{\mathcal{O}}(z, y) = y$. So, there are at most 6 keys $z \in \mathcal{S}_G(x_t)$ that have extended inversion with y: 4 internal keys (at most one key z such that $I_G(z)$ contains $\mathsf{left}_{\mathcal{O}}(y)$, at most one key z such that $I_G(z)$ contains $\mathsf{right}_{\mathcal{O}}(y)$, at most two key z such that $I_G(z)$ contains y), and (possibly) 2 top-external keys. So, the extended inversion potential function increases by at most 6 here.

Consider each $y \in \mathcal{S}_{\mathcal{O}}(x_t)$ such that $y \neq x_t$. Assume without loss of generality that $y < x_t$. For each touched key $z \in (y, \mathsf{right}_{\mathcal{O}}(y))$, we have that scope$(z) = LCA_{\mathcal{O}}(z, x_t)$. Since $LCA_{\mathcal{O}}(z, x_t)$ is a strict descendant of y in $\mathcal{O}$, z cannot have a z-coin at y. This means only $z \in (\mathsf{left}_{\mathcal{O}}(y), y] = \text{range}(y)$ can have a z-coin in y. There are at most 4 keys $z \in \mathcal{S}_G(x_t)$ in range(y) that have extended inversion with y: key z such that $I_G(z)$ contains $\mathsf{left}_{\mathcal{O}}(y)$, key z such that $I_G(z)$ contains y, and (possibly) 2 top-external keys. See Figure 4.5 for the non-top-external cases. We now know that the increase in $\widetilde{\Phi}'$ is at most $4|\mathcal{S}_{\mathcal{O}}(x_t)|$, which is too much. We show that we can use the decrease of potential V' to make up for it.

Let $\tilde{\mathcal{R}} = \{r \in \mathcal{R} \mid r \cap \mathcal{S}_G(x_t) \neq \emptyset\}$. In words, we only consider those range(y) with a touched key inside. Let $r_{\min}$ and $r_{\max}$ be the leftmost and rightmost range in $\tilde{\mathcal{R}}$. These two ranges contribute to the increase of $\widetilde{\Phi}'(\cdot, r_{\min}) + \widetilde{\Phi}'(\cdot, r_{\max})$ by at most 8. For y such that range$(y) \in \tilde{\mathcal{R}} \setminus \{r_{\min}, r_{\max}\}$, we will

use coins of range(y) to convert into z-coins at y. It suffices to show that each range(y) $\in \tilde{\mathcal{R}} \setminus \{r_{\min}, r_{\max}\}$ is virgin (so V' would decrease by a value of 4 and overall $||\widetilde{\Phi}''|| + ||V''||$ does not increase at y). We will show that, if there is a non-virgin range range(y') with a touched key inside, then such range must be either $r_{\min}$ or $r_{\max}$. Assume w.l.o.g. that range(y') $< x_t$. Let $t_{y'}$ be the last atomic time before time t that range(y') is created. Notice that, after time $t_{y'}$, we cannot access any key less than y' because finger has to move up to y', which will erase range(y'). When range(y') became non-virgin, there is a key $z \in$ range(y') that is touched. By Observation 4.2, no key less than z is touched since then. So, range(y') must be $r_{\min}$. □

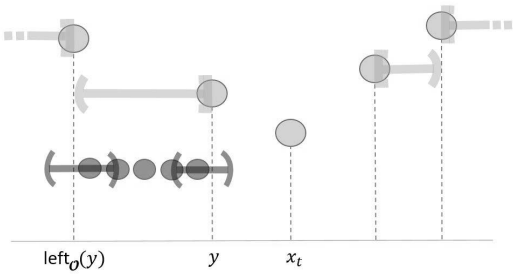


Figure 4.5. Grey intervals represent range of each light node. Black intervals show the only two non-top-external keys z that can have z-coin at y.

[Up phase of $t+1$] In this phase, for each key z, $I_G(z)$ is unchanged, but scope(z) might become higher up. The potential V does not increase since no new active range is created. The same goes for E. The following lemma bounds the increase in extended inversion potential function.

Lemma 4.22. *Let v be a child of v'. In each step that f moves up from v to v', the increase of $\widetilde{\Phi}$ is at most 6. Hence, in total, $||\widetilde{\Phi}'''|| - ||\widetilde{\Phi}''|| \le 6D^{\uparrow}$.*

Proof. Assume w.l.o.g. that v is the left child of v'. First, observe that, in order for scope(z) to be increased, z must be in $I_{\mathcal{O}}(v) = (\text{left}_{\mathcal{O}}(v'), v')$. We claim that there are at most 6 keys z such that $I_G(z)$ contains left$_{\mathcal{O}}(v')$ or v' and z needs z-coin at v' (hence increasing the potential function.) See Figure 4.6.

There are always 2 top external keys. For external keys, by definition of intervals of external keys, there is one interval that contains v' and one interval that contains left$_{\mathcal{O}}(v')$, so in total there are two such external keys.

Finally, we look at internal keys. We will argue that there is at most one key z for which $v' \in I_G(z)$ that needs z-coin, and also at most one z for which left$_{\mathcal{O}}(v') \in I_G(z)$ that needs z-coin. We show the proof for only v'; the other case can be proved analogously.

Assume for contradiction that there are two keys $z_1, z_2 : z_1 < z_2 < v'$ for which $v' \in I_G(z_1)$ and $v' \in I_G(z_2)$, and also need their coins at v'. There are

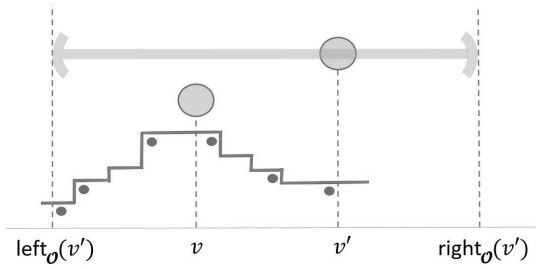


Figure 4.6. Red rectilinear segments are keys in Greedy. Inside, there are 6 nodes that might need coins when the finger moves from v to v'.

two cases. In the first case, if $\tau(z_1,t) \le \tau(z_2,t)$, then $\mathsf{right}_G(z_1) \le z_2$ must hold (otherwise, we cannot touch z_2 at time $\tau(z_2,t)$ without touching z_1). This contradicts the fact that $v' \in I_G(z_1)$. In the second case, if $\tau(z_1,t) > \tau(z_2,t)$, we will show that $\mathsf{scope}(z_2) \ge v'$ before the finger moves, so z_2 must have z_2-coin at v' already. Since $\tau(\mathsf{right}_G(z_1),t) \ge \tau(z_1,t) > \tau(z_2,t)$, by Observation 4.3, we must have an access $x_{t'} \ge \mathsf{right}_G(z_1)$ after time $\tau(z_2,t)$. This means the finger moved up from v to v' at some point after $\tau(z_2,t)$ (because $\mathsf{right}_G(z_1) > v'$). So, $\mathsf{scope}(z_2)$ would have been already equal or above v' and z_2 must have z_2-coin at v' already. This is a contradiction. □

4.4 Sequential access

Given an initial tree, we consider Greedy on sequential access $X = (1,2,...,n)$. [26] shows that the total cost is at most $4n-2$. We improve this to $3n-2$.

