1) Claim: If a non-empty 2-3 tree T has n elements (where $n \geq 1$), then T has at most n-1 internal nodes.

Proof: This will be proved by strong induction on n. The cases when $n = 1$ will be considered in the basis.

Basis: If $n = 1$, then the tree will only consist of a leaf. So

T will have at most $0 = 1 - 1 = n - 1$ internal nodes, as required for this case.

Inductive Step: Let $k \geq 1$ be an arbitrary integer. It is necessary and sufficient to use the following inductive hypothesis to establish the following inductive claim.

Inductive Hypothesis: If a 2-3 tree T has m elements (where m is an integer from $1 \leq m \leq k$), then T has at most m-1 internal nodes.

Inductive Claim: If a 2-3 tree T has k+1 elements, then T has at most k internal nodes.

Hilroy

Now, suppose that T_x is a 2-3 tree with node x as root, and that T_x has $k+1$ elements.

Since $k \geq 1$, then $k+1 \geq 2$, so the root of T_x, node x, is an internal node. Then according to the 2-3 Tree Properties, we have two cases: the root x is either a 2-tuple node or a 3 tuple node.

Case 1: The root x is a 2-tuple node.

Then according to the 2-3 Tree Properties, x will have two children, y and z, and these children will either be leaves or internal nodes. Let T_y be the subtree with y as root, and let T_z be the subtree with z as root.

Note that the number of elements in T_x is equal to the number of elements in T_y plus the number of elements in T_z.

Then let h denote the number of elements in T_y, where $1 \leq h \leq k$. Then T_z will contain $(k+1) - h$ elements.

Since h is between 1 and k, it means that $(k+1)-h$ is between 1 and k, so we can apply the Inductive Hypothesis. Hilroy

According to the Inductive Hypothesis, T_y has at most $h-1$ internal nodes and T_z has at most $(k+1)-h-1 = k-h$ internal nodes. Since the number of internal nodes of T_x is at most the number of internal nodes of T_y + the number of internal nodes of T_z + 1, it follows that the number of internal nodes of T_x is at most $h-1+k-h+1 = k$, as required for this case.

Case 2: x is a 3-tuple node.

Then according to the 2-3 Tree Properties, x will have three children w, y and z, and these children will either be leaves or internal nodes. Let T_w be the subtree with w as root, let T_y be the subtree with y as root, and let T_z be the subtree with z as root. Note that the number of elements in T_x is equal to the number of elements in T_w plus the number of elements in T_y plus the number of elements in T_z.

Then let h denote the number of elements in T_w, and let j denote the number of elements in T_y, such that

$1 \leq h \leq k-1$, $1 \leq j \leq k-1$, and $2 \leq j+h \leq k$. Then T_z will contain $(k+1)-j-h$ elements. Since both j and h are between 1 and k, and $k+j$ is between 2 and k, it means that $(k+1)-j-h$ is between 1 and k. So we can apply the Inductive Hypothesis.

According to the Inductive Hypothesis, T_w has at most $h-1$ internal nodes, T_y has at most $j-1$ internal nodes, and T_z has at most $k-j-h$ internal nodes.

Since the number of internal nodes of T_x is at most the total number of internal nodes (at most) of all of it's children plus one, it follows that the total number of internal nodes of T_x is at most $h-1+j-1+k-j-h+1 = k-1$. Then it follows that the total number of internal nodes of T_x is at most k, as required for this case.

Since the inductive claim has been established in both cases, it means that the inductive step is now complete. Thus, we have proved this claim by strong induction on n.

Hilroy

2) Claim: If T is a non-empty 2-3 tree with depth $d \in \mathbb{N}$ and n is the size of the subset of E represented by T, then $2^d \leq n \leq 3^d$.

Proof: This will be proved by induction on d. Strong form of induction will be used, and the case when $d=0$ will be considered in the basis.

Basis ($d=0$): At depth 0, a 2-3 tree T will only contain a leaf.

So the size n of the elements of T will be 1.

