Dalhousie University
CSCI 6057/4117 — Advanced Data Structures
Winter 2022
Assignment 3 Solutions

Questions:

1. [15 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:

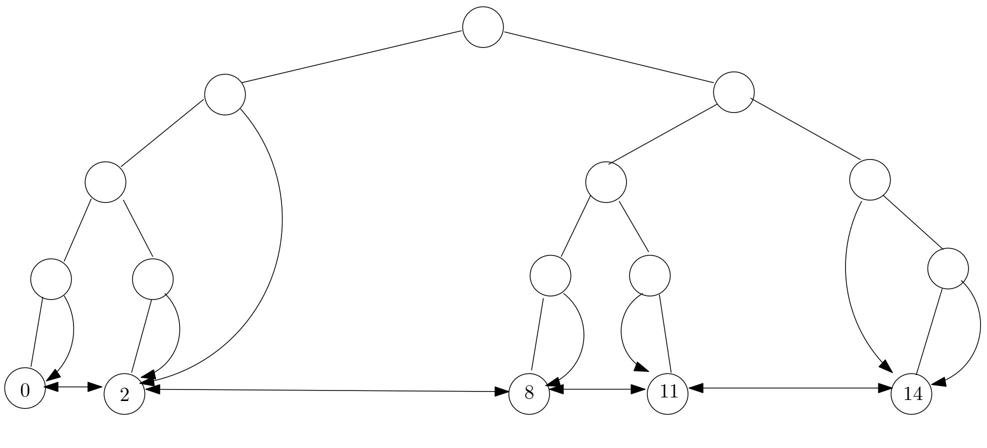

Figure 1: X-fast trie for question 1.

   Keys in the hash table:

$$\epsilon, 0, 1, 00, 10, 11, 000, 001, 100, 101, 111, 0000, 0010, 1000, 1011, 1110$$

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: Assume that initially, the list is $a_1a_2a_3 \ldots a_n$.

The following is the form of the request sequence: $a_na_{n-1}a_na_{n-1}a_na_{n-1}\ldots$.

For this request sequence, if we maintain the list using TR, then we always look for the item at position n of the list. Thus the total cost is mn.

If we maintain the list using S_{opt}, then, if m is odd, then a_n is the first object, and a_{n-1} is the second. The relative order of the remaining objects does not matter. The total cost in this case is $(m+1)/2 + (m-1)/2 \times 2 = 3m/2 - 1/2$.

If m is even, then a_n and a_{n-1} are the first two objects (their relative order does not matter), and the relative order of the remaining objects does not matter. The total cost in this case is $m/2 + m/2 \times 2 = 3m/2$.

Thus, the total cost of TR divided by that of S_{opt} is at least $(mn)/(3m/2) = 2n/3 = \omega(1)$.

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

Solution: For the request sequence given in (i), let C_{opt}, $C_{S_{opt}}$ and C_{TR} be the total costs when the list is maintained by the offline optimal algorithm, the S_{opt} algorithm and the TR algorithm. Then $C_{TR}/C_{opt} \geq C_{TR}/C_{S_{opt}} = \omega(1)$. Thus TR is not competitive.

3. [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 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 2 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 $A_1, A_2, \ldots, A_{n-1}$