Theorem 4.23. *Given an initial tree R, Greedy on sequential access cost at most $3n-2$.*

Let $G(X)$ be the Greedy points on X. Notice that the points in X lie on the diagonal line $x = y$. Decompose $G(X)$ into $X \cup Y_L \cup Y_R$ where $Y_L = \{q \in G(X) \mid q.y > q.x\}$ and $Y_R = \{q \in G(X) \mid q.y < q.x\}$. In words, the sets Y_L and Y_R are the points strictly on the left and right of the diagonal line respectively.

Observation 4.24. $|Y_L| \le n-1$.

Given a static BST tree T, we define a *right spine* P_z of node z as the path from z to the rightmost node in the subtree rooted at z, T_z. For each node z, let $h_z = |P_{left(z)}|$ where $left(z)$ is a left child of z (if z has no left child, $h_z = 0$).

Lemma 4.25. *$\forall z \in [n]$, z is touched exactly h_z times in Y_R.*

Proof. Let $k = |P_{left(z)}|$ and $p_1, p_2, ..., p_k$ be elements in $P_{left(z)}$ such that $p_1 < p_2 < ... < p_k$. Let p_0 be the left ancestor of x with the highest key value

($p_0 = 0$ if x has no left ancestor). For $i \in [k]$, we claim that z gets the i^{th} touch at time $p_{i-1}+1$. We show by induction. The base case is easy to verify. Assume that z gets the $(i-1)^{th}$ touch at time $p_{i-2}+1$. Notice that key p_{i-1} is also touched at this time. This means that key p_{i-1} prevents key z from forming an empty rectangle with any access point $x_{\leq p_{i-1}}$. At time $p_{i-1}+1$, the rectangle formed by key z and the access point $x_{p_{i-1}+1}$ becomes empty again. See Figure 4.7. □

Observation 4.26. $\sum_{x \in T} h_x \leq n-1$.

This is because every tree has at least one node that is not a left child of any node (root).

Since there are n input points, the total cost is $|X| + |Y_I| + |Y_R| \leq n + (n -$

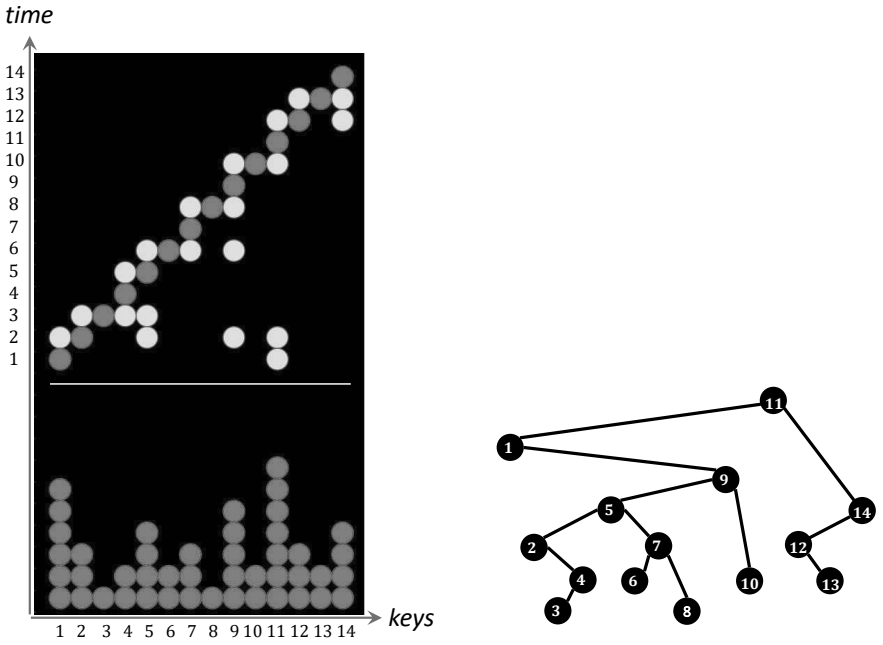


Figure 4.7. Left: an example of Greedy on sequential access. The red points below the line represent the initial tree. The red points above the line represent the input points. The yellow points are the non-accessed keys that are touched each time. For example, at time $t = 2$, key 2 is input, and keys 11,1,9,5,2 are touched. Right: the corresponding initial tree. The figure is taken from the software credit to Thatchaphol Saranurak.

Remark. The analysis can be modified when we use the Greedy execution log as a starting point instead of the initial tree by using almost the same argument.

4.5 Discussion

Our potential function is based on the idea of inversion, which measures the distance between the two states of algorithms. The accounting method makes it illustrative and allows us to modify it to prove stronger results. It would be intriguing to explore the potential applicability of this approach to other data structures. In the context of BST, some important related open questions are, for example, proving the lazy finger bound on Splay trees. This seems to require more understanding of Splay variant of external keys, which will allow us to upper bound the number of added coins per access. Another open question is to compare Greedy or Splay to other dynamic trees (or even compare them to each other). With our technique, the reference trees are limited to those trees that restructure only the keys that Greedy or Splay also touch.

Part II

Pattern Avoidance

5. Forbidden submatrices & BST

Greedy is one of the candidates for being dynamically optimal. For almost every n-key input sequence of length m, the offline optimal cost is $\Theta(m \log n)$ [10]. This means that it can be achieved by any balanced static BST. In the previous part, we have shown that Greedy is statically optimal. So, Greedy is optimal for almost every input sequence. However, for those input sequences in which the offline optimal cost is $o(m \log n)$, Greedy has to exploit the structure of the sequence. In previous chapters, amortized analysis is done via the potential function method, which would be easier when the algorithm designer is allowed to tailor an algorithm towards a tentative analytical method they have in mind. However, in the context of analyzing Greedy or Splay, the algorithms are already fixed in advance (e.g. these are the algorithms that tend to work well in practice), so we have no control over the "design" part. In such cases, the state-of-the-art understanding of potential function design is much more ad-hoc and mostly tailored to specific cases. Indeed, there has been no systematic, efficient way known for the task of designing a potential function, when an algorithm is fixed in advance. The fact that the aforementioned conjectures have remained open for decades clearly underlines the lack of understanding on this front.

In this chapter, we set up all the components in order to use extremal combinatorics in an amortized analysis of BSTs. The technique exploits the fact that the input of BSTs has some specific structures. Roughly speaking, this method encodes the execution log of BSTs in the form of a matrix and applies powerful theorems from forbidden matrix literature to upper bound the cost. This chapter is partly taken from [18].