Then $1 \leq n \leq 1$

$2^0 \leq n \leq 3^0$

$2^d \leq n \leq 3^d$ (since $d=0$), as required for this case.

Inductive Step: Let k be an integer such that $k \geq 0$.

It is necessary and sufficient to use the following Inductive Hypothesis to prove the following Inductive Step.

Inductive Hypothesis: Let T be a non-empty 2-3 tree with depth m (for all integers m such that $0 \leq m \leq k$) and let n be the size of the subset of E represented by T.

Then $2^m \leq n \leq 3^m$.

Hilbert

Inductive Claim: Let T be a non-empty 2-3 tree with depth $k+1$ and let n be the size of the subset of E represented by T. Then $2^{k+1} \leq n \leq 3^{k+1}$.

Now, suppose T_x is a non-empty 2-3 tree with root x and depth $k+1$. Let n be the size of the subset of E represented by T_x. Since $k \geq 1$, then $k+1 \geq 2$, so the root of T_x, node x, is an internal node. According to the 2-3 Tree Properties, the node x will have two cases: the root x is either a 2-tuple node or a 3-tuple node.

Case 1: The root x is a 2-tuple node.

Then according to the 2-3 Tree Properties, x will have two children, y and z, and these children will either be leaves or internal nodes. Let T_y be the subtree with y as root, and let T_z be the subtree with z as root. Note that:

n = the number of elements in T_y + the number of elements in T_z.

Then let the number of elements in $T_y = j$ such that $1 \leq j \leq n-1$, and let the number of elements in $T_z = n-j$.

Now, since both T_y and T_z have depth k, and k is between 0 and k, it means that the Inductive Hypothesis can be used.

Then according to the Inductive Hypothesis, $2^k \leq j \leq 3^k$ and $2^k \leq n-j \leq 3^k$.

Since $n = n-j+j$, it follows that $2^k+2^k \leq n-j+j \leq 3^k+3^k$,

so $2^{k+1} \leq n \leq 2 \cdot 3^k$, then

$2^{k+1} \leq n \leq 3 \cdot 3^k$ (since $2 \cdot 3^k \leq 3 \cdot 3^k$), then

$2^{k+1} \leq n \leq 3^{k+1}$, as required for this case.

Case 2: The root x is a 3-tuple node.

Then according to the 2-3 tree properties, x will have three children, u, y and z, and these children will either all be leaves or internal nodes.

Let T_u be the subtree with u as root, let T_y be the subtree with y as root, and let T_z be the subtree with z as root. Note that:

n = the number of elements in T_u + the number of elements in T_y + the number of elements in T_z.

Hilroy

Then let the number of elements in $T_1 = j$ and the number of elements in $T_3 = h$ such that $2 \leq j+h \leq n-1$. Then the number of elements in $T_2 = n-j-h$.

Now, since T_1, T_3 and T_2 all have depth k, and k is between 0 and k, it means that the Inductive Hypothesis can be used. Then according to the Inductive Hypothesis, $2^k \leq j \leq 3^k$, $2^k \leq h \leq 3^k$ and $2^k \leq n-j-h \leq 3^k$

Since $n = j+h+n-j-h$, it follows that

$$2^k + 2^k + 2^k \leq j+h+n-j-h \leq 3^k + 3^k + 3^k, \text{ then}$$

$$3 \cdot 2^k \leq n \leq 3^k, \text{ then}$$

$$2 \cdot 2^k \leq n \leq 3^k \quad (\text{since } 2 \cdot 2^k \geq 3 \cdot 2^k),$$

as required for this case.

Since the inductive claims for both cases have been established, it follows that the inductive step has been completed.

Thus, a non-empty 2-3 tree with depth $d \in \mathbb{N}$ and representing subset E of size n has $2^d \leq n \leq 3^d$.

3) Claim: Let $d \in \mathbb{N}$ and suppose that $|E| \geq 2^d$. Then there exists a non-empty 2-3 tree with depth d, representing a subset E with size exactly 2^d.

Proof: This will be proved by example. We will sketch a tree with the required characteristics.

