York University
EECS 4101/5101, Winter 2023
Assignment 2

Due Date: February 12th, at 23:59

Paths are made by walking ...

Franz Kafka

All problems are written problems; submit your solutions electronically **only via Crowdmark**. You are welcome to discuss the general idea of the problems with other students. However, you must write your answers individually and mention your peers (with whom you discussed the problems) in your solution. There is also a bonus question, which is harder than other questions. It is recommended to approach this question only if you have completed other questions. Please refer to the course webpage for guidelines on academic integrity.

Problem 1 List-Update Algorithms [10 marks]

Consider the Move-By-Bit algorithm for the list update problem. Here, each item has a bit associated with it. In the beginning, all bits are 0. After access to an item x, Move-By-Bit moves x to the front if the bit of x is 1; otherwise, it keeps x at its position. In addition, after each access, the bit of the accessed item is flipped.
Use a potential function argument to show the competitive ratio of Move-By-Bit is at most 3. You should indicate your potential function and the amortized cost of the algorithm for different scenarios, and provide an upper bound for the ratio between the amortized cost of the algorithm and Opt . (In case you wonder, the actual competitive ratio of Move-By-Bit is 2.5.)

Hint: You must change the definition of potential. In doing so, you need to consider the bits of the two items involved in an inversion.

Answer: Define inversions as we did for the analysis of MTF. Consider a pair of items which appear as $x \to y$ in the list of Move-By-Bit and $y \to x$ in the list of Opt (hence they form an inversion). Define the weight of the inversion between x and y as 2 if the bit of y is 0 and 1 if the bit of y is 1. Define the potential as the total weight of all inversions. Assume there is a request to an item a. As in the analysis of MTF, assume a is at index i of the algorithm and index j of Opt , and assume Opt makes k paid exchanges before serving a. Note that the cost of Opt is $j + k$. Any paid exchange of Opt creates at most one inversion and increases the potential by at most 2. So, the increase in potential because of Opt 's actions is $\Delta_{opt}\Phi \leq 2k$. For the algorithm, we need to consider two events:

- Assume Move-by-Bit moves a to the front (it means the bit of a was 1 before the access). In this case, at least $i - j$ inversions are removed (all having weight 1 since bit of a is 1), and at most j inversions are added (each having a weight of at most 2). So, the difference in potential because of the actions of the algorithm is $\Delta_{Alg}\Phi \leq -(i-j) + 2j = -i + 3j$. The amortized cost will be $actual\_cost + \Delta_{Alg}\Phi + \Delta_{opt}\Phi \leq (i) + (-i + 3j) + 2k = 3j + 2k$. Compared with the cost $j + k$ of OPT , we conclude that amortized cost is less than 3 times the cost of opt.

- Assume Move-by-Bit does not move a to the front (it means the bit of a was 0 before the access). In this case, the number of inversions does not change; but the weight of all inversions that include a will decrease from 2 to 1. As before, the number of such inversions is at least $i - j$. Hence for the increase of potential for actions of the algorithm, we have $\Delta_{Alg}\Phi \leq -(i-j) = -i + j$. The amortized cost will be $actual\_cost + \Delta_{Alg}\Phi + \Delta_{opt}\Phi \leq (i) + (-i + j) + 2k = j + 2k$. Compared with the cost $j + k$ of OPT , we conclude that amortized cost is at most twice the cost of Opt in this case.

In both cases, the amortized cost is less than 3 times the cost of OPT . Consequently, the competitive ratio of the algorithm is at most 3.

Problem 2 Compression [3+3+3+3 = 12 marks]

a) Apply the Burrows-Wheeler transform on the following string; show your work and the output.

HONEYPONY$

Assume $ precedes all characters when you sort rotations.

Answer:

HONEYPONY$		$HONEYPONY
ONEYPONY$H		EYPONY$HON
NEYPONY$HO		HONEYPONY$
EYPONY$HON	$\Longrightarrow^{sort}$	NEYPONY$HO
YPONY$HONE		NY$HONEYPO
PONY$HONEY		ONEYPONY$H
ONY$HONEYP		ONY$HONEYP
NY$HONEYPO		PONY$HONEY
Y$HONEYPON		Y$HONEYPON
$HONEYPONY		YPONY$HONE

The result will be the last column, i.e., YN$OOHPYNE

b) Assume an initial list $\$ \to A \to B \to E \to H \to N \to O \to P \to Y$, i.e., initially $ is at index 0, A is at index 1, etc. Assume we use Move-To-Front on the above list to encode the outcome of the BWT transform from part (a). Show what numbers will be encoded (you need to show how the list is updated).

