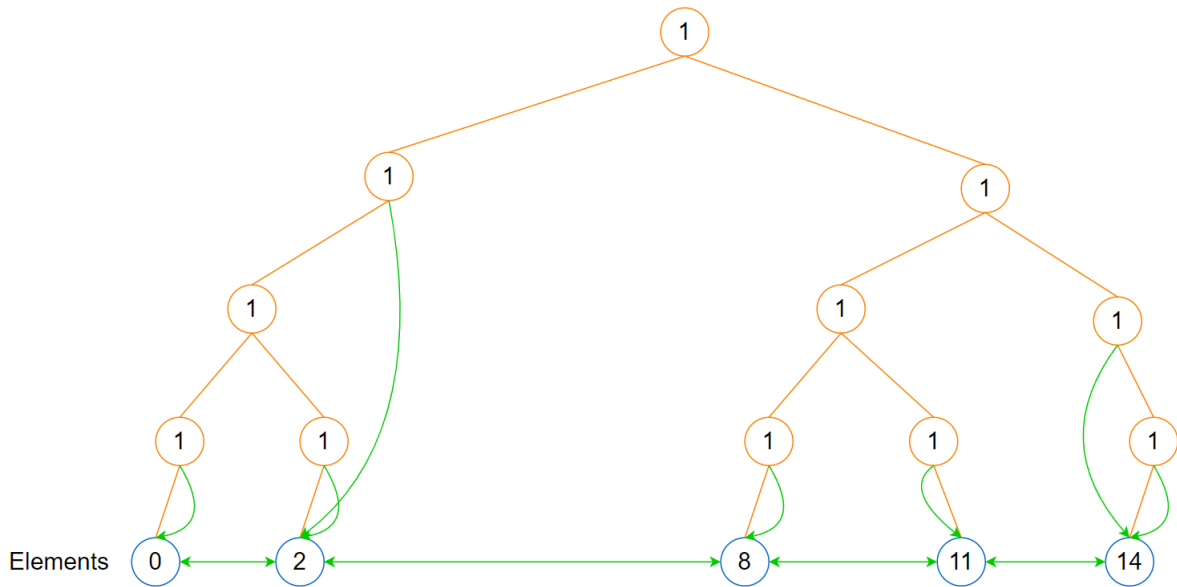


CSCI 6057 Advanced Data Structures - Assignment 3

Yansong Li
B00755354
yansong.li@dal.ca

Question 1.

Since $n = 5, u = 16, S = \{0, 2, 8, 11, 14\}$, the x-fast tris is shown as the following:



The keys that are stored in the dynamic perfect hash table:

ϵ	0	00	000	0000	001	0010	1	10	100	1000	101	1011	11	111	1110
------------	---	----	-----	------	-----	------	---	----	-----	------	-----	------	----	-----	------

Note: ϵ mean is the empty bit vector.

Question 2.

(i) Construct a sequence of requests S .

Since we are going to construct a sequence of requests S such that $\frac{\text{cost of TR}}{\text{cost of } S_{\text{opt}}} = \omega(1)$, the

following sequence of requests S is proposed. Assume we have a list $a_1, a_2, a_3, \dots, a_n$, and sequence of requests S is to access a_n then access a_{n-1} as a loop and repeat such a loop for n times ($S = (a_n, a_{n-1})^n$).

When we use TR algorithm, it cost n for each access because we are always accessing the last item in the list, and there are n times accesses for both a_n and a_{n-1} . Hence, the total cost of TR would be $n * n * 2 = 2n^2$.

When we use static optimality algorithm, since a_n and a_{n-1} are accessed n times (highest probabilities), they would be at the beginning of the list. Therefore, accessing cost of a_n and a_{n-1} would be 1 and 2. As a result, the total cost of S_{opt} is $3n$.

Therefore, $\frac{\text{cost of TR}}{\text{cost of } S_{opt}} = \frac{2n^2}{3n} = \frac{2n}{3}$. Since $\lim_{n \rightarrow \infty} \frac{2n}{3} = \infty$, which satisfy $\omega(1)$.