Figure 1: A 2-3 Tree

In Figure 1, the tree consists of a single leaf that stores the element of 1. Since this satisfies the 2-3 Tree Properties, this is a 2-3 tree with depth $d=0$ and containing $n=1$ element.

So then,

Size of tree $= 1 = 2^0 = 2^d$ (since depth $d=0$),

as required to prove this claim.

Hilroy

4) Claim: The depth of a 2-3 subtree with root x is a bound function for the recursive algorithm get.

Proof: This will be proved using the definition of a bound function of a recursive algorithm.

Suppose that the get algorithm is executed with a non-null key $k \in E$ and a non-null node x which is in the 2-3 tree T. Let the depth of the subtree with root x be the integer $d \geq 0$. Assume that T satisfies the 2-3 Tree Properties.

- Since the depth of the subtree with root x is an integer d such that $d \geq 0$, it follows that the depth of the subtree with root x is an integer valued, total function.
- The only recursive calls in this algorithm are at steps 7, 8, 10, 12 and 13. In each of these steps, this algorithm is being recursively called with k and a child of node x as input. (Before each of these recursive calls are made, it is checked that x has the appropriate child).

Since the depth of the subtree with child of x as root Hilbroy

is one less than the depth of the subtree with x as root, it means that each time the algorithm is being called recursively, the depth of the subtree with the input node as root decreases by one. So every time the algorithm calls itself recursively, the value of the function decreases by at least one.

- The only scenario in which the bound function is less than or equal to zero is when the depth of the subtree with root x is zero (since the precondition requires that the depth be greater than or equal to 0).

So when the depth $d = 0$, it indicates that the node x is a leaf, so the test at line 1 is passed. Then depending on whether the key is equal to the element stored at x or not, either step 3 or step 4 is executed and the execution terminates. Now since steps 1, 2 and 3 did not contain any recursive executions, it shows that the algorithm does not call itself recursively when the depth d is less than or equal to 0.

Thus, using the definition of a bound function of a recursive algorithm, we have proved that the depth of the subtree with root x is a bound function of the get algorithm.

Hilroy

5) Claim: Suppose a 2-3 tree T satisfies the 2-3 Tree Properties. If the get algorithm is executed with with a non-null key $k \in E$ and a non-null node x of T as input, then the algorithm eventually terminates with the 2-3 tree not being changed. Moreover, if the key k is stored in the subtree of T with root x, then the leaf (in this subtree) storing x is returned as output. Otherwise, a NoSuchElementException is thrown.

We will prove this claim by standard induction on the depth d of the subtree with root x and $k \in E$ as input. The case when d=0 will be considered in the basis.

Basis: Suppose the get algorithm is executed with $k \in E$ and root x as input. Suppose the depth of the subtree with root x is d=0. Since depth d=0, it indicates that x is a leaf, so the test at line 1 is passed. Then there are two cases

Case 1): The key is equal to the element stored at x.

Then the test at line 2 is passed, and the leaf storing x is returned as output, as required for this case.

Case 2) The key is not equal to the element stored at x.

Then it means that the subtree with root x does not contain the key k, so NoSuchElementException is thrown, as required for this case.

Since the postcondition is satisfied for both cases, we have established the basis.

Inductive Step: Let k be an arbitrary integer such that $k \geq -1$. It is necessary and sufficient to use the following Inductive Hypothesis to prove the following Inductive Claim.

Inductive Hypothesis: Suppose a 2-3 tree T of depth k satisfies the 2-3 Tree Properties. If the get algorithm is executed with a non-null key $k \in E$ and a non-null node x (where the subtree with root x is a subtree of T and

input, then the algorithm eventually terminates with T not being changed. Moreover, if the key k is stored in the subtree of T with root x, then the leaf (in this subtree) storing x is returned as output. A NoSuchElementException is thrown otherwise.

Inductive Claim: Suppose a 2-3 tree T satisfies the 2-3 Tree Properties. If the get algorithm is executed with a non-null key $k \in E$ and a non-null node x (where the subtree with root x is a subtree of T and has depth of $k+1$) as input, then the algorithm eventually terminates with T not being changed. Moreover, if the key k is stored in the subtree of T with root x, then the leaf (in this subtree) storing x is returned as output. A NoSuchElementException is thrown otherwise.