5.1 Background

One can ask popular conjectures of BST in two settings: (1) when the initial BST can be pre-processed or (2) when it cannot be pre-processed. There is a gap in our understanding of these two settings. For example, it is not known if Greedy's cost for preorder traversal is the same in both

Avoid pattern(s)	Previous known	This thesis	Remark
$(2,1)$	$O(n)$[26]	$O(n)$	Sequential access
$(2,3,1)$ and $(2,1,3)$	$O(n2^{\alpha(n)})$ [13]	$O(n\alpha(n))$	Path sequence
$(2,3,1)$	$n2^{\alpha(n)^{O(1)}}$ [14]	$O(n2^{\alpha(n)})$	Preorder sequence
$(1,3,2)$	$n2^{\alpha(n)^{O(1)}}$ [14]	$O(n)$	Postoder sequence
$(3,2,1)$	$O(n)$ [14, 21]	-	2-increasing sequence

Table 5.1. Sequences that we consider and bounds for Greedy BSTs in this thesis. Note that these results are in the setting when the pre-processing of an initial tree is not allowed (setting (2)).

settings. In setting (1), Chalermsook et al. [14] showed that Greedy takes $O(n)$ for the preorder traversal. One can also solve this problem using the ideas in [32]. For Splay trees, if preprocessing is allowed, then Splay trees take $O(n)$ for the preorder traversal [20]. We will consider setting (2) in this part of the thesis.

In this setting, we usually consider the algorithm $\mathcal{A}$ on some specific sequence X such that $\mathsf{OPT}(X)$ is known so that we can immediately say that $\mathcal{A}$ can achieve the same bound as OPT (or refute the dynamic optimality conjecture if bounds are different[1]). Throughout this part, we only consider Greedy on some specific permutation sequence X. Let $\pi = (\pi_1,...,\pi_k)$ be a permutation of length k, we says that $X = (x_1,...,x_n)$ *contains* π if there is a subsequence X' of X such that X' is order-isomorphic to π. Two sequences are order-isomorphic if they have the same pairwise comparison, for example, $(2,6,3)$ is order-isomorphic to $(1,3,2)$. We say that X *avoids* π if X does not contain π. Note that X can avoid more than one sequence at the same time. Sequences X that we consider in this thesis are in Table 5.1.

In the remark column, we put the name of each family of sequences. When X avoids (2,1), the only possible sequence is $X = (1,2,...,n)$. Hence, this is called sequential access. When X avoids $(2,3,1)$ and $(2,1,3)$, the sequence becomes min-max sequence, i.e., each $x_t \in X$ is either minimum or maximum of those elements that were not previously accessed. For example, when $n = 5$, one possible X is $(1,5,4,2,3)$, which corresponds to access (min, max, max, min, min), respectively. Any min-max sequences can be obtained by traversing a *path* BST. A path BST is a BST where each node has at most one child. When X avoids $(2,3,1)$, there is a corresponding BST T_X such that a preorder traversal of T_X gives us X. The same goes for X that avoids $(1,3,2)$ and postorder traversal.

We note that, for all these sequences X, it is known that $\mathsf{OPT}(X) = O(n)$ [48, 32, 14, 28]. More literature can be found in Chapter 4 of [35]. Next, we will show that, for Greedy, these results already cover all sequences that avoid patterns of length 2 or 3.

[1]Refer to the dynamic optimality $O(\mathsf{OPT}(X)+n)$ version

5.2 Greedy

We consider input in the matrix X, that is, $X(i,j) = 1$ if and only if key i is accessed at time j. Notice that each matrix row contains exactly one 1-entry. Denote by $Y = G_T(X)$ the matrix corresponding to the execution log of Greedy on sequence X and initial tree T, that is, $Y(i,j) = 1$ if and only if key i is touched by Greedy on initial tree T at time j. We have $X \subseteq G_T(X)$ (Greedy always touches the input).

We explain how $G_T(X)$ is constructed. Inputs are matrix X (one point per row) and matrix T (n rows and n columns). The columns of $G_T(X)$ are $[n]$ and the rows of are indexed by $\{-(n-1),\ldots,0,1,\ldots,m\}$. The non-positive rows are exactly by T. Greedy starts adding points into $G_T(X)$ by processing rows $t = 1,\ldots,m$ in this order. At time t, we initialize $S \leftarrow \emptyset$. Let p be a point in X on t-th row. For each $a \in [n]$, if the rectangle $\square_{(a,\tau(a,t)),p}$ is empty, then we add point (a,t) to S. After we process all keys a, we add points in S to $G_T(X)$. See Figure 5.1 below.

We say that points in X are accessed and points in $G_T(X)$ are touched. Moreover, for point p in $G_T(X)$ we say that key $p.x$ is touched at time $p.y$. Throughout the paper, our statements hold for every initial tree T, hence we use $G(X)$ instead of $G_T(X)$.

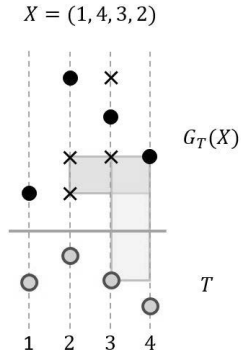


Figure 5.1. An example of $G_T(X)$ when $X = (1,4,3,2)$.

5.2.1 Symmetric on flip operation

For each matrix (point set) M with n columns, define the flipped matrix M^{flip} as a matrix M' obtained by reflecting M around y-axis, that is, $M'(i,j) = M(n-i+1,j)$ for all i,j. The following observation using the fact that Greedy is left-right symmetric:

Observation 5.1. *For any X and initial tree T, we have $(G_T(X))^{flip} = G_{T^{flip}}(X^{flip})$.*

This observation can be proved by induction on the number of rows.

Observation 5.2. *X avoids π if and only if X^{flip} avoids π^{flip}.*

This means that once we get the result for inputs that avoid $(2,1)$, we also get the result for inputs that avoid $(1,2)$. For the avoiding pattern of length 3, there are $3! = 6$ permutations. We know that all of them give a linear bound in n except $(2,1,3)$ and $(2,3,1)$. Interestingly, no linear bound in n is known even when we restricted the input to avoid both $(2,1,3)$ and $(2,3,1)$ (a path sequence).

5.3 Forbidden submatrix theory

5.3.1 Matrix and geometry

Let M be a binary matrix. For geometric reasons, we write matrix entries column-first, and the rows are ordered bottom-to-top, i.e. the first row is the bottom-most. Strictly speaking, $M(i,j)$ is the value of i-th column and j-th row of M. The matrix M can be interchangeably viewed as the set of points $\mathcal{P}(M)$ such that $(i,j) \in \mathcal{P}(M)$ if and only if $M(i,j) = 1$. We abuse the notation and sometimes write M (viewing M as both the matrix and the point set corresponding to 1-entries) instead of $\mathcal{P}(M)$. Denote by $|M|$ the number of 1s in M.

For point $p \in [n]\times[m]$, we use $p.x$ and $p.y$ to refer to the x and y-coordinates of p respectively. Let $I \subseteq [n]$ and $J \subseteq [m]$ sets of consecutive integers. We refer to $R = I \times J$ as a rectangle. We say that matrix M is *empty* in rectangle R if $M(i,j) = 0$ for all $(i,j) \in R$; or equivalently, the rectangle R is *M-empty*.

Let $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(k))$ be a permutation. We can view any permutation σ as a matrix M_σ where $M_\sigma(\sigma(i), i) = 1$ and other entries are zero.

5.3.2 Pattern avoidance

We say that matrix M contains pattern P if P can be obtained by removing rows, columns and non-zero entries of M. If M does not contain P, we say that M avoids P. See Figure 5.2.

$$\pi_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad \pi_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5.2. An example that M contains pattern π_1, but avoids π_2.