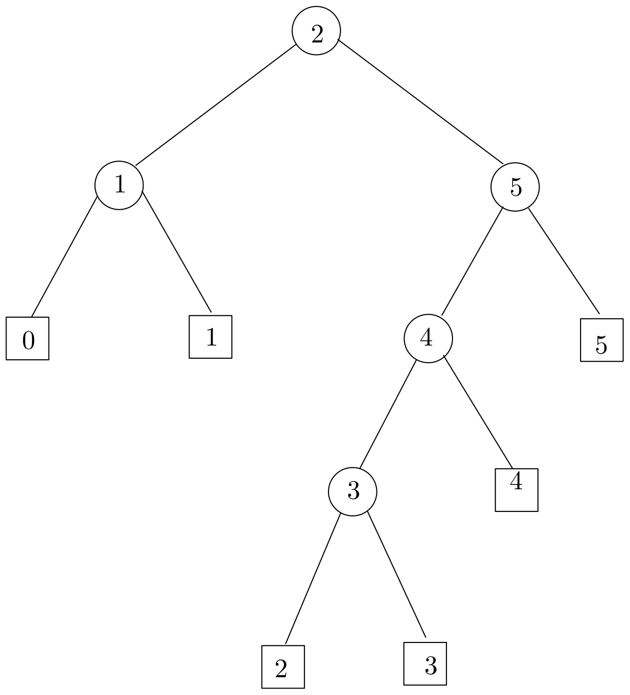

Figure 2: Re-drawing the example shown in class.

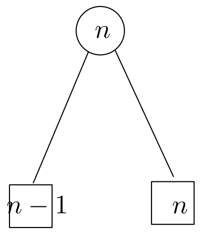

Figure 3: The structure used to replace the dummy node labeled $n-1$.

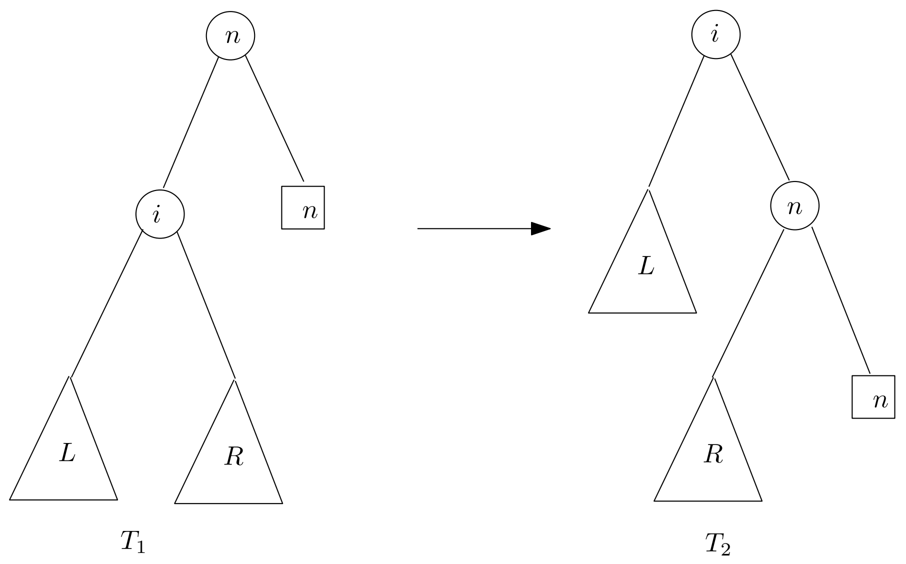

Figure 4: The single rotation performed in the solution to Question 4.

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 3.

Solution: There are two reasonable ways of defining the cost of accessing a dummy node: a) define the cost to be the number of comparisons needed to reach this node, and b) define the cost to be the number of nodes (including dummy nodes) examined to reach this node. As we do not perform any comparison after we reach a dummy node, the cost under the second assumption is 1 larger than the cost under the first assumption. Both assumptions are acceptable. My solution will work for either assumption.

In our solution, we say node A_i when we refer to the node storing item A_i, and dummy node q_i when we refer to the node corresponding to the gap between A_i and A_{i+1}. In any valid binary search tree for $A_1, A_2, \ldots, A_n$, the dummy node q_n must always be the right child of the node A_n, as A_n is the largest item.

We first prove that, if $p_n = q_n = 0$, then in any optimal binary search tree constructed over $A_1, A_2, \ldots, A_n$, the dummy node q_{n-1} must always be the left child of the node storing A_n. That is, the subtree rooted at A_n must be the same as the tree drawn in Figure 3.

