CSCI 6057
Advanced Data Structures

ASSIGNMENT - 3

Balaji Sukumaran (bl664064@dal.ca)

Banner ID: B0094897

Table of contents

1. [10 marks] Draw the x-fast trie that maintains the following set $S = \{0, 2, 8, 11, 14\}$ in universe $\{0, 1, 2, \ldots, 15\}$. Thus, $n = 5$ and $u = 16$.

   Your solution should include the tree structure, the descendant links, values stored in the leaves, and pointers between leaves.

   To show the dynamic perfect hash table constructed, simply give all the keys stored in the hash table.

Solution:

For the given universe u, the tree looks like the following

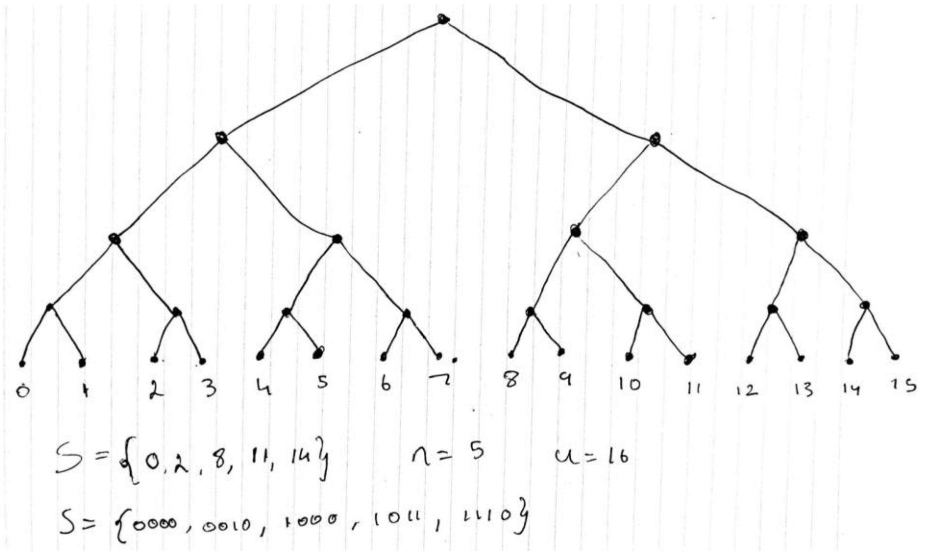


Figure 1: Depiction of universe in BST

In X-Fast Trie Data Structure:

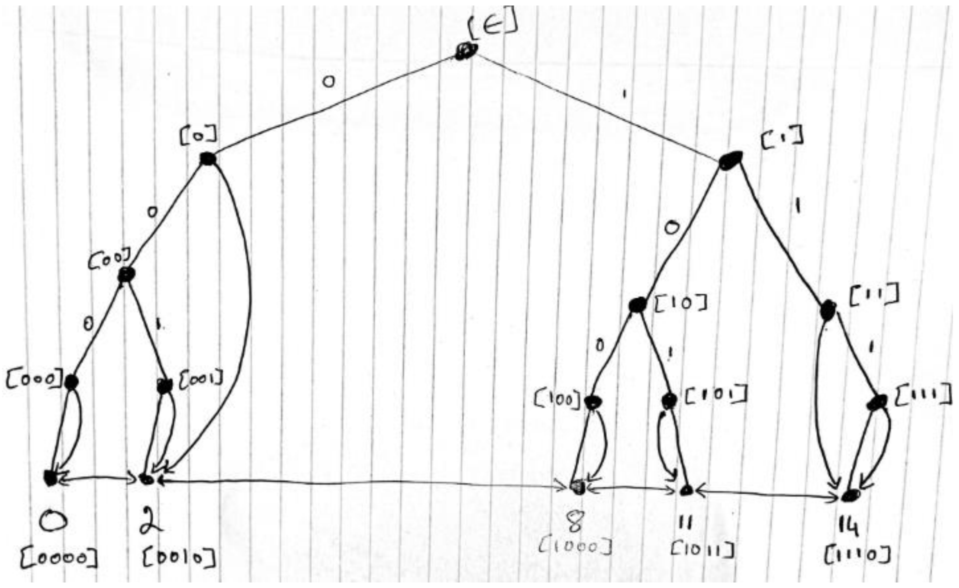


Figure 2: X-Fast Trie with descendent link and doubly linked list

1. The Binary tree stores only the path to the values in set S.
2. The keys are linked together in doubly linked list.
3. Hash Table which stores the pre-fixes as keys at every level.

   H = { ∈, 0, 1, 00, 10, 11, 000, 001, 100, 101, 111, 0000, 0010, 1000, 1011, 1110 }

2. [15 marks] This question asks you to perform competitive analysis of transpose (TR).