The theory of forbidden submatrices focuses on understanding the following extremal bound: $\mathrm{ex}(n,P)$ is defined as the maximum number of

1-entries in an n-by-n matrix that avoids P. For readability, zero entries in P are left as blank in writing.

We will use the following known bounds from [27, 42]:

Theorem 5.3 ([27, 42])**.** *(i)* $\mathrm{ex}\left(n, \begin{bmatrix} & 1 \\ 1 & \end{bmatrix}\right) = O(n)$.

(ii) $\mathrm{ex}\left(n, \begin{bmatrix} & 1 & \\ 1 & & 1 \end{bmatrix}\right) = O(n\alpha(n))$.

(iii) $\mathrm{ex}\left(n, \begin{bmatrix} & & 1 \\ & 1 & \\ 1 & & \end{bmatrix}\right) = O(n\alpha(n))$.

(iv) $\mathrm{ex}\left(n, \begin{bmatrix} & 1 & & \\ 1 & & 1 & \end{bmatrix}\right) = O(n2^{\alpha(n)})$.

5.4 Gadgets

We extend the pattern avoidance notation to handle multiple types of points. Let "•" denote each input point in X and "×" each point in $G(X) \setminus X$.

The notation $Y = G(X)$ contains $\begin{matrix} t_2 \\ t_1 \end{matrix}\!\!\begin{matrix} a & b \end{matrix}\begin{bmatrix} \bullet & \\ & \times \end{bmatrix}$ is used in the most intuitive way: Entry $Y(a, t_2)$ contains an input, and $Y(b, t_1)$ contains a touched point (or equivalently, b is touched at time t_1).

5.4.1 Formality of multi-typed pattern avoidance

For convenience, we extend the pattern avoidance terminology to allow points to have different types. Let M be a point set and $\mathbb{T}$ be the set of type of points. A type function of M is a mapping $f : M \to \mathbb{T}$. In the matrix view, the type f assigns a value in $\mathbb{T}$ to each non-zero entry of M.

Let M be a matrix and P a pattern. Let μ and π be types of M and P respectively. We say that (M, μ) contains (P, π) if and only if

- M contains P or there exists a submatrix M' obtained by removing rows, columns, and points of M such that $M' = P$. Also, $\pi' : M' \to \mathbb{T}$ be the type function π induced on M'.
- For all $(i,j) \in P$, we have $\pi'(i,j) = \mu(i,j)$.

In this thesis, our types are $\mathbb{T} = \{\times, \bullet\}$ where $\times$ and $\bullet$ are the touched (but non-accessed) and accessed points respectively. A Greedy matrix $G(X)$ is naturally associated with a type function $f : G(X) \to \mathbb{T}$ which assigns $\times$ to points in $G(X) \setminus X$ and $\bullet$ to points in X. Therefore, in the statement

$$(G(X), f) \text{ avoids } \begin{bmatrix} & \times \\ \bullet & \end{bmatrix}$$

we will often omit the types and simply say $G(X)$ instead of $(G(X), f)$.

If M contains a k-by-q pattern P, then there exists columns $c_1 \le \ldots \le c_q$ and rows $r_1 \le \ldots \le r_k$ such that the induced submatrix of M on those contains P. In such case, we use the following notation to specify such rows and columns where the pattern appears:

$$M \text{ contains } \begin{array}{c} \\ r_k \\ \vdots \\ r_1 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & \ldots & c_q \end{array} \\ \begin{bmatrix} & & \\ & P(i,j) & \\ & & \end{bmatrix} \end{array}$$

5.4.2 Input-revealing properties of Greedy

We use a small matrix gadget that allows us to "reveal" the location of an input point in X. We name some specific matrices as follows:

$$\wedge = \begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} & \times & \\ \times & & \times \end{bmatrix} \end{array} \qquad \rhd = \begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} & \times \\ \times & \end{bmatrix} \end{array} \qquad \trianglerighteq = \begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} & \times \\ \bullet & \end{bmatrix} \end{array}$$

$$\lhd = \begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \times & \\ & \times \end{bmatrix} \end{array} \qquad \trianglelefteq = \begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \times & \\ & \bullet \end{bmatrix} \end{array}$$

Proposition 5.4 (Generic Capture Gadget [14])**.** *If $G(X)$ contains $\wedge$, then the input matrix X is non-empty in the rectangle $[a+1, c-1] \times [t_1+1, t_2]$.*

Proposition 5.5 (One-sided Capture Gadget)**.** *If $G(X)$ contains $\rhd$ or $\trianglerighteq$, then input matrix X is non-empty in rectangle $[a+1,\infty)\times[t_1+1,t_2]$. This holds symmetrically for $\lhd$ and $\trianglelefteq$.*

Proof. Let (c,t') be a touched point in the rectangle $[a+1,\infty)\times[t_1+1,t_2]$ with smallest t'. We will show that X is non-empty in the rectangle $[a+1,\infty)\times[t_1+1,t']$, which will imply the Claim. Assume the rectangle $[a+1,\infty)\times[t_1+1,t']$ is input-empty. Let p be an input point at time t' such that $p.x \le a$. Since c is touched at time t', the rectangle $[p.x,c]\times[\tau(c,t'),p.y]$ must be empty. This contradicts to the fact that $(a,t_1)\in[p.x,c]\times[\tau(c,t'),p.y]$. □

Figure 5.3. Illustrations of Proposition 5.4 (left) and Proposition 5.5 (right).

Corollary 5.6. *If $G(X)$ contains $\rhd$ or $\trianglerighteq$, then input matrix X is non-empty in rectangle $([b,\infty)\times[t_1+1,t_2])$ or in $([a+1,\infty)\times[t_2,t_2])$. This holds symmetrically for $\lhd$ and $\trianglelefteq$.*

Proof. Assume the input matrix X is empty in the rectangle $([b,\infty)\times[t_1+1,t_2])$ and in the rectangle $([a+1,\infty)\times[t_2,t_2])$. Let (c,t') be the top most input point in $[a+1,b-1]\times[t_1+1,t_2-1]$. From Proposition 5.5, X must be non-empty in $[a+1,b-1]\times[t_1+1,t_2-1]$. This contradicts to the fact that (c,t') is the topmost input point. □

6. Matrix decomposition

Extremal combinatorics methods (such as forbidden submatrix theory) have been used successfully in the amortized analysis as an alternative to potential function design. In the context of binary search trees, such attempts were pioneered by Pettie [40, 41] and more recently extended by [13, 14]. Informally speaking, one can encode an execution log of Greedy as a binary matrix, then exploit the fact that the execution log of Greedy is restricted in some specific ways, and thus apply the extremal bounds from the forbidden submatrix theory. The connection between BSTs and the theory of forbidden matrices (see, e.g., in [13, 14]) relies on a "reduction statement", which says that if X avoids a pattern π of size k, then the Greedy matrix $G(X)$ avoids a (tensored) pattern π' of size $3k$. Therefore, existing extremal bounds can be immediately used to upper bound $|G(X)|$. For example, for preorder and postorder traversals (that avoid patterns of size 3), $G(X)$ avoids a pattern of size 9, which gives the bound of $n2^{\alpha(n)^{O(1)}}$ for $|G(X)|$.