Hence, we have constructed a sequence of requests such that $\frac{\text{cost of TR}}{\text{cost of } S_{opt}} = \omega(1)$, which is $S = (a_n, a_{n-1})^n$ (access a_n , then access a_{n-1} ; repeat such a process for n times).

(ii) Argue that TR is not competitive.

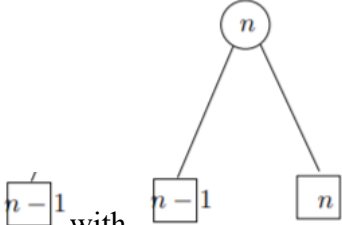
A method is “competitive” if and only if the competitive ratio of the algorithm is a constant. Therefore, for argue that TR is not competitive, we need to argue the competitive ratio of TR is not a constant, which is arguing $\frac{\text{worst case running time of TR}}{\text{optimal offline time}}$ not a constant. From the last

question we can see that accessing last two elements in the list alternatively take n time because we need to go through the entire list for each access; also, there is no paid exchanges during accesses, so the worst-case running time of TR is $2n$. Meanwhile, when we access the last two elements alternatively using offline algorithm, it would cost 3 since they have the highest probabilities, so they will be arranged at the beginning of the list. Hence, $\frac{\text{worst case running time of TR}}{\text{optimal offline time}} = \frac{2n}{3}$. Therefore, we have should the competitive ratio of TR is $\frac{2n}{3}$,

which is not a constant. Hence, TR is not competitive.

Question 3.

For an optimal binary search tree (OBST) for $A_1 \dots A_n$ that has $p_n = q_n = 0$, the probability that search for the real node A_n ($p_n = 0$) or the dummy node A_n are both 0 ($q_n = 0$). Therefore, no matter the costs of real node A_n and dummy node A_n , they don't affect the total cost of this OBST. Hence, the total cost of such an OBST $= \sum_{i=1}^{n-1} p_i * c_i + \sum_{i=1}^{n-1} q_i * c_i$. As a result, the OBST for $A_1 \dots A_n$ and the OBST $A_1 \dots A_{n-1}$ have the same $\sum_{i=1}^{n-1} p_i$ and $\sum_{i=1}^{n-1} q_i$. Hence, the next thing we need check is how could the node cost $\sum_{i=1}^{n-1} c_i$ change with the replacement.

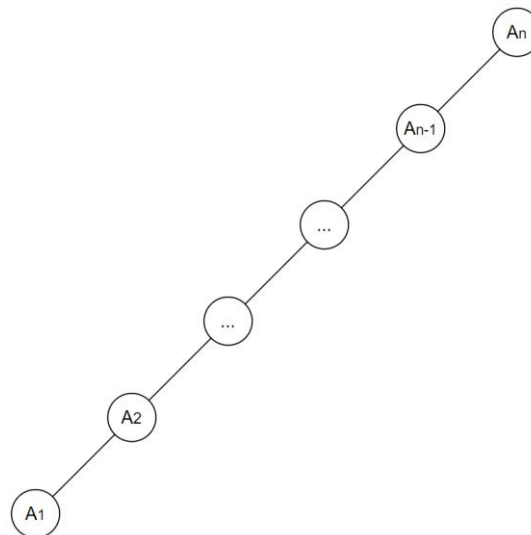
By replacing , we are actually adding a real node and dummy node

both with 0 probabilities ($p_n = q_n = 0$), which will not change the total cost of original OBST for $A_1 \dots A_{n-1}$. However, for the dummy node of $n-1$, its depth is down by one, so its cost increase by 1; therefore, the total cost of OBST for $A_1 \dots A_n$ is only increase by p_{n-1} . Since a OBST for $A_1 \dots A_n$ at least has the same cost of any OBST for $A_1 \dots A_{n-1}$ and at least has the cost of $p_n * c_n + q_n * c_n + q_{n-1} * c_{n-1}$ (obtained from the replaced tree structure). Meanwhile, the replacement doesn't add any new node with probability > 0 , and doesn't change any cost node cost except the cost of A_{n-1} , which is c_{n-1} . Hence, after the replacement, the tree is an OBST for $A_1 \dots A_n$. Therefore, we have proved that statement.