(i) [9 marks] Suppose that you are maintaining a list of n elements under access operation only. The cost of access to the i-th element in the list is i. Let S be a request sequence of m access operations over this list.

For any sufficiently large m, construct a request sequence S such that for this request sequence, the total cost of TR divided by the total cost of S_{opt} is $\omega(1)$.

Solution:

Let the initial state of the list be,

a_1, a_2, a_3, a_4,…….. a_n

In order to maximize the cost for TR list accesses, In our request sequence, we will access a_n first, which makes the system to swap a_n with a_{n-1}.

Following this we will access a_{n-1}, which will be swapped with a_n. We will repeat this cycle until there are m accesses in our request sequence.

Based on this our request sequence looks like this, S = {a_n, a_{n-1}, a_n, a_{n-1},……}.

TR cost for this request sequence is **mn** since we always look for the element in the **n**'th location **m** times.

***S_{opt}* cost for this request sequence**: For calculating this we only have to worry about the cost and order of a_n, a_{n-1}. Since only these two elements are there in our request sequence.

Case #1: If M is odd

$$\{a_n, a_{n-1}, a_n, a_{n-1} \cdots a_n\}$$

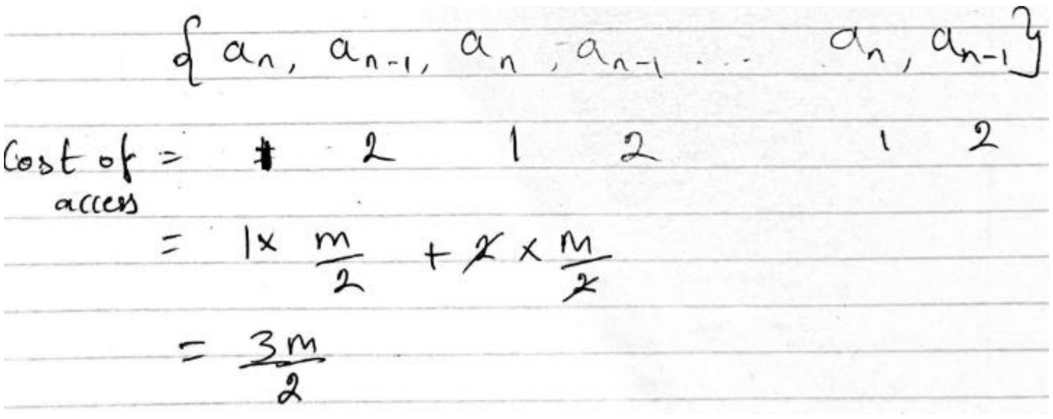
Cost of access = 1 2 1 2 1

$$= 1 \times \left\lceil \frac{m}{2} \right\rceil + 2 \times \left\lfloor \frac{m}{2} \right\rfloor$$

$$= \frac{m+1}{2} + 2\left(\frac{m-1}{2}\right)$$

$$= \frac{3m}{2} - \frac{1}{2}$$

Case #2: If M is even.



$$\{a_n, a_{n-1}, a_n, a_{n-1} \dots a_n, a_{n-1}\}$$

Cost of access $= \cancel{1} \quad 2 \quad 1 \quad 2 \quad 1 \quad 2$

$$= 1 \times \frac{m}{2} + \cancel{2} \times \frac{M}{\cancel{2}}$$

$$= \frac{3m}{2}$$

So, the total cost of TR is at least 3m/2.

Total cost of TR divided by S_{opt} = (mn)/(3m/2) = 2n/3 = $\omega(1)$

Here $\omega(1)$, indicates that the cost difference large as n grows, which implies that S_{opt} is asymptotically more efficient than TR for large n.

(ii) [6 marks] Use the result of (i) to argue that TR is not competitive.

Solution:

An online algorithm is considered competitive if its performance is within a constant factor of the optimal offline algorithm.

1. From part (i), Total cost C_{TR} for a sequence of request using TR is at least 2/3 times the cost of serving the same using an optimal static list algorithm, S_{opt}. As proved in (i) $C_{TR}/C_{S_{opt}}$ >= $\omega(1)$
2. The optimal static list algorithm S_{opt}, is not the best possible algorithm, a truly optimal offline algorithm, C_{opt}, which knows the entire sequence of requests in advance, can always perform at least as well as S_{opt}, so the cost of optimal offline algorithm is less than or equal to the cost of optimal static list algorithm C_{opt} <= $C_{S_{opt}}$.
3. Combining these points, if C_{TR} is at least $\omega(1)$ times $C_{S_{opt}}$ and C_{opt} is always less than or equal to $C_{S_{opt}}$, then it must be true that C_{TR} is atleast $\omega(1)$ times C_{opt}. In other words C_{TR}/C_{opt} >= $\omega(1)$.