In this chapter, we show applications of the technique introduced in the last chapter by deriving improved bounds for sequences in Table 5.1. We start with sequential access as a warm-up. Then, we show how to use our decomposition techniques to prove the results from path sequence, preorder sequence, and postorder sequence. The key to all these results is to decompose (based on the input structures) the execution log of Greedy into several simpler-to-analyze subsets for which classically forbidden submatrix bounds can be leveraged. We believe that this simple method will find further applications in doing amortized analysis of data structures via extremal combinatorics. This chapter is partly taken from [18].

Define $\mathrm{gex}(n,\{P_1,...,P_k\})$ as the maximum execution cost of Greedy $G(X)$ over all permutation input X that avoids every pattern $P_1,...,P_k$ and initial trees T. We abuse the notation as $\mathrm{gex}(n,P)$ when X avoids one pattern P. Here, we state the known reductions.

Lemma 6.1 ([13, 14])**.** *Let* $Q_0 = \begin{bmatrix} 1 & & 1 & & 1 \\ & 1 & & 1 & \end{bmatrix}$*, and let* Q_1 *and* Q_2 *be the following matrices.*

$$Q_1 = \begin{bmatrix} & 1 & & & & & & & \\ 1 & & 1 & & & & & & \\ & & & & & & & 1 & \\ & & & & & & 1 & & 1 \\ & & & & 1 & & & & \\ & & & 1 & & 1 & & & \end{bmatrix}, \text{ and } Q_2 = \begin{bmatrix} & & & & 1 & & & & \\ & & & 1 & & 1 & & & \\ & & & & & & & 1 & \\ & & & & & & 1 & & 1 \\ & & 1 & & & & & & \\ 1 & & & & 1 & & & & \end{bmatrix}.$$

(we omit zero entries for clarity).

- *If* X *is a path sequence, then* $G(X)$ *avoids* Q_0*. Therefore,* $|G(X)| \le \mathrm{ex}(n, Q_0) = O(n2^{\alpha(n)})$*.*
- *If* X *is preorder traversal sequence, then* $G(X)$ *avoids* Q_1*. Therefore,* $|G(X)| \le \mathrm{ex}(n, Q_1) \le n2^{\alpha(n)^{O(1)}}$*.*
- *If* X *is postorder traversal sequence, then* $G(X)$ *avoids* Q_2*. Therefore,* $|G(X)| \le \mathrm{ex}(n, Q_2) \le n2^{\alpha(n)^{O(1)}}$*.*

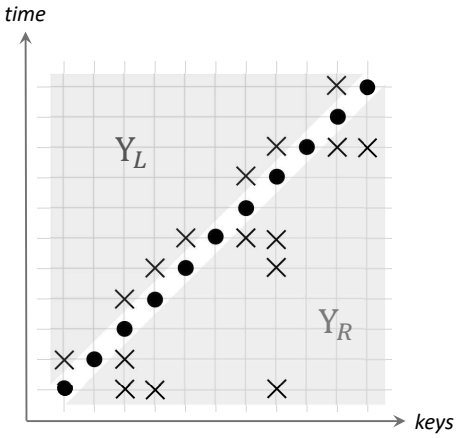
6.1 Sequential access

The cost of Greedy on the sequential access sequences is known to be $O(n)$[26]. In this thesis, we already showed the bound of $3n-2$. Here we will show the bound of $O(n)$ using pattern avoiding technique. A key idea of this proof is similar to Theorem 4.13 of [35].

Theorem 6.2. $\mathrm{gex}(n, (2,1)) = O(n)$*.*

Let X be a sequence that avoids $(2,1)$ (equivalently, X is the permutation $(1,2,\ldots,n)$) and $G(X)$ be the Greedy points on X. Notice that the points in X lie on the diagonal line $x = y$. Decompose $G(X)$ into $X \cup Y_L \cup Y_R$ where $Y_L = \{q \in G(X) \mid q.y > q.x\}$ and $Y_R = \{q \in G(X) \mid q.y < q.x\}$. In words, the sets Y_L and Y_R are the points strictly on the left and right of the diagonal line respectively. See Figure 6.1.

Lemma 6.3. Y_L *avoids* $\begin{bmatrix} \times & \\ & \times \end{bmatrix}$*, so* $|Y_L| \le O(n)$*.*



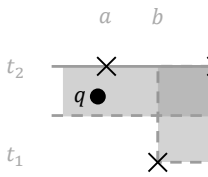


Figure 6.1. An example of $G(X)$ on sequential access.

Proof. Assume otherwise that Y_L contains $\begin{matrix} & a & b \\ t_2 & \times & \\ t_1 & & \times \end{matrix}$ for some time indices $t_1 < t_2$ and keys $a < b$. Applying Proposition 5.5, there exists input point q in the region $(-\infty, b-1] \times [t_1+1, t_2]$. Let p be an input point at time t_1, so $p.x > b$. Notice that q and p form $(2,1)$ a contradiction. □

Lemma 6.4. *Y_R avoids* $\begin{bmatrix} \times & & \times \\ & \times & \end{bmatrix}$. *Therefore, $|Y_R| = O(n)$.*

Proof. Assume otherwise that Y_R contains $\begin{matrix} & a & b & c \\ t_2 & \times & & \times \\ t_1 & & \times & \end{matrix}$ for some time indices $t_1 < t_2$ and keys $a < b < c$. Let q denote an input point at time t_2. Because $(a, t_2) \in Y_R$, we have that $q.x < a$. Applying Proposition 5.5 and the fact that Y_R contains $\begin{matrix} & b & c \\ t_2 & & \times \\ t_1 & \times & \end{matrix}$, we would have that the rectangular region $[b+1, \infty) \times [t_1+1, t_2]$ must contain input point p, which cannot be the same point as q; so we have $p.y < q.y$. The points p and q induce pattern $(2,1)$, a contradiction. See Figure 6.2 for illustration. □

Figure 6.2. Illustrations of the proofs for Lemma 6.4 (left) and Lemma 6.9 (right).

6.2 Path sequence

In a deque sequence, one allows to insert or delete minimum or maximum elements. Path sequence is equivalent to delete-only deque sequence.

Theorem 6.5. $\mathrm{gex}(n, \{(2,3,1),(2,1,3)\}) = O(n\alpha(n))$.