Now, suppose a 2-3 tree T whose depth is $k+1$ satisfies the 2-3 Tree Properties. Suppose the get algorithm is executed with a non-null key $k \in E$ and the node x, which is the root of T, as input.

Since $k \geq 0$, then $k+1 \geq 1$, which means that the node x is an internal node. So at this point, according to the 2-3 Tree Properties, we have two cases:

Hilroy

case 1) x stores a 2-tuple, so x has two children.

In this case, the test at step 1 is failed, and the execution moves on to and passes the test at step 5. Then if the value of the key k is less than or equal to the largest element stored at x, the test is passed at step 6 and step 7 is executed, where the result of a recursive execution with k and the first child of x as input is returned as output. And if the value of the key k is greater than the largest element stored at x, then the test at step 6 is failed and step 8 is executed, where the result of a recursive execution with k and the second child of x as input is returned as output.

In both cases, the depth of the subtree with any child of x as root will be one less than k+1, so the depth will be k in both cases. This means that the Inductive Hypothesis can be applied to both cases.

Then according to the Inductive Hypothesis, the recursive execution in both cases will terminate with the subtree with the child of x as root not being changed. Moreover, if the key k is stored in this subtree, then the leaf (in this subtree) storing the child of x is returned as output. A No Such Element exception is thrown otherwise.

So whether step 7 or 8 is executed, both steps will eventually terminate and the post condition will be satisfied when that happens, as required for case 1.

Case 2) x stores a 3-tuple, so x has three children.

In this case, the tests at step 1 and 5 are failed, so the execution moves on to the test at step 9. At this point, we have a further three cases:

Case 2.1) If the value of the key is less than or equal to the largest element stored in the first subtree, the test at step 9 is passed and Step 10 is executed, where the result of a

Hilbert

recursive execution with k and the first child of x as input is returned as output.

Case 2.2) If the value of k is greater than the largest element stored in the first subtree, but less than or equal to the largest element in the second subtree, the test at step 9 is failed and the execution moves on to and passes the test at step 11. At step 11, the result of a recursive execution with k and the second child of x as input is returned as output.

Case 2.3) If the value of k is greater than the largest element stored in the second subtree, the tests at steps 9 and 10 are failed and the execution moves on and executes step 13. At step 13, the result of a recursive execution with k and the third child of x as input is returned as output.

In all cases 2.1, 2.2 and 2.3, the depth of the subtree with any child of x as the root will be one less than $k+1$, so the depth of the subtree with any of the three children as the root will

be K. This means that the Inductive Hypothesis can be applied to all the cases 2.1, 2.2 and 2.3.

So then according to the Inductive Hypothesis, the recursion execution in cases 2.1, 2.2 and 2.3 will terminate with the subtree with the child of x as root not being changed. Moreover, if the key k is stored in this subtree, then the leaf (in this subtree) storing the appropriate child of x (the same child of x used as input in the recursive execution) is returned as output. A NoSuchElement exception is thrown otherwise.

So whether case 2.1, 2.2 or 2.3 occurs, the recursive execution with it's postcondition satisfied will eventually terminate. Then it follows that the algorithm itself will terminate, and the postcondition will be satisfied when that happens, as required for case 2.

Now we have established the inductive claim for both case 1 and 2, which means that we have completed the inductive step.

Thus, it follows that the get algorithm correctly solves the "Searching in a Subtree of This 2-3 Tree" problem. Milroy

6) Let $T_{get}(k)$ be an upper bound for the number of steps used by the get algorithm where the depth of the subtree with input node x as root is k, where $k \in \mathbb{N}$ and $k \geq 0$. We are assuming Uniform Cost Criterion.

Suppose the get algorithm is executed with key $\in E$ and a non null node x as input, where the depth of x is k.

- If $k=0$, then we have two cases:

  Case 1) Steps 1, 2, 3 are executed before the algorithm

  Case 2) Steps 1, 2, 4 are executed before the algorithm ends.