Therefore, the competitive ratio of TR is not constant and instead grows with the number of requests. If an online algorithm is competitive, the ratio should remain bounded above by a constant factor, even as **m** grows.

3. [15 marks] The algorithm MTF_o for self-adjusting lists work as follows: Move the item to the front on every odd request for that item.

   That is, the first time an item is accessed, it is moved to the front of the list. The second time this same item is accessed, it is not moved at all (i.e., the list does not self adjust after this access). The third time this item is accessed, it is moved to the front of the list again. The fourth time this item is accessed, it is not moved...

   Similarly, the algorithm MTF_e move the item to the front on every even request for that item.

   It is known that both MTF_o and MTF_e are competitive. However, the proof is rather involved, so I am NOT asking you to prove it here.

   Instead, you are asked to prove the following two claims:

   (i) [10 marks] The competitive ratio of MTF_o is at least 5/2.

   Hint:

   – Given a list of n items, initially ordered as $a_1, a_2, \ldots, a_n$, it may be helpful to consider the following request sequence of length $8sn$ (for simplicity, you can assume that the length of a request sequence is a multiple of $8n$):

   $$(a_1, a_2, \ldots, a_n, a_1^3, a_2^3, \ldots, a_n^3, a_n, a_{n-1}, \ldots, a_1, a_n^3, a_{n-1}^3, \ldots, a_1^3)^s$$

   – It also helps to think of the following algorithm A: Do not self adjust when handling the first n requests, and use MTF for the next $3n$ requests. Repeat this pattern.

Solution:

<u>The Total cost when an **optimal offline algorithm** is used</u>

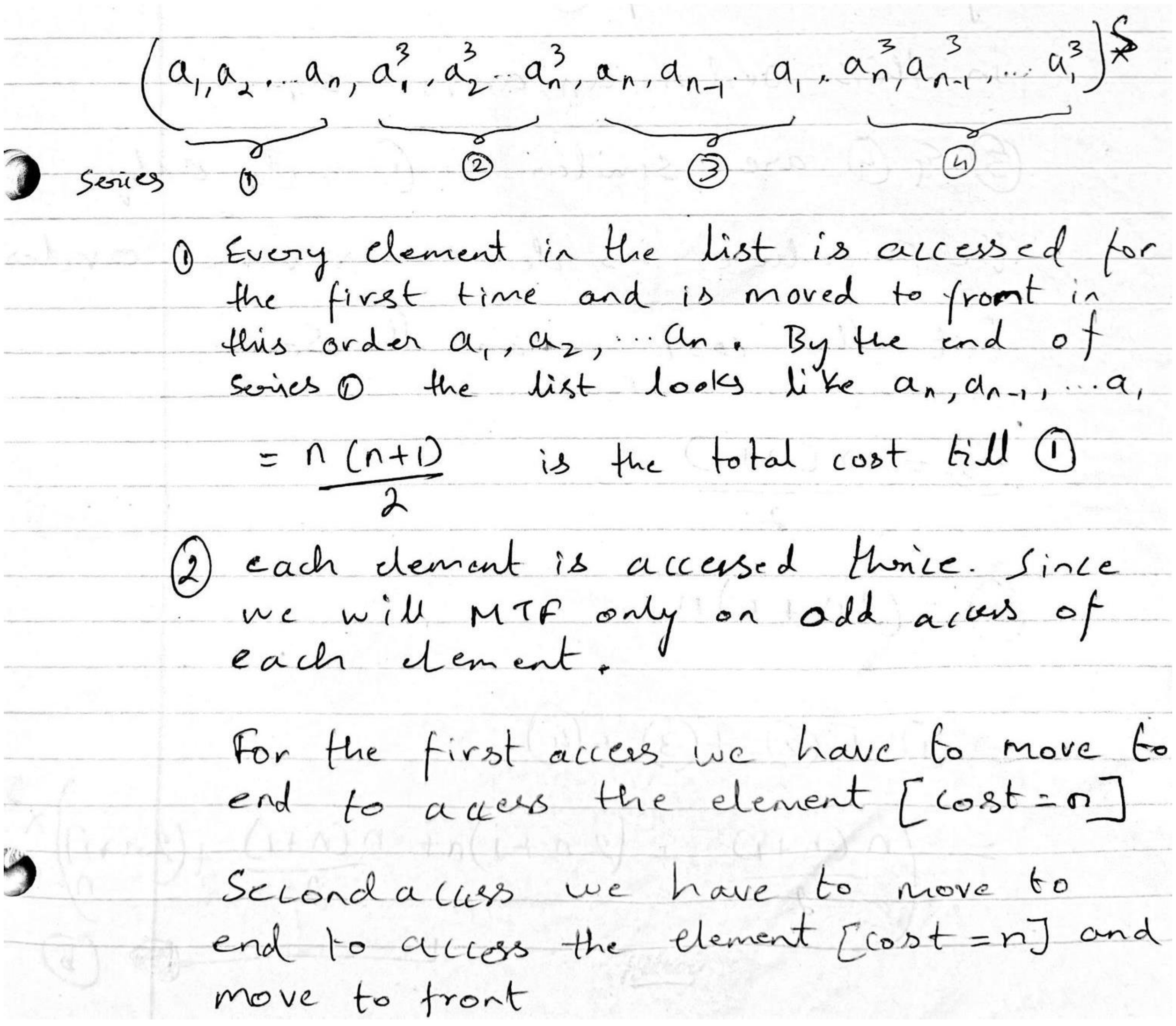
(1) Request Sequence :-

$$(\underbrace{a_1, a_2 \ldots a_n}, \underbrace{a_1^3, a_2^3 \ldots a_n^3}, \underbrace{a_n, a_{n-1} \ldots a_1}, \underbrace{a_n^3 a_{n-1}^3 \ldots a_1^3})^S$$

Cost of optimal offline algorithm $= \dfrac{n(n+1)}{2} + \dfrac{n(n+1)}{2} \times 3 + \dfrac{n(n+1)}{2} + \dfrac{n(n+1)}{2} \times 3$

$$= 8\,\frac{n(n+1)}{2} \times S \qquad \text{———} \; \textcircled{a}$$

The Total cost when **Move-To-Front on odd accesses** MTF_o is used



$$\left(\underbrace{a_1, a_2, \ldots a_n}_{\textcircled{1}}, \underbrace{a_1^3, a_2^3 \cdots a_n^3}_{\textcircled{2}}, \underbrace{a_n, a_{n-1} \cdots a_1}_{\textcircled{3}}, \underbrace{a_n^3, a_{n-1}^3, \ldots a_1^3}_{\textcircled{4}}\right)^{S}$$

Series

① Every element in the list is accessed for the first time and is moved to front in this order $a_1, a_2, \cdots a_n$. By the end of series ① the list looks like $a_n, a_{n-1}, \cdots a_1$

$= \frac{n(n+1)}{2}$ is the total cost till ①

② each element is accessed thrice. Since we will MTF only on odd access of each element.

For the first access we have to move to end to access the element [cost = n]

Second access we have to move to end to access the element [cost = n] and move to front

Third time the element will be in front
So, [Cost = 1]

~~t~~this is done for n times to access
n elements

$\textcircled{2} = (n+n+1)n$

$= (2n+1)n$

By the end of $\textcircled{2}$ the list will be

in this order $a_n, a_{n-1}, \dots a_1$

$\textcircled{3}$ & $\textcircled{4}$ are similar to $\textcircled{1}$ & $\textcircled{2}$ only

the procedure is done in reverse order

but the cost remains the same.

$$\textcircled{3} = \frac{n(n+1)}{2}$$

$$\textcircled{4} = (2n+1)n$$

$$= \textcircled{1} + \textcircled{2} + \textcircled{3} + \textcircled{4}$$

$$= \left(\frac{n(n+1)}{2} + (2n+1)n + \frac{n(n+1)}{2} + (2n+1)n\right) \times S$$

$\textcircled{b}$

$$= \left(\frac{n(n+1)}{2} + (2n+1)n + \frac{n(n+1)}{2} + (2n+1)n \right) \times S$$

$$= \left[n(n+1) + 2n(2n+1) \right] \times S \hookrightarrow ⓑ$$

In order for MTF$_0$ to be $\frac{5}{2}$ competitive the following rule should stand

$$C_{MTF_0}(s) \le \frac{5}{2} C_{opt}(s) \quad —— ⓒ$$

From ⓐ & ⓑ & ⓒ

$$\cancel{s} \times \left[n(n+1) + 2n(2n+1) \right] \le \frac{5}{\cancel{2}} \left[\frac{\overset{\cancel{4}\,2}{\cancel{8}}\, n(n+1)}{\cancel{2}} \right] \times \cancel{s}$$