Question 4.

(i) The layout of splay tree T .

After we search for $A_1 \dots A_n$, the splay tree T will look like as the following figure:



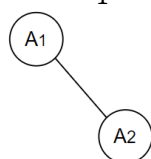
Specifically, every node in T will only has its left child. The root of the tree is A_n , and the only leaf of the tree is A_1 . The parent of a node is the element that just larger than it, and its only child (left child) is the element just smaller than it. For example, if we are looking at A_i , then its parent is A_{i+1} , and its only child is A_{i-1} .

(ii) Prove the answer in (i) by induction.

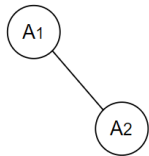
Base case:

Assume there are only two elements to search (two nodes in the tree):

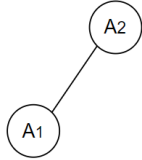
When A_1 is the root and A_2 is the leaf.



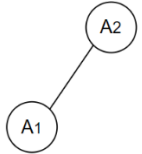
After searching for A_1 , the tree does not change, A_1 is still the root.



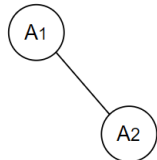
After searching for A_2 , the root changes to A_2 , and A_1 becomes the leaf. We have searched every element in ascending order, and we got the **same tree layout** in (i).



When A_2 is the root and A_1 is the leaf.



After searching for A_1 , the root changes to A_1 , and A_2 becomes the leaf.

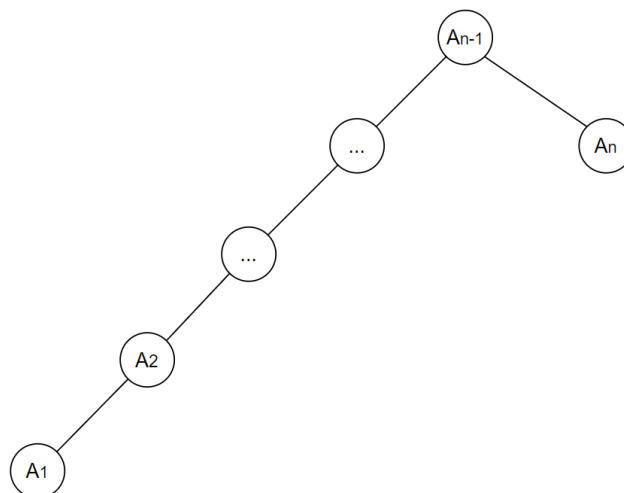


After searching for A_2 , the root changes back to A_2 , and A_1 becomes the leaf again. We have searched every element in ascending order, and we got the **same tree layout** in (i).

Induction Step:

Assume the answer in (i) holds for $A_1 \dots A_{n-1}$, prove it holds for $A_1 \dots A_n$.

After we search $A_1 \dots A_{n-1}$ one by one, the elements from A_1 to A_{n-1} are formed as answer in (i), so A_{n-1} will be the root and A_1 is the only leaf. Next, if we add A_n to the tree, it will be the right child of A_{n-1} since A_n is larger than A_{n-1} . If A_n is already in the tree, it will also be the right child of A_{n-1} since A_n is larger than A_{n-1} . Hence, any splay tree with elements $A_1 \dots A_n$ is formed as the following after searched A_1 to A_{n-1} in an ascending order:



Next, we search for A_n , which is the same case as the base case that A_1 is the root and A_2 is the leaf (treat A_{n-1} as A_1 and A_n as A_2). Hence, A_n becomes the new root, and A_{n-1} becomes its left child. The left sub tree in the figure is still the left sub tree of A_{n-1} . As a result, it forms the same layout as we showed in (i). Therefore, we have proved our answer in (i) by induction.