Let X be an input permutation that avoids both $(2,3,1)$ and $(2,1,3)$. Let $r \in X$ be an input point on the top row of X. We decompose $G(X)$ into $X \cup G_< \cup G_> \cup G_=$ where

- $G_< = \{q \in G(X) \setminus X \mid q.x < r.x\}$
- $G_> = \{q \in G(X) \setminus X \mid q.x > r.x\}$
- $G_= = \{q \in G(X) \setminus X \mid q.x = r.x\}$

To highlight the structure of this sequence, we similarly break X into $X = \{r\} \cup X_< \cup X_>$ where $X_< = \{p \in X : p.x < r.x\}$ and $X_> = \{p \in X : p.x > r.x\}$. See I

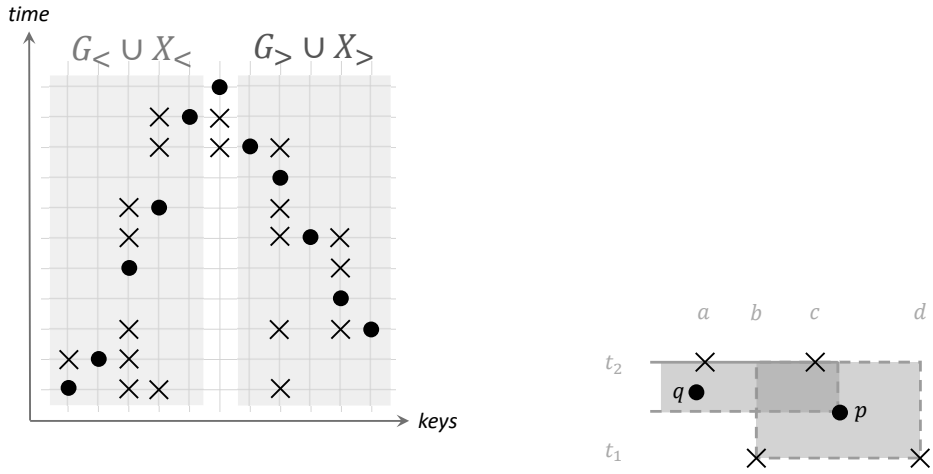


Figure 6.3. An example of $G(X)$ on path sequence.

Observation 6.6. *The points in $X_<$ and $X_>$ avoid $(2,1)$ and $(1,2)$ respectively.*

This means that the input points in $X_<$ form an increasing sequence, while those in $X_>$ form a decreasing sequence. Intuitively, we view input points as a triangle without a base, then we partition the plane into the middle vertical column, left side, and right side of the triangle.

Observation 6.7. $|G_=| \le n$.

Next we will show that $|G_<| \le O(n\alpha(n))$. The left-right symmetric arguments also hold for upper bounding $|G_>|$. We further decompose $G_<$ into $Y_L \cup Y_R$ where $Y_L = \{q \in G_< \mid \exists s \in X \text{ with } (q.y = s.y) \wedge (q.x < s.x < r.x)\}$ and $Y_R = G_< \setminus Y_L$. In words, the sets Y_L and Y_R are the points that are "outside" and "inside" the triangle respectively.

Observation 6.8. Y_L *avoids* $\begin{bmatrix} \times & \\ & \times \end{bmatrix}$, *so* $|Y_L| \le O(n)$.

Proof. We use exactly the same arguments as Lemma 6.3. □

Lemma 6.9. Y_R *avoids* $\begin{bmatrix} \times & & \times & \\ & \times & & \times \end{bmatrix}$. *Therefore,* $|Y_R| = O(n\alpha(n))$.

Proof. Assume otherwise that Y_R contains $\begin{matrix} & a & & b & & c & & d \\ t_2 & \multicolumn{7}{c}{\left[\begin{matrix} \times & & & & \times & & & \\ & & \times & & & & & \times \end{matrix}\right.} \\ t_1 & & & & & & & & \end{matrix}$ for some time indices $t_1 < t_2$ and keys $a < b < c < d$. Applying Proposition 5.4 and the fact that Y_R contains $\begin{matrix} & b & & c & & d \\ t_2 & \multicolumn{5}{c}{\left[\begin{matrix} & & \times & & \\ \times & & & & \times \end{matrix}\right.} \\ t_1 & & & & & \end{matrix}$, we would have that the rectangular region $[b+1, d-1] \times [t_1+1, t_2]$ must contain input point $p \in X_<$. Because $(a, t_2) \in Y_R$, we have that $p.y < t_2$. Applying Proposition 5.5 and the fact that Y_R contains $\begin{matrix} & a & & p.x \\ t_2 & \multicolumn{3}{c}{\left[\begin{matrix} \times & & \\ & & \bullet \end{matrix}\right.} \\ p.y & & & \end{matrix}$, we can conclude that the rectangle $(-\infty, p.x - 1] \times [p.y+1, t_2]$ contains an input point $q \in X_<$. The points p and q induce pattern $(2,1)$, a contradiction. □

6.3 Preorder traversal sequence

In this section, we prove the following theorem.

Theorem 6.10. $\mathrm{gex}(n, (2,3,1)) = O(n2^{\alpha(n)})$.

Let X be an input matrix that corresponds to a permutation that avoids $(2,3,1)$. We partition the Greedy points Y into four parts based on the location of the points with respect to input points.

Observation 6.11. *For each point $q \in Y \setminus X$, there are (unique) input points $p_1, p_2 \in X$ such that $p_1.x = q.x$ and $p_2.y = q.y$.*

Using this observation, for each such point q, if $p_1.y < q.y$, we say that q is a bottom point; otherwise, we say that it is a top point, so this will partition $Y \setminus X$ into $T \cup B$ where T and B are the top and bottom points respectively. Similarly, we define the left/right partition $Y \setminus X = L \cup R$ where L and R are the left and right points.

These can be used to define our partition as follows: $BR = B \cap R, BL = B \cap L, TR = T \cap R, TL = T \cap L$.

Lemma 6.12. *BR avoids* $\begin{bmatrix} \times & \\ & \times \end{bmatrix}$. *Therefore, $|BR| = O(n)$.*

Proof. Assume otherwise that BR contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \times & \\ & \times \end{bmatrix} \end{array}$ for some time indices $t_1 < t_2$ and keys $a < b$. Since the point (b, t_1) is a touched point in BR, there are input points p and q that are at the bottom and right of (b, t_1) respectively. From Proposition 5.5, the region $(-\infty, b-1] \times [t_1+1, t_2]$ must contain an input point r. The points p, q and r induce pattern $(2,3,1)$ in X, a contradiction. □

Lemma 6.13. *BL avoids* $\begin{bmatrix} \times & & \times \\ & \times & \end{bmatrix}$. *Therefore, $|BL| = O(n)$.*

Proof. Assume otherwise that BL contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} \times & & \times \\ & \times & \end{bmatrix} \end{array}$ for some time indices $t_1 < t_2$ and keys $a < b < c$. Let p be an input point at the bottom of (b, t_1). Let r be an input point on the left of (a, t_2). Applying Proposition 5.5 and the fact that BL contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array} \begin{array}{c} \begin{array}{cc} b & c \end{array} \\ \begin{bmatrix} & \times \\ \times & \end{bmatrix} \end{array}$, we would have that the rectangular region $[b+1, \infty) \times [t_1+1, t_2]$ must contain input point q. Notice that $q \neq r$, so we have that p, q and r induce pattern $(2,3,1)$ in X, a contradiction. □

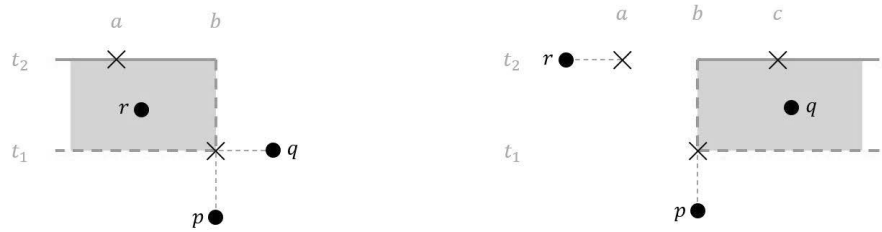


Figure 6.4. Illustrations of the proofs for *BR* (left) and *BL* (right) in preorder.

