Solutions to Assignment 1

Instructor: Valerie King
Scribe: Zhiming Huang

October 11, 2018

1 Solution to Problem 1

i. The amortized cost for inserting an item is \$3. \$1 for insertion and the other \$2 go as credits. One credit for copying itself when expanding the table and the other for one former item to be copied.

As for the deletion cost, consider the situation where every table starts at least half full. If you pay a dollar to remove an item and pay a dollar for one item still in the table, then there is a dollar on each item when the table is $\frac{1}{4}$ full and this dollar will pay to shrink the table. An example is shown in Fig. 1. The table would be halved if two items were removed from the table. Deletion costs \$1 and copying the remaining items to the halved table also costs \$1 for each item. The number of items to be removed is identical to the number of items to be copied. Hence we can put \$2 on each item to be removed. One for deletion and the other one pays for the copy.

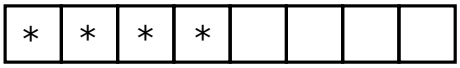

Figure 1: Deletion case for the first question

ii. The size of table will become $\frac{2}{3}$ if the number of items in the table is less than the $\frac{1}{3}$ table size. Similarly, The amortized cost for inserting an item is \$3.

As for the deletion cost, also consider the situation where every table starts at least half full. When deletion cause the table to be $\frac{1}{3}$ full then the new table will be $\frac{2}{3}$ full. So the $\frac{1}{2} - \frac{1}{3}$ removed items must pay for the $\frac{1}{3}$ items to be copied. So $\$2 + \$1$ are required to pay for the deletion. An example is shown in Fig. 2. Now if we delete an item, the table will shrink to $\frac{2}{3}$ size. The first and second items will be copied to the new table, which need \$2. Therefore we need put \$3 on the item we want to delete.

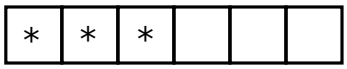

Figure 2: Deletion case for the second question

2 Solution to Problem 2

i. Because a linear sized array is allowed to used, we can firstly do an in-order traversal through the ordinary BST and put the vertexes in the array, which will take O(n) time. And we can rebuild the 1/2-balanced tree based on the array. First we can find the middle vertex of the array as the root of the whole tree. Then make the middle vertices of the left and right sub-array as the left and right children of the root. Do it recursively, which will take O(n) time to rebuild the whole tree. Therefore the algorithm takes O(n)+O(n) = O(n) time.

ii. The height h is bounded by O(logn), i.e., $h \leq k \cdot \log n$. We convert the problem into proving $n \geq 2^{\frac{h}{k}}$.

Proof 2.1 *1. Base Case, when $h = 0$, $n_0 = 0$, which satisfies $n \geq 2^{\frac{h}{k}}$.*

2. Assume when $h = i$, $n_i \geq 2^{\frac{h}{k}}$ is satisfied.

3. When $h = i+1$, there must be a child whose height satisfies $h = i$. So the size of the child $n_i \geq 2^{\frac{h}{k}}$. Because the tree is a 2/3-balanced three. The size of the other child is at least $\frac{1}{2} \cdot 2^{\frac{h}{k}}$. Therefore

$$n_{i+1} = 2^{\frac{h}{k}} + \frac{1}{2} \cdot 2^{\frac{h}{k}} + 1 = \frac{3}{2} \cdot 2^{\frac{h}{k}} + 1. \tag{1}$$

When we set $k \geq \frac{1}{\log \frac{3}{2}}$, n_{i+1} satisfies $n_{i+1} \geq 2^{\frac{i+1}{k}}$.

Therefore, the induction proves $n \geq 2^{\frac{h}{k}}$ also proves $h \leq k \cdot \log n$, i.e., $h =$ O(logn).

iii. From the problem description, we need to know the precondition of the BST is a 1/2-balanced BST. When we do a series of insertion and deletion, any part of the tree are not 2/3-balanced BST will be rebuilt to a new 1/2-balanced tree.

In such a process, we start with a 1/2-balanced BST and end with a 1/2-balanced BST. Assume the initial vertex number of the 1/2-balanced BST is $2n$. We need to add at least n vertices in one subtree to rebuild the whole tree. Because rebuilding a tree with n vertices costs n credits. Every time we insert a vertex, we need to add 3 credits on the root of the tree, so that we have enough credits to rebuild the tree after we add n vertices. The process is shown in Fig. 3. Every vertex in the path from the inserted vertex to the root of the whole tree is a root of a subtree. The insertion will also cause those subtrees to lost balance. Hence we need to put 3 credits on those vertices in the path for rebuilding use. Because the insertion and the traversal also cost 1 credit, we need to place 4 credits on the vertices in the path from the inserted vertex to the root.