$$n(n+1) + 2n(2n+1) \le 10n(n+1)$$

$$2\cancel{n}(2n+1) \le 9\cancel{n}(n+1)$$

$$4n+4 \le 9n+1$$

$$4n+3 \le 9n$$

$$-5n+3 \le 0$$

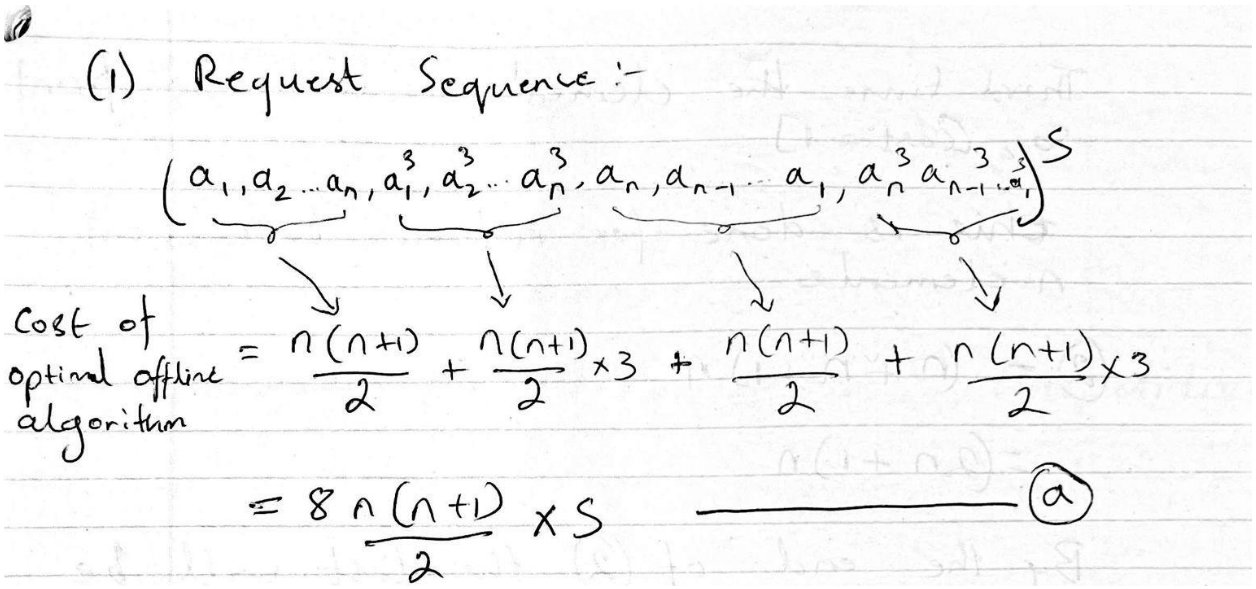
Since n cannot be less than 0, i[illegible] the equation ⓒ is true

(i) [5 marks] The competitive ratio of MTF_e is at least 5/2.

Hint: The simplest way of proving this is to prepend a subsequence of requests to that in the sequence given in the hint for the first subquestion.

Solution:

The Total cost when an **optimal offline algorithm** is used



(1) Request Sequence :-

$$\left(\underbrace{a_1, a_2 \dots a_n}, \underbrace{a_1^3, a_2^3 \dots a_n^3}, \underbrace{a_n, a_{n-1} \dots a_1}, \underbrace{a_n^3 a_{n-1}^3 \dots a_1^3}\right)^S$$

Cost of optimal offline algorithm $= \frac{n(n+1)}{2} + \frac{n(n+1)}{2} \times 3 + \frac{n(n+1)}{2} + \frac{n(n+1)}{2} \times 3$

$$= 8\,\frac{n(n+1)}{2} \times S \text{ ——— } \textcircled{a}$$

The Total cost when **Move-To-Front on even accesses** MTF_e is used

$$S$$

$$\Big(\underbrace{a_1, a_2 \cdots a_n}_{①},\ \underbrace{a_1^3, a_2^3 \cdots a_n^3}_{②},\ \underbrace{a_n, a_{n-1}, \cdots a_1}_{③},\ \underbrace{a_n^3, a_{n-1}^3 \cdots a_1^3}_{④}\Big)$$

① The total cost to access first request from Sequence is

$$= \frac{n(n+1)}{2}$$

but we wont be moving any elements in the list. Since every element is accessed only once

② First a_1 is accessed thrice. First access the element and move to front. Second time we access the element from front third access we will move to front again Since it is accessed for the fourth time.

For a_1 we can't imagine the Series but for a_2, a_3, a_4 we will be able to see how the cost changes.

$a_1^3 = 1 + 1 + 1$ $\qquad a_1, a_2, a_3 \ldots$

$a_2^3 = 2 + 1 + 1$ $\qquad a_2, a_1, a_3$

$a_3^3 = 3 + 1 + 1$ $\qquad a_3, a_2, a_1 \ldots$

$\vdots$

$a_n^3 = n + 1 + 1$ $\qquad a_n, a_{n-1} \cdots a_1$

Total cost for ②

$$= \frac{n(n+1)}{2} + 2n$$

③ & ④ is similar to ① & ② the cost also remains the same. Only difference is the request calls the elements from a_n, a_{n-1} to a_1.