  So if $k=0$, then at most three steps are executed before the algorithm ends.

- If $k \geq 1$, then we have two cases:

  Case 1) x has two children. Then:

  - Case 1.1) Steps 1, 5, 6, 7 are executed before the algorithm ends.
    However, there is a recursive call at step 7 with key and first child of x as input.

  - Case 1.2) Steps 1, 5, 6, 8 are executed before the algorithm ends.
    However, there is a recursive call at step 8 with key and second child of x as input.

Note that the subtree with a child of x as root will have depth of k-1. So at most $4 + T_{get}(k-1)$ steps will be executed before the execution terminates.

Case 2) x has three children. Then:

- Case 2.1) Steps 1, 5, 9, 10 are executed before the execution terminates. However, there is a recursive call at step 10 with key and the first child of x as input.
- Case 2.2) Steps 1, 5, 9, 11, 12 are executed before the execution terminates. However, there is a recursive call at step 12 with key and the second child of x as input.
- Case 2.3) Steps 1, 5, 9, 11, 13 are executed before the execution terminates. However, there is a recursive call at step 13 with key and the third child of x as input.

Note that the subtree with a child of x as root will have depth of k-1. So for case 2, at most $5 + T_{get}(k-1)$ steps will be executed before the execution ends.

Now we get the following recurrence for $T_{get}(k)$:

$$T_{get}(k) \leq \begin{cases} 3 & \text{if } k = 0 \\ 5 + T_{get}(k-1) & \text{if } k \geq 1 \end{cases}$$

7) The following function is an upper bound in closed form for $T_{get}(k)$:

$$T_{get}(k) \leq 5k + 3 \quad \text{(where } k \geq 0\text{)}.$$

We will now prove this.

Claim: $T_{get}(k) \leq 5k + 3$ for all integers $k \geq 0$.

Proof: This will be proved by induction on k. The standard form of induction will be used, and the case where $k = 0$ will be considered in the basis.

Basis: If $k = 0$, then

$$T_{get}(k) \leq 3 \quad \text{(by using the definition of } T_{get}(k)\text{)}$$

$$= 5(0) + 3$$

$$= 5(k) + 3 \quad \text{(since } k = 0\text{)},$$

as required for this case.

Inductive Step: Let $m \geq 0$ be an arbitrary integer. It is necessary and sufficient to use the following

Inductive Hypothesis: $T_{get}(m) \leq 5m + 3$

to complete the following

Hilroy

Inductive Step: $T_{get}(n+1) \leq 5(n+1) + 3$

$= 5n + 8$

Now, since $n \geq 0$, then $n+1 \geq 1$, so by the definition of the $T_{get}(n+1)$ recurrence, we get

$T_{get}(n+1) \leq 5 + T_{get}(n)$ (since $n+1 \geq 1$).

Note that since $n \geq 0$, we can apply the Inductive Hypothesis to $T_{get}(n)$, so we get

$T_{get}(n+1) \leq 5 + 5n + 3$ (by using the IH)

$= 5n + 8$,

as required to establish the Inductive Claim and complete the Inductive Step.

Thus, $T_{get}(k) \leq 5k + 3$ for all integers $k \geq 0$.

8a) 1) Create an empty node.

2) Set the value of the node to the value of the key (so it is now a leaf).

3) Set this leaf to be a root of the tree.

b) 1) If the value of the root is equal to the value of the key, throw a ElementFoundException

2) Otherwise, add the value of the key to the leaf, so it is now a 2-tuple internal node.

3) Make sure this 2-tuple node is in increasing order.

4) Create an empty node, set it's value to the smallest value of the root, and set this node to be the first child of the root.

5) Create an empty node, set it's value to the largest value of the root, and set this node to be the second child of the root.

Hilbert

Question 9)

1) We can "split" the 4-tuple node into two 2-tuple nodes, where the first 2-tuple node will have the first child and second child of the 4-tuple node as it's own respective first child and second child. Accordingly, the second 2-tuple node will have the third child and fourth child of the 4-tuple node as it's own respective first child and second child.