Answer:

$\$ \to A \to B \to E \to H \to N \to O \to P \to Y \Longrightarrow Y : 8$ is encoded

$Y \to \$ \to A \to B \to E \to H \to N \to O \to P \Longrightarrow N : 6$ is encoded

$N \to Y \to \$ \to A \to B \to E \to H \to O \to P \Longrightarrow \$: 2$ is encoded

$\$ \to N \to Y \to A \to B \to E \to H \to O \to P \Longrightarrow O : 7$ is encoded

$O \to \$ \to N \to Y \to A \to B \to E \to H \to P \Longrightarrow O : 0$ is encoded

$O \to \$ \to N \to Y \to A \to B \to E \to H \to P \Longrightarrow H : 7$ is encoded

$H \to O \to \$ \to N \to Y \to A \to B \to E \to P \Longrightarrow P : 8$ is encoded

$P \to H \to O \to \$ \to N \to Y \to A \to B \to E \Longrightarrow Y : 5$ is encoded

$Y \to P \to H \to O \to \$ \to N \to A \to B \to E \Longrightarrow N : 5$ is encoded

$N \to Y \to P \to H \to O \to \$ \to A \to B \to E \Longrightarrow E : 8$ is encoded

So, the text is encoded as 8627078558.

c) Assume an initial list $\$ \to A \to B \to C \to D$, i.e., initially \$ is at index 0, A is at index 1, etc. Assume we use Timestamp on the above list to encode $DB\$AABCD$. Show what numbers will be encoded (you need to show how the list is updated and, in particular, its last state).

Answer: The list is updated as follows:

$\$ \to A \to B \to C \to D \Longrightarrow DB\$A$: the first accesses to these items does not change the list, 4, 2,

$\$ \to A \to B \to C \to D \Longrightarrow A$: 1 is encoded

$A \to \$ \to B \to C \to D \Longrightarrow B$: 2 is encoded

$A \to B \to \$ \to C \to D \Longrightarrow C$: 3 is encoded

$A \to B \to \$ \to C \to D \Longrightarrow D$: 4 is encoded

$A \to B \to D \to \$ \to C$

So, the text is encoded as 42011234.

d) Assume an initial list $\$ \to A \to B \to C \to D$. A compressing scheme that uses Move-To-Front has encoded the following numbers for a text T. Show what the actual text is. The numbers are 2 2 0 3 3.

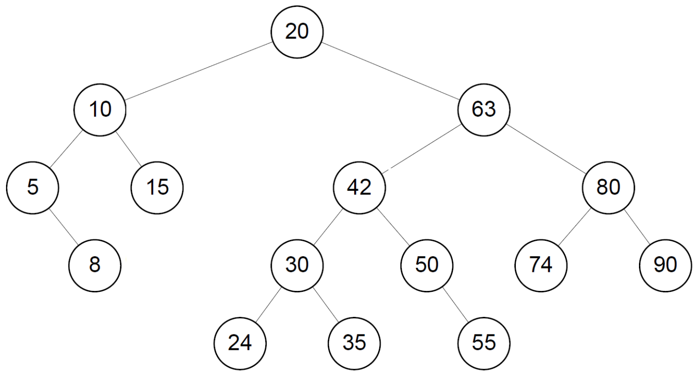
Answer:

$$\$ \to A \to B \to C \to D \Longrightarrow^{\text{2:B is decoded}}$$
$$B \to \$ \to A \to C \to D \Longrightarrow^{\text{2:A is decoded}}$$
$$A \to B \to \$ \to C \to D \Longrightarrow^{\text{0:A is decoded}}$$
$$A \to B \to \$ \to C \to D \Longrightarrow^{\text{3:C is decoded}}$$
$$C \to A \to B \to \$ \to D \Longrightarrow^{\text{3:\$ is decoded}}$$

So, the text is decoded as $BAAC\$$.

Problem 3 AVL Trees [3+3+3+3 = 12 marks]

This problem will concern operations on the AVL tree T shown in Figure 1.



a) Show that T is an AVL tree by writing in the balance at each node.

Answer: See Figure 1

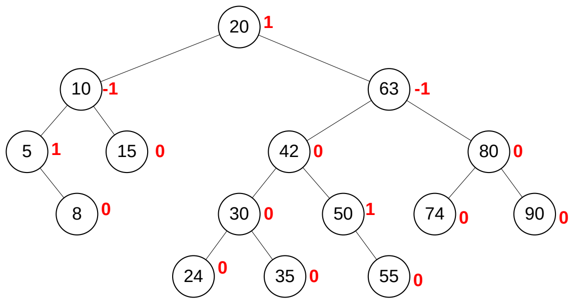


Figure 1: The answer to problem 4-a

b) Draw the tree after performing operation insert(33). Indicate any rotations that are required at each step.