Total cost combining ① ② ③ ④

$$= \underbrace{\left[\frac{n(n+1)}{2} + \frac{n(n+1)}{2} + 2n\right] \times 2 \times S}_{ⓑ}$$

In order for MTF_e to be $\frac{5}{2}$ competitive the following rule should stand

$$C_{MTF_e}(s) \leq \frac{5}{2} C_{opt}(s) \quad — ⓒ$$

from ⓐ , ⓑ , ⓒ

8~~[illegible]~~n

$$\left[\frac{n(n+1)}{2} + \frac{n(n+1)}{2} + 2n\right] \cancel{2}\$ \leq \frac{5}{\cancel{2}}\left[\overset{2}{\cancel{\overset{4}{\cancel{8}}}}\, \frac{n(n+1)}{2}\right] \cancel{\$}$$

$$\frac{n(n+1)}{2} + \frac{n(n+1)}{2} + 2n \leq 10\left(\frac{n(n+1)}{2}\right)$$

Let $a = \frac{n(n+1)}{2}$

$$2a + 2n \leq 10a$$

$$\cancel{2}n \leq \overset{4}{\cancel{8}}a$$

$$\cancel{n} \leq \frac{\cancel{n}(n+1)}{\cancel{2}} \times \cancel{4}^{2}$$

$$1 \leq 2(n+1)$$

Since n is always positive ⓒ stands true

4. [10 marks] In class, we discussed the problem of computing a static optimal binary search tree. Now we draw each binary search trees in a slightly different way by adding nodes representing failures in searches: We first draw the same tree structure. Then

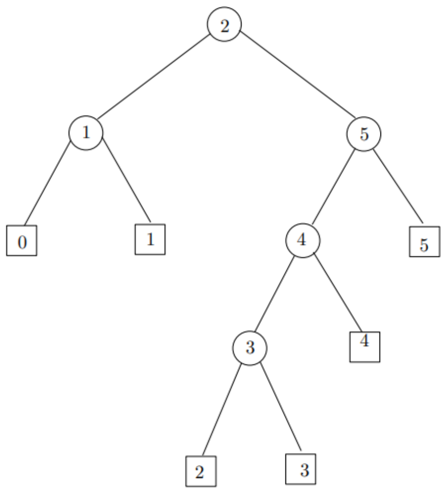


Figure 1: Re-drawing the example shown in class.

we label a node i if it stores A_i. Next, we augment the tree by adding children to each node that has less than two children, so that each node in the original tree has two children in the augmented tree. Nodes added in this way are called *dummy nodes*. When performing the search for a value not in $\{A_1, A_2, \ldots, A_n\}$ using the augmented binary search tree, we will reach a dummy node, and thus each dummy node represents a range between two consecutive given values. We label a dummy node with i if it represents the range (A_i, A_{i+1}). Thus q_i is the probability of reaching a dummy node labeled i.

Figure 1 re-draws the example shown in class, in which squares represent dummy nodes.

Now prove the following statement: If $p_n = q_n = 0$, then an optimal binary search tree storing $A_1, A_2, \ldots, A_n$ with probabilities $p_1, \ldots, p_n, q_0, \ldots, q_n$ can be obtained by making the following change to any optimal binary search tree for $A_1, A_2, \ldots, A_{n-1}$ with probabilities $p_1, \ldots, p_{n-1}, q_0, \ldots, q_{n-1}$: simply replace the dummy node labeled $n-1$ with the structure shown in Figure 2.

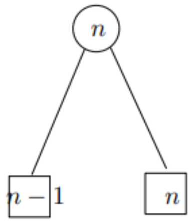


Figure 2: The structure used to replace the dummy node labeled $n-1$.

Solution:

The probability of accessing elements in the tree is P_1, P_2,… P_{n-1}

The probability of accessing dummy nodes is q_0, q_1, …q_{n-1}

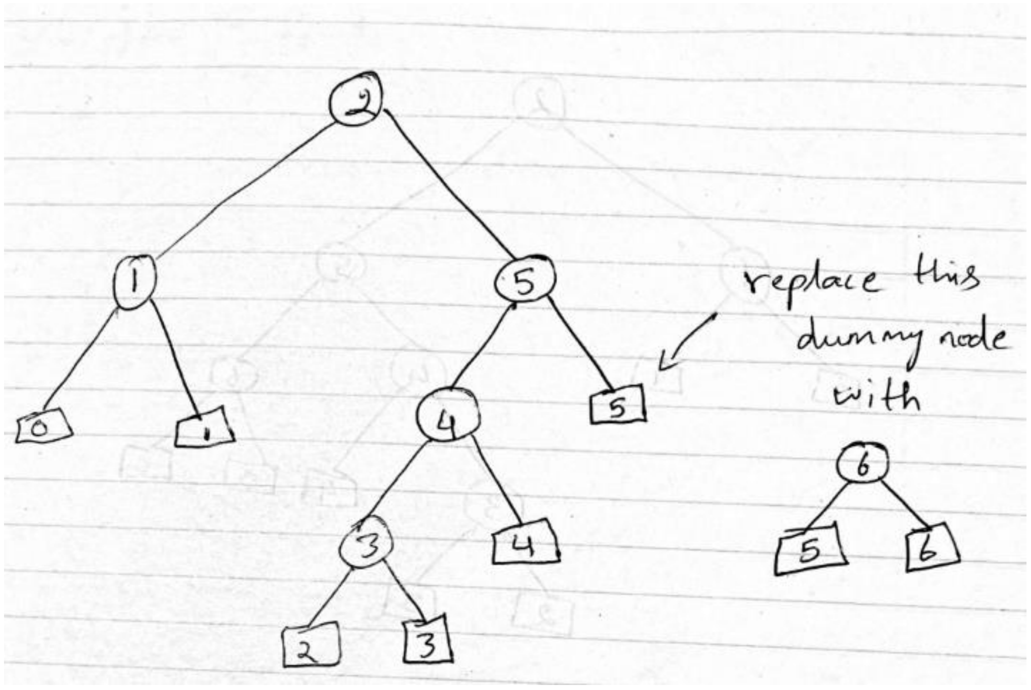


Figure 3: BST before replacing the dummy node

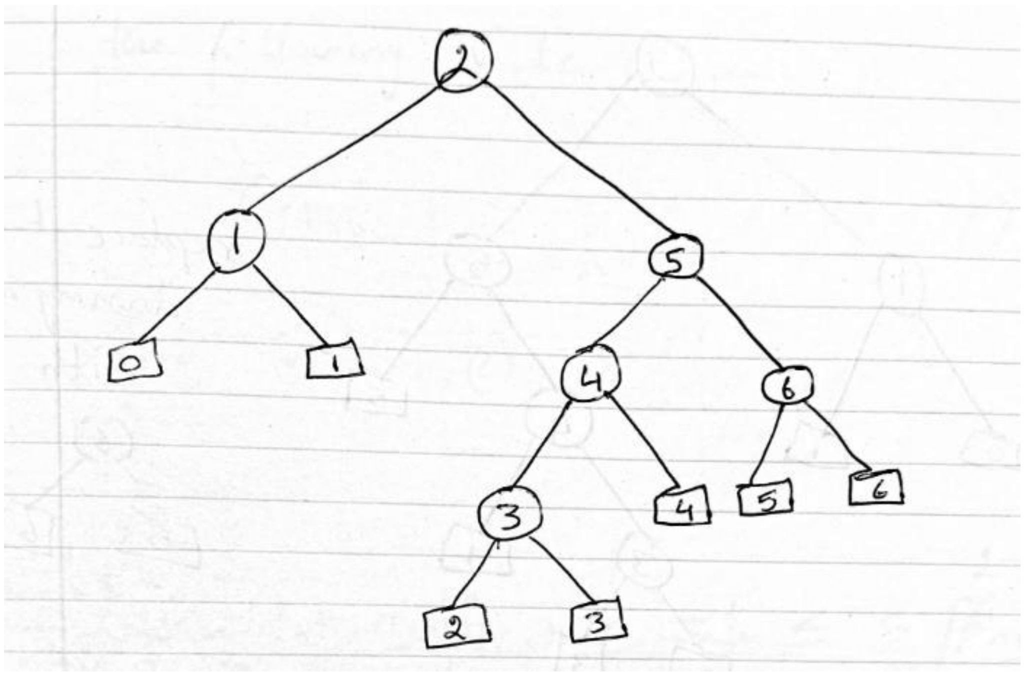
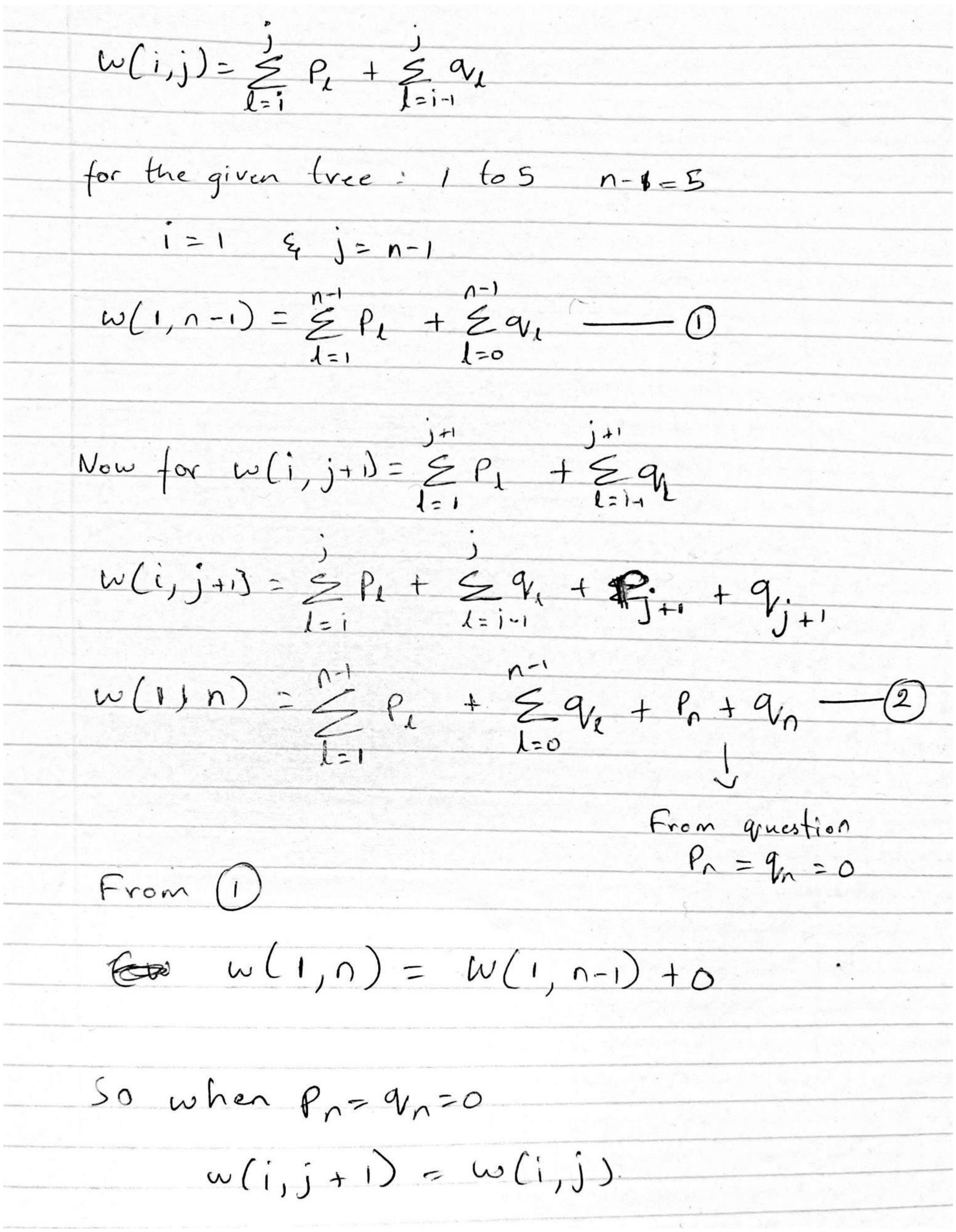