For the convenience to analyze the deletion cost, we assume the initial BST has $4n$ vertex, with $2n$ left side and $2n$ right side. We need to remove at least n vertices from one side to make the tree not 2/3-balanced. The number of remaining vertex is $3n$, and $3n$ credits are needed to rebuild the tree. Thus we can also put 3 credits for rebuilding use on the root every time we remove a vertex. The process is shown in Fig. 4. For the subtrees rooted to the vertexes

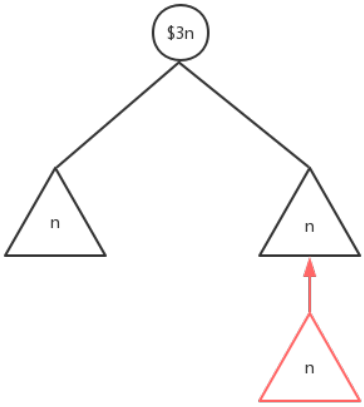

Figure 3: Case for insertion cost analysis

in the path from deleted vertex to the root, we also need to put 3 credits on those vertexes for them to rebuild their subtrees. Because the deletion and the traversal also cost 1 credit, we need to put 4 credits on the vertices in the path from the deleted vertex to the root.

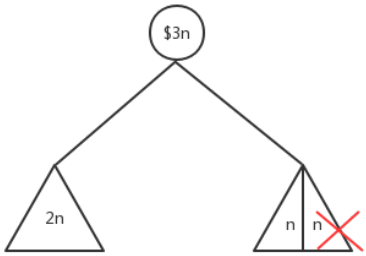

Figure 4: Case for deletion cost analysis

As for a mix of insertions and deletions, we can assume the initial vertex number of the 1/2-balanced BST is $6n$, with $3n$ in left subtree and $3n$ in right subtree. When we do a series of insertions and deletions, the tree will be unbalanced if we remove at least n vertices from one subtree and add at least n vertices to the other subtree, as shown in Fig. 5. The number of operations on vertices is $2n$ and the credits needed to rebuild the whole tree is $6n$. Thus every time whether we do an insertion or a deletion, we need to put 3 credits on the root of the tree. For the subtrees rooted to the vertexes in the path from deleted or added vertex to the root, we also need to put 3 credits on those vertexes for them to rebuild their subtrees. Because the deletion, insertion and the traversal also cost 1 credit, we need to put 4 credits on each vertex in the path from the deleted or added vertex to the root.

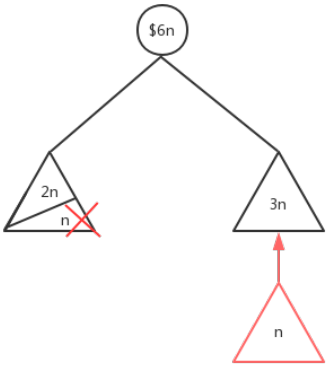

Figure 5: Case for mix operations cost analysis

To sum up, we need to put 4 credits on the vertices in the path from the vertex inserted or deleted.

3 Solutions to Problem 4

The worst case is shown in Fig. 6. V_1 is the root of the whole tree. Assume the number of vertices is n and the number of edges is m. There are $\frac{n}{2}$ vertices which link to each other and we label the group by G. The vertices in G is therefore sharing the same level. Another group of vertices link one by one. So the maximum level of the tree before cutting is $(\frac{n}{2} - 1)$. When cutting off the link between G and V_1, the level of G drops from 1 to $\frac{n}{2}$. For every drop, the cost for every edge is O(1). Hence the cost for every edge to drop $(\frac{n}{2} - 1)$ levels is O(n). There are totally $(m - (\frac{n}{2} + 1))$ edge drops per level. Therefore the overall cost is O(mn)

How to reduce the worst case running time? We can add a process C, which will generate a new BST after a deletion, which stops when the process A or B stops. So the worst case running time becomes O(m), where m is the number of edges.

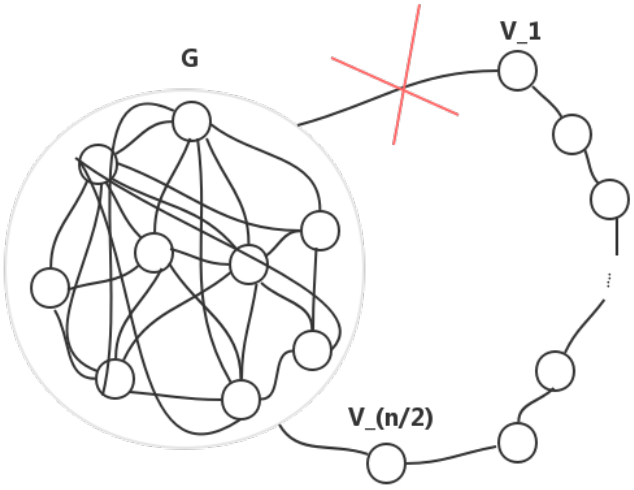

Figure 6: Worst case for problem 4