We give a proof by contradiction. Assume to the contrary that there exists an optimal binary search tree, T_1, in which dummy node q_{n-1} is not the left child of A_n. Then this dummy node must be the right child of A_{n-1}, and the left child of A_n is a node A_i for some $i < n$. Let L and R denote the left and right child subtrees of A_i in T_1, respectively. We then construct another binary search tree, T_2, by single rotating A_i. Figure 6 illustrates T_1 and T_2.

After the single rotation, the cost of examining any node in L or node A_i is decreased by 1, the cost of examining any node in R remains unchanged, and the cost of accessing node A_n or node q_n is increased by 1. As the probability of accessing A_n or q_n is 0, this means that the expected search cost in T_2 is less than the expected cost of searching in T_1, which contradicts the fact the T_1 is an optimal binary search tree.

With this proved, we need only consider binary search trees that contain the subtree in Figure 3 when looking for an optimal binary search tree. Let T_3 be an arbitrary tree among these binary search trees. Then, if we replace the subtree in Figure 3 by dummy node q_{n-1} only, then we will immediately have a binary search tree, T_4, for $A_1, A_2, \ldots, A_{n-1}$. Observe that A_n and q_n are the only nodes that exist in T_3 but not in T_4, and their access probabilities are both 0. Furthermore, q_{n-1} is the only node whose access cost differs by 1 in T_3 and T_4. Therefore, the difference between the expected search cost in T_3 and that in T_4 is equal to q_{n-1}, which is a fixed value. Then if T_4 is an optimal binary search tree for $A_1, A_2, \ldots, A_{n-1}$, T_3 is also an optimal binary search tree for $A_1, A_2, \ldots, A_n$.

4. [10 marks] Let T be an arbitrary splay tree storing n elements $A_1, A_2, \ldots, A_n$, where $A_1 \leq A_2 \leq \ldots \leq A_n$. We perform n search operations in T, and the ith search operation looks for element A_i. That is, we search for items $A_1, A_2, \ldots, A_n$ one by one.

   (i) [5 marks] What will T look like after all these n operations are performed? For example, what will the shape of the tree be like? Which node stores A_1, which node stores A_2, etc.?

   Solutions: After all these operations are performed, the splay tree would be a chain of left children, and the nodes, from root to leaf, store $A_n, A_{n-1}, \cdots, A_1$.

   (ii) [5 marks] Prove the answer you gave for (i) formally. Your proof should work no matter what the shape of T was like before these operations.

   Solution: We prove by induction on i that, for any $i \in [1, n]$, after accessing A_i and splaying it to the root, the subtree rooted at the left child of A_i consists of elements $A_{i-1}, A_{i-2}, \ldots, A_1$, forming a chain of left children, and all other items are in the subtree rooted at the right child of A_i.

   In the base case, $i = 1$. Since there are no elements less than A_1, the left subtree is empty, and the base case holds.

   Assume that this is true for $i-1$. We now prove it for A_i. Consider the last splay step which splays A_i to the root. There are two cases:

   First, the last splay step is a zig step (you could also call this a zag step since it is the case symmetric to the zig step taught in class). In this case, A_{i-1} is the parent of A_i before this step. Since $A_{i+1}, \ldots, A_n$ are all greater than A_i, A_i does not have a left child before this step. Let S be the subtree rooted at its right child. Then, after splaying, $A_i, A_{i-1}, \ldots, A_1$ form a left chain, while S remains