Figure 4: BST after replacing the dummy node

Taking the summation of probabilities:



$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

for the given tree : 1 to 5 $\quad n-\cancel{1}=5$

$i=1$ & $j=n-1$

$$w(1,n-1) = \sum_{l=1}^{n-1} p_l + \sum_{l=0}^{n-1} q_l \quad \text{———} \; ①$$

Now for $w(i,j+1) = \sum_{l=1}^{j+1} p_l + \sum_{l=i-1}^{j+1} q_l$

$$w(i,j+1) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l + p_{j+1} + q_{j+1}$$

$$w(1,n) = \sum_{l=1}^{n-1} p_l + \sum_{l=0}^{n-1} q_l + p_n + q_n \quad \text{———} \; ②$$

From question $p_n = q_n = 0$

From ①

$$w(1,n) = W(1,n-1) + 0$$

So when $p_n = q_n = 0$

$$w(i,j+1) = w(i,j)$$

References

[1] Lecture 10 Notes from winter 2022. Accessed: Mar. 4, 2024. [Online]. Availability: Lecture 10 notes from winter 2022 - CSCI6057 CSCI4117 - Advanced Data Structures - Sec: 01 - 2023/2024 Winter (brightspace.com)

[2] Lecture 11 Notes from winter 2022. Accessed: Mar. 4, 2024. [Online]. Availability: Lecture 11 Notes from Winter 2022 - CSCI6057 CSCI4117 - Advanced Data Structures - Sec: 01 - 2023/2024 Winter (brightspace.com)

[3] Lecture 12 Notes from winter 2022. Accessed: Mar. 4, 2024. [Online]. Availability: Lecture 12 Notes from 2022 - CSCI6057 CSCI4117 - Advanced Data Structures - Sec: 01 - 2023/2024 Winter (brightspace.com)

[4] Jon L. Bentley and Catherine C. McGeoch. 1985. Amortized analyses of self-organizing sequential search heuristics. Commun. ACM 28, 4 (April 1985), 404–411. https://doi.org/10.1145/3341.3349

[5] Knuth, D.E. Optimum binary search trees. Acta Informatica 1, 14–25 (1971). https://doi.org/10.1007/BF00264289