Lemma 6.14. *$TL \cup TR$ avoids* $\begin{bmatrix} \times & & \times \\ & \bullet & \end{bmatrix}$.

Proof. Assume otherwise that $TL \cup TR$ contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} \times & & \times \\ & \bullet & \end{bmatrix} \end{array}$. Let r be an input point at the top of (a, t_2). Applying Proposition 5.5 and the fact that $TL \cup TR$ contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{cc} b & c \end{array} \\ \begin{bmatrix} & \times \\ \bullet & \end{bmatrix} \end{array}$, we can conclude that the rectangle $[b+1, \infty) \times [t_1+1, t_2]$ contains an input point q. Let $p = (b, t_1)$. Notice that p, q and r induce pattern $(2,3,1)$, a contradiction. See Figure 6.5. □

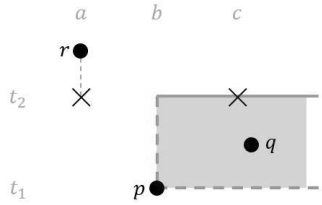


Figure 6.5. An illustration of the proof for $TL \cup TR$ in preorder.

Corollary 6.15. *Each TL and TR avoids* $\begin{bmatrix} \times & & \times & & \times \\ & \times & & \times & \end{bmatrix}$. *Therefore,* $|TL| + |TR| \leq O(n2^{\alpha(n)})$.

Proof. We only present the proof for TL; the arguments for TR are symmetric. Assume otherwise that TL contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} \times & & \times & & \times \\ & \times & & \times & \end{bmatrix} \end{array}$. Applying

Proposition 5.4 to the pattern $\begin{matrix} & b & c & d \\ t_2 & & \times & \\ t_1 & \times & & \times \end{matrix}$, there must be an input point $q \in [b+1, d-1] \times [t_1+1, t_2]$. If $q.y < t_2$, we would be done since it would contradict Lemma 6.14, so assume that $q.y = t_2$. Since $a < q.x < e$, this implies that $(a, t_2) \in R$, a contradiction. □

6.4 Postorder traversal sequence

In this section, we prove the following theorem.

Theorem 6.16. $\mathrm{gex}(n, (1,3,2)) = O(n)$.

Let X be a permutation that avoids $(1,3,2)$. We partition the Greedy points $G(X)$ into four sets BL, BR, TL, TR in the same way as in the last section.

Lemma 6.17. *BR avoids* $\begin{bmatrix} & \times & \\ \times & & \times \end{bmatrix}$. *Therefore,* $|BR| \leq O(n)$.

Proof. Assume otherwise that BR contains $\begin{matrix} & a & b & c \\ t_2 & & \times & \\ t_1 & \times & & \times \end{matrix}$ for some time indices $t_1 < t_2$ and keys $a < b < c$. Let $p = (a, t_0)$ be an input at the bottom of (a, t_1). Let $q = (c', t_1)$ be an input on the right of (c, t_1). Using the Proposition 5.4, there must be an input point r in the rectangle $[a+1, c-1] \times [t_1+1, t_2]$. The points p, q and r induce pattern $(1,3,2)$ in X, a contradiction. □

Lemma 6.18. *BL avoids* $\begin{bmatrix} & \times & \\ \times & & \times \end{bmatrix}$. *Therefore,* $|BL| \leq O(n)$.

Proof. Assume for contradiction that BL contains $\begin{matrix} & a & b & c \\ t_2 & & \times & \\ t_1 & \times & & \times \end{matrix}$ for some time indices $t_1 < t_2$ and keys $a < b < c$. Let $p = (a, t_0)$ be the input point at the bottom of (a, t_1). Applying Corollary 5.6 with the submatrix $\begin{matrix} & a & c \\ t_1 & & \times \\ t_0 & \bullet & \end{matrix}$, it follows that there exists input point q in the rectangle $[c, \infty) \times [t_0+1, t_1]$ or $([a+1, \infty) \times [t_1, t_1])$. Since an input point at time t_1 has to be on the left of a,

This means q must be in the rectangle $[c,\infty)\times[t_0+1,t_1-1]$. Finally, applying Proposition 5.4, there exists input point r in the rectangle $[a+1,c-1]\times[t_1+1,t_2]$. The points p,q and r induce $(1,3,2)$ in X, a contradiction. □

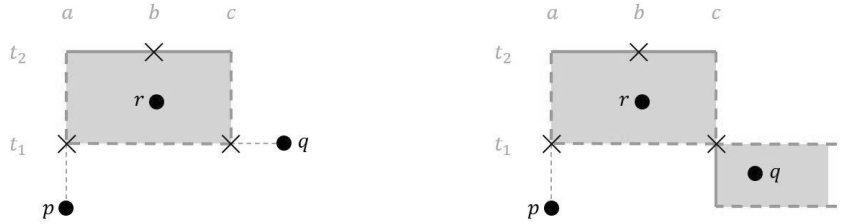


Figure 6.6. Illustrations of the proofs for *BR* (left) and *BL* (right) in postorder.

Lemma 6.19. *TR avoids* $\begin{bmatrix} \times & & \times \\ & \times & \end{bmatrix}$. *Therefore,* $|TR| \le O(n)$.

Proof. Assume otherwise that TR contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} \times & & \times \\ & \times & \end{bmatrix} \end{array}$ for some time indices $t_1 < t_2$ and keys $a < b < c$. Since (c,t_2) is in TR, there are input points q and r at the right and top of it respectively. Applying Proposition 5.5 to the submatrix $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \times & \\ & \times \end{bmatrix} \end{array}$, there must be an input point p in the region $(-\infty,b]\times[t_1+1,t_2]$. Since X is a permutation, we have that $p.y < q.y$ (in particular, $p \neq q$). The points p,q and r induce $(1,3,2)$ in X, a contradiction. □

Lemma 6.20. *TL avoids* $\begin{bmatrix} \times & \times \\ \times & \end{bmatrix}$. *Therefore,* $|TL| \le O(n)$.

Proof. Assume otherwise that TL contains $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \times & \times \\ \times & \end{bmatrix} \end{array}$ for time indices $t_1 < t_2$ and keys $a < b$. Let r be an input at the top of (a,t_2), and p be the input at the left of (a,t_1). Using Proposition 5.5 for the submatrix $\begin{array}{c} \\ t_2 \\ t_1 \end{array}\!\!\begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} & \times \\ \times & \end{bmatrix} \end{array}$, there must exist point q in the region $[b,\infty)\times[t_1+1,t_2]$. The points p,q and r induce $(1,3,2)$ in X, a contradiction. □

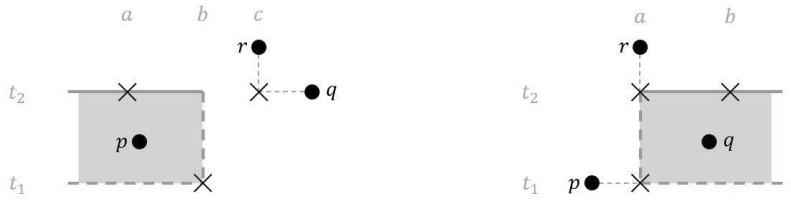


Figure 6.7. Illustrations of the proofs for TR (left) and TL (right) in postorder.