2) Then we can create a third new 2-tuple node, and set it's first value to be the second value of the first 2-tuple node, and it's second value to be the second value of the second 2-tuple node.

3) Then we can set the first child and second child of this third 2-tuple node to be the first 2-tuple node and the second 2-tuple node respectively.

4) Set this third node to be the parent of the first two nodes.

5) Then set this third node to be the root.

This process has used a constant number of steps, and the 2-3 Tree properties have now been restored without changing the subset.

10) Suppose that the precondition for the "Insertion into Subtree of This 2-3 Tree" problem is satisfied, and the insertIntoSubtree method is executed with the input key and a node x that is a leaf in T.

Then since x is a leaf, the test at step 1 will be passed, and then at step 2, the variable e of type E stores the element stored at x. Then two cases are considered:

Case 1) The element stored at node x is equal to the key.

Then since e stores the element stored at x, it would mean that e is equal to the input key, so the test at step 2 would be passed. Then at step 4, an ElementFoundException would be thrown, and the execution would end, with the postcondition being satisfied for this case.

Case 2) The element stored at x is not equal to the key.

Then since e stores the element stored at x, it would mean that e is not equal to the input key, so the test at

Hilbert

step 2 would not pass and the execution would move on to step 5, where a NoSuchElementException would be thrown and then this execution would end, with the postcondition satisfied.

So with the input key and a node x that is a leaf in the 2-3 tree T, the insertIntoSubtree method would terminate with the postcondition for the "Insertion into subtree of This 2-3 Tree" problem satisfied (and we are assuming the precondition was satisfied at the beginning of the method execution).

11) Suppose that the precondition for the "Insertion into Subtree of This 2-3 Tree" problem is satisfied and the insert Into Subtree method is executed with the key and with an internal node x whose children are are leaves, as inputs.

Then since x is not an internal node, the tests at step 1 will fail, and then steps 6 and 7 will be executed and then the test at step 8 will be reached. Then two cases are considered.

Case 1) x has two children.

Then step 8 is passed, and a further two subcases are considered:

Case 1.1) Value of key is less than or equal to the largest value stored at the first subtree of x.

In this case the test at step 9 will pass, and also means that if the key is going to be added to the set represented by the 2-3 tree with x as root, it will be added to the set represented by the subtree with the first child of x as root.

So that is why at step 10, a recursive execution Hilroy

with key and first child of x as input is executed.

Case 1.2) Value of key is more than the largest value stored at the first subtree of x.

In this case the test at line 9 will fail, and the execution will move on to step 11.

This also means that if the key is going to be added to the set represented by the 2-3 tree with x as root, it will be added to the set represented by the subtree with the second child of x as root.

So that is why at step 11, a recursive execution with key and second child of x is executed.

Case 2) x has three children.

In this case, the test at step 8 will fail and the execution will move on to the test at step 12.

A further three subcases will be considered:

Case 2.1) Value of key is less than or equal to the largest value stored at the first subtree of x.

In this case the test at step 12 will pass, and this subcase indicates that if the key is going to be added to the set represented by the 2-3 tree with x as root, it will be represented by the subtree with the first child of x as root.

So that is why at step 13, a recursive execution with key and first child of x as input is executed.

Case 2.2) Value of key is more than the largest value stored at the first subtree of x, but is less than or equal to the largest value stored at the second subtree of x.

In this case, the test at step 12 will fail, and the execution will move on to the test at step 14 and pass it. Then the execution will move on to step 15.

This subcase indicates that if the key is going to be added to the set represented by the 2-3 tree with x as root, it will be added to the set represented by the subtree with the second child of x as root. So that is why at step 15, Hilroy

a recursive execution with key and second child of x is executed.

(case 2.3) Value of key is more than or equal to the largest value stored at the third subtree of x.

In this case the tests at steps 12 and 14 will fail, and the execution will move on to step 15.

This subcase indicates that if the key is going to be added to the set represented by the 2-3 tree with x as root, it will be added to the set represented by the subtree with the third child of x as root.