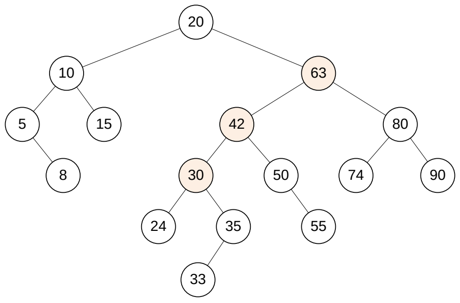
Answer: Left-left scenario; a right rotation is applied. See Figure 2.

c) Draw the *original tree* after performing operation delete(10). Swap with its *predecessor*. It suffices to draw the final tree.

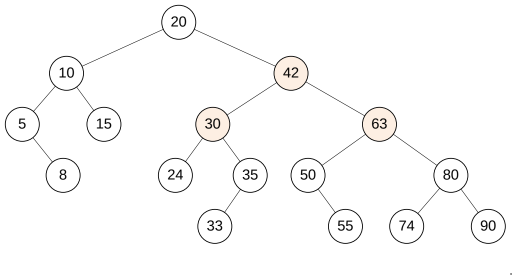
Answer: See Figure 3.

d) Draw the *original tree* after performing operation delete(10). This time, swap with its *successor*. It suffices to draw the final tree.

Answer: **Answer:** See Figure 4.

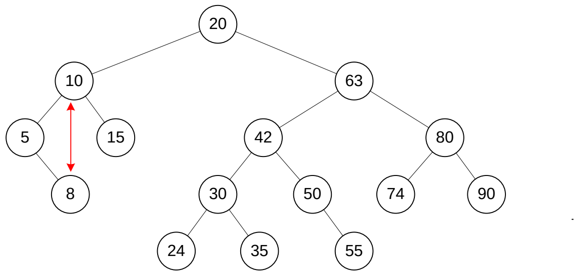


(a) Where the node 33 is inserted.

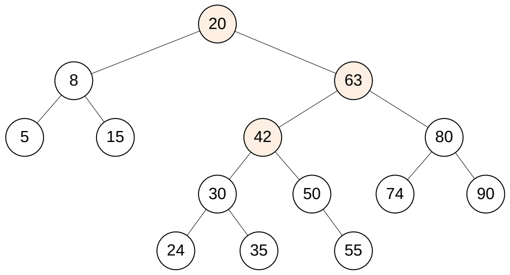


(b) The final tree.

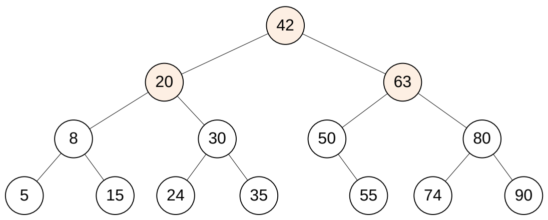
Figure 2: The answer to problem 4-b.



(a) Node 10 is replaced by 8 (its predecessor).