6.5 Discussion

We show the usefulness of the matrix decomposition technique. Here, we show examples of submatrix that $G(X)$ contains if we do not partition it into many parts. In Figure 6.8 (left), the points (a,t_2),(b,t_1),(c,t_2) and ,(d,t_1) form the pattern $\begin{bmatrix} 1 & & 1 & \\ & 1 & & 1 \end{bmatrix}$. In Figure 6.8 (right), the points

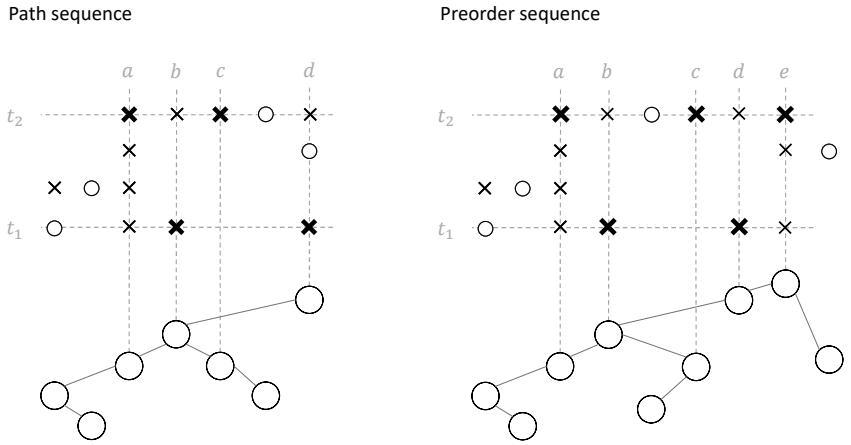


Figure 6.8. (Left) Greedy on path sequence with initial tree below time t_1. (Right) Greedy on preorder traversal sequence with initial tree below time t_1.

The analysis of our preorder result shows that the bottleneck is on the set $TL \cup TR$, which contains all possible patterns of size four. This means we cannot achieve better bound with this way of decomposition. Therefore, it is essential to explore various approaches to decomposition that can alleviate the bottleneck and enable us to achieve an improved bound.

References

[1] M AdelsonVelskii and Evgenii Mikhailovich Landis. An algorithm for the organization of information. Technical report, JOINT PUBLICATIONS RESEARCH SERVICE WASHINGTON DC, 1963.

[2] Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

[3] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[4] Susanne Albers, Bernhard Von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.

[5] Brian Allen and Ian Munro. Self-organizing binary search trees. *Journal of the ACM (JACM)*, 25(4):526–535, 1978.

[6] Christoph Ambühl, Bernd Gärtner, and Bernhard Von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoretical Computer Science*, 268(1):3–16, 2001.

[7] Christoph Ambühl, Bernd Gärtner, and Bernhard von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31:1–31:18, 2013.

[8] Ran Bachrach, Ran El-Yaniv, and M. Reinstadtler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002.

[9] Jon Louis Bentley and Catherine C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Commun. ACM*, 28(4):404–411, 1985.

[10] Avrim Blum, Suchi Chawla, and Adam Tauman Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36(3):249–260, 2003.

[11] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

[12] Prosenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The power and limitations of static binary search trees with lazy finger. In *International Symposium on Algorithms and Computation*, pages 181–192. Springer, 2014.

[13] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. *CoRR*, abs/1506.08319, 2015.

[14] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 410–423. IEEE, 2015.

[15] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *Algorithms-ESA 2015*, pages 300–312. Springer, 2015.

[16] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *CoRR*, abs/1603.04892, 2016.

[17] Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Multi-finger binary search trees. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[18] Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy bsts via matrix decomposition. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 509–534. SIAM, 2023.

[19] Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[20] Ranjan Chaudhuri and Hartmut Höft. Splaying a search tree in preorder takes linear time. *ACM SIGACT News*, 24(2):88–93, 1993.

[21] Josef Cibulka. Extremal combinatorics of matrices, sequences and sets of permutations. 2013.

[22] Richard Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

[23] Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.

[24] Erik D Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Patraşcu. The geometry of binary search trees. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505. SIAM, 2009.

[25] Rebecca Fiebrink. Amortized analysis explained. *See http://www. cs. princeton. edu/~ fiebrink/423/AmortizedAnalysisExplained_Fiebrink. pdf*, 2007.

[26] Kyle Fox. Upper bounds for maximally greedy binary search trees. In *Workshop on Algorithms and Data Structures*, pages 411–422. Springer, 2011.

[27] Zoltán Füredi and Péter Hajnal. Davenport-schinzel theory of matrices. *Discrete Mathematics*, 103(3):233–251, 1992.

[28] Navin Goyal and Manoj Gupta. Better analysis of binary search tree on decomposable sequences. *Theoretical Computer Science*, 776:19–42, 2019.

[29] Leo J Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 8–21. IEEE, 1978.

[30] Torben Hagerup. Online and offline access to short lists. In *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, pages 691–702, 2007.

[31] Dion Dion Kane Harmon. *New bounds on optimal binary search trees*. PhD thesis, Massachusetts Institute of Technology, 2006.

[32] John Iacono and Stefan Langerman. Weighted dynamic finger in binary search trees. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 672–691. SIAM, 2016.

[33] Shahin Kamali and Alejandro López-Ortiz. *A Survey of Algorithms and Models for List Update*, pages 251–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

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

[35] László Kozma. Binary search trees, rectangles and patterns. 2016.

[36] Caleb Carson Levy and Robert Endre Tarjan. A new path from splay to dynamic optimality. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1311–1330. Society for Industrial and Applied Mathematics, 2019.

[37] Joan Marie Lucas. *Canonical forms for competitive binary search tree algorithms*. Rutgers University, Department of Computer Science, Laboratory for Computer Science Research, 1988.

[38] Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5(4):287–295, 1975.

[39] J Ian Munro. On the competitiveness of linear search. In *European Symposium on Algorithms*, pages 338–345. Springer, 2000.

[40] Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1115–1124, 2008.

[41] Seth Pettie. Applications of forbidden 0–1 matrices to search tree and path compression-based data structures. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1457–1467. SIAM, 2010.

[42] Seth Pettie. Generalized davenport–schinzel sequences and their 0–1 matrix counterparts. *Journal of Combinatorial Theory, Series A*, 118(6):1863–1895, 2011.

[43] Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. *Information Processing Letters*, 60(2):75–80, 1996.

[44] Nick Reingold, Jeffery Westbrook, and Daniel Dominic Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[45] Luís MS Russo. A study on splay trees. *Theoretical Computer Science*, 2018.

[46] Frank Schulz. Two new families of list update algorithms. In *International Symposium on Algorithms and Computation*, pages 100–109. Springer, 1998.

[47] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

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

[49] Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985.

[50] Boris Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.

[51] Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM journal on Computing*, 18(1):56–67, 1989.

Aalto-DT 77/2023

ISBN 978-952-64-1282-5 (printed)
ISBN 978-952-64-1283-2 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Computer Science
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL THESES