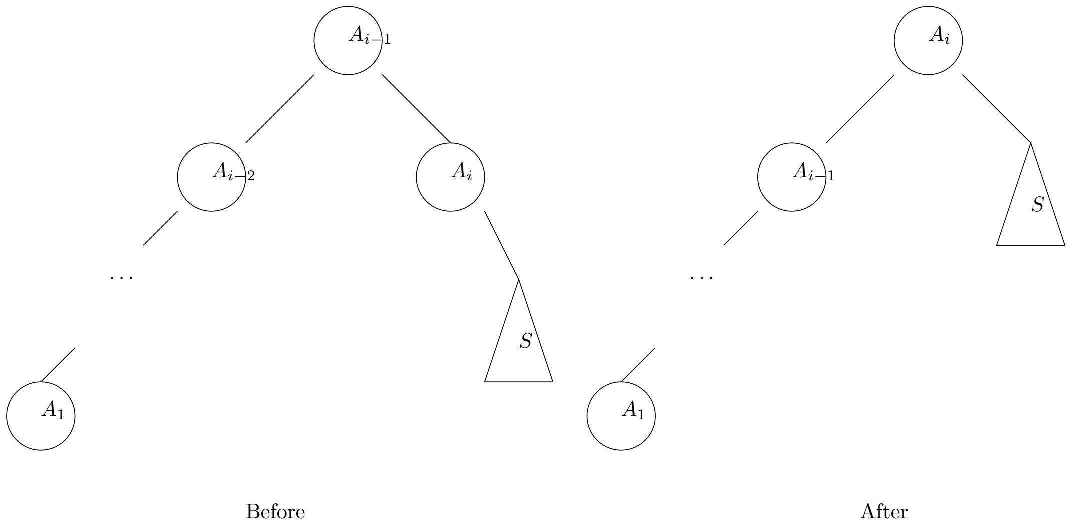

Figure 5: The zig case.

to be the subtree rooted at the right child of A_i, containing all elements greater than A_i. See Figure 5.

In the second case, the last splay step involves, A_i, the parent of A_i (call it A_k), and A_{i-1}. Since $A_k > A_i > A_{i-1}$, before this step, A_k is the right child of A_{i-1}, and A_i is the left child of A_k. A_i does not have a left child. Let S be the subtree rooted at the right child of A_i and S' the subtree rooted at the right child of A_k. Since this is a zig-zag step (you could also call it a zag-zig step), after splaying, A_i becomes the root, and all the items greater than A_i are in the subtree rooted at the right child of A_i. Since A_i did not have a left child before this step, $A_{i-1}, A_{i-2}, \dots, A_1$ form a chain of left children after this step. See Figure 6.

Hence the induction goes through in both cases.

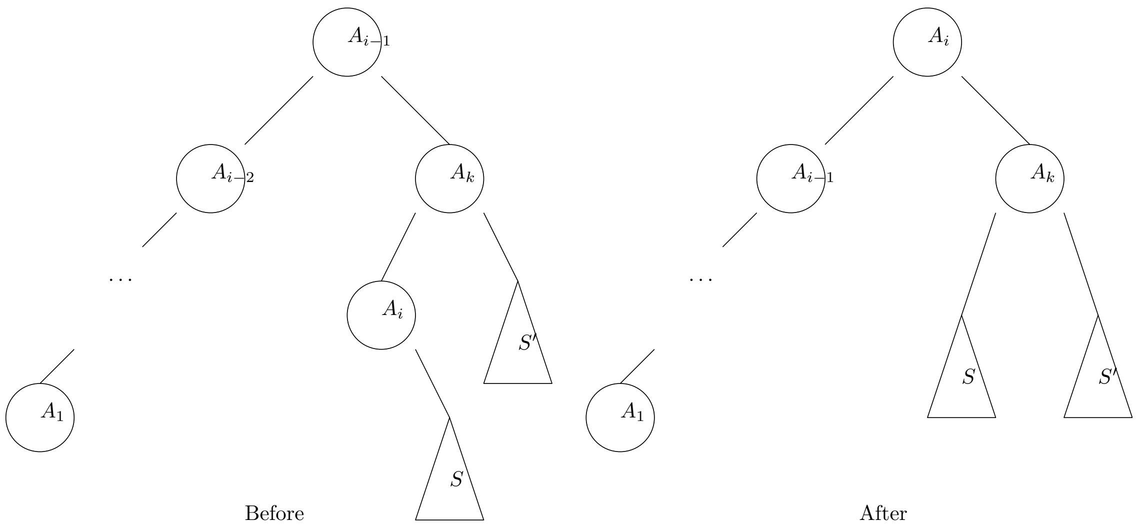

Figure 6: The zig-zag case.