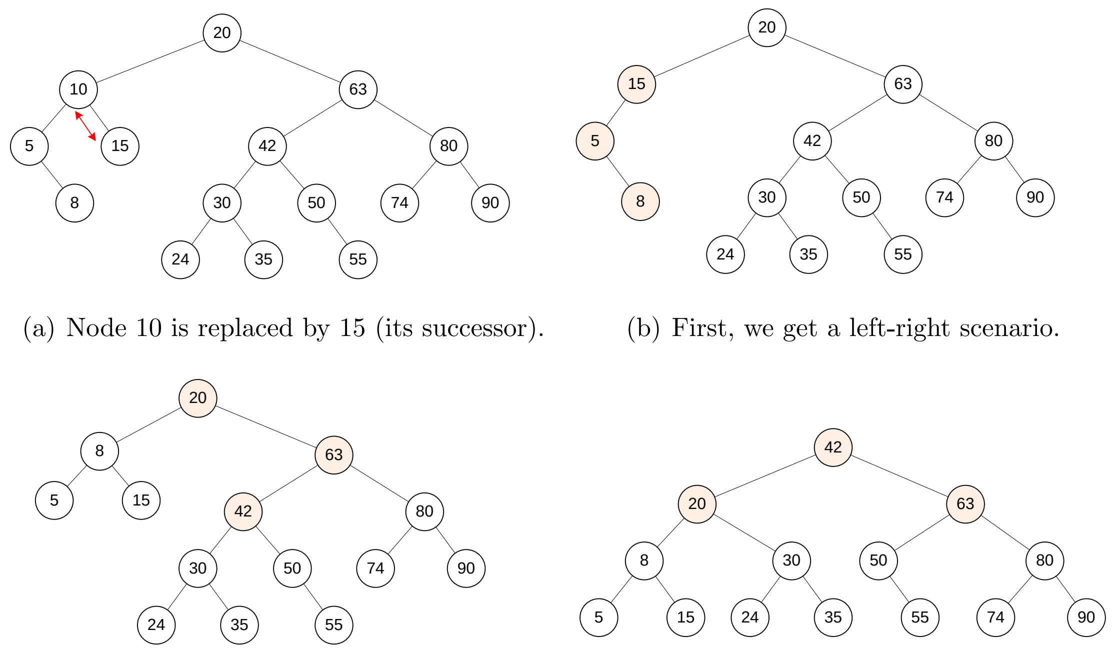


(b) Right-left scenario.



(c) Final tree.

Figure 3: The answer to problem 4-c.



(a) Node 10 is replaced by 15 (its successor).

(b) First, we get a left-right scenario.

(c) Then, we get a right-left scenario.

(d) Final tree.

Figure 4: The answer to problem 4-d.

Problem 4 (More) Binary Search Trees [8 marks]

We define *bar trees* as follows. A bar tree is a binary search tree where the heights of the left and right subtree for every node differ by at most 5. Prove that a bar tree with n nodes has height $O(\log n)$. **Answer:** Let $N(h)$ denote the number of nodes in a bar tree of height h. We can write $N(0) = 1$ and $N(-1) = 0$. For $h > 1$, we have $N(h) \geq N(h-1) + N(h-6)$; this is because one of the subtrees of a tree with height h has height $h - 1$ and the other one can have height at least $h - 6$ by the definition of bar trees. Following the inequality, we get $N(h) \geq 2N(h-6)$, and iterating $h/6$ times, we get $N(g) \geq 2^{h/6}N(0) = 2^{h/6}$. In other words, $\log(N(h)) = h/6$ or $h \leq 6\log(N(h))$. This means that a bar tree of n nodes has height $O(\log n)$.

Problem 5 Augmentation [5 + 8 marks]

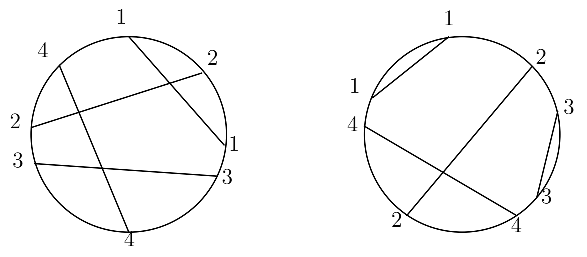
(a) Given a set of n KVPs with integer keys, describe a data structure that takes $O(n)$ space and supports the following two queries, both in $O(\log n)$ time:

- *range*(k_1, k_2) returns the number of KVPs in the dictionary whose key is in the range $(k_1, k_2]$. For example, for the dictionary with keys $\{1, 3, 4, 6, 7, 9, 13, 23\}$, the query *range*$(5, 12)$ must return 3.
- *rev-rank(k) returns the number of KVPs larger than k.*

Answer: For (a), you need to observe $range(k_1, k_2) = rank(k_2) - rank(k_1)$; therefore, it suffices to maintain an augmented AVL tree (which can be done in $O(n)$). For (b), we further augment the tree by adding another variable, n, which keeps the number of items in the tree. Then, the answer to rev-ran(k) is $n - rank(k)$. You must note that the value of n can be updated without hassle after an insertion/deletion by incrementing/decrementing n.

(bonus) Consider a disk of circular shape, and suppose we are given n line-segments, with endpoints on the disk. Describe an time algorithm that runs in $O(n \log n)$ and reports the number of pairs of line-segments that intersect. Assume that no two line-segments share an endpoint. For example, in the left figure below, the outcome must be 3, and on the right figure, it must be 1.

Hint: Start from an endpoint, and form an array of size $2n$ of endpoints (labelled by their interval) by visiting endpoints in clockwise order. In the left example below, the resulting array will be $[1, 2, 1, 3, 4, 3, 2, 4]$, and for the right example, the array will be $[1, 2, 3, 3, 4, 2, 4, 1]$. Now, scan the array and maintain (an initially empty) augmented AVL tree while scanning the array.



Answer: Find a detailed answer in `https://people.csail.mit.edu/rivest/pubs/BLRVW84.pdf`