So that is why at step 15, a recursive execution with key and second child of x is executed.

So in all the cases, there is a recursive execution with key and the appropriate child of x as input.

Now since the children of x are leaves, what we have shown in question 10) in the assignment will apply to the recursive

execution.

So if the child of x contains the same value as the key, then an ElementFoundException is thrown by the recursive execution. The original execution then recieves this exception, and it is left uncaught and the execution ends.

Since the input key already belonged to the subset represented by the subtree with root x, an elementNotFoundException was thrown and the execution terminated, as required to satisfy the post condition for this case.

However, if the child of x does not contain the same value as the key, then a NoSuchElementException is thrown by the recursive execution, and this recursive execution ends. The original execution recieves this exception, and this exception is caught at step 18. Then in step 19, the addLeaf method uses a constant number of steps to give node x a new child with is a leaf and has the same value as the key, with either the 2-3 Tree properties or the Modified 2-3 Tree Properties being satisfied. Hilbert

Then after the addLeaf method finishes executing, the execution of the insertIntoSubtree method ends as well.

So in this case the input key did not belong to the subset represented by the subtree with root x, so this value was added as a child of x (as a leaf). This means that the precondition was satisfied when the execution terminated.

So with input key and a node x (that has leaves as children) as input, for all cases the insertIntoSubtree method would terminate with the postcondition for the "Insertion into subtree of this 2-3 Tree" problem satisfied (and we are assuming the precondition was satisfied at the beginning of the method execution).

12) Outline of proof:

- To prove that the insertIntoSubtree algorithm correctly solves the "Insertion into Subtree of This 2-3 Tree" problem, we will have to use induction on n, where n is depth of the subtree with node x as the root, and this non-null node x and key $\in k$ is used as input for this algorithm. Specifically, strong induction on n will be used.
- The cases when $n = 0$ and $n = 1$ will be considered in the basis.
- Inductive step: Let k be an integer such that $k \geq 1$. It is necessary and sufficient to use the following inductive hypothesis to prove the following inductive claim.
- Inductive Hypothesis: Suppose that this 2-3 tree satisfies the 2-3 Tree Properties. If the insertIntoSubtree algorithm is executed with a non-null key $\in E$ (that is either not stored in any leaf of this tree, or is stored in a leaf of a subtree of this tree) and a non-null node x of this tree as input, where the depth of the subtree with root x is k, then this algorithm eventually terminates.

Milroy

Moreover, the following properties will be satisfied when that happens:

a) If the input key already belongs to the set represented by the tree, then an ElementFoundException is thrown and the tree is not changed.

b) If x is a leaf that stores an element of E that is not equal to the input key, then an ElementNotFoundException is thrown and the tree is not changed.

c) If x is an internal node and the input key does not (initially) belong to the subset stored at the leaves of the subtree of this tree, then

- The input key is added to the set stored at the leaves of this tree, and this tree is otherwise unchanged.
- Either this tree satisfies the 2-3 Tree Properties or this tree satisfies the Modified 2-3 Tree Properties and x is now an internal node with four children.

- Inductive Claim: This is very similar to the Inductive Hypothesis, but with the following difference:

  - The depth of the subtree with root x is m, where m is an arbitrary integer such that $0 \leq m \leq k$.

- Cases that must be handled in this proof:

  1) x is a leaf that stores the same value as key. This case is handled in the basis, when $n = 0$

  2) x is a leaf that stores a different value as key. This case is handled in the basis, when $n = 0$.

  3) x is an internal node with leaves as children. This case is handled in the basis, when $n = 1$.

  4) x is an internal node, with two children (that are all internal nodes). Some further subcases arise:

     4.1) key $\leq$ the first element stored at x.

     4.2) key $>$ the first element stored at x.

  Case 4 and it's subcases are handled in the inductive step. (when depth of subtree with node x as root is k+1) Hilroy

5) x is an internal node, with three children (that are all internal nodes). Further subcases arise from this case:

5.1) key $\leq$ the first element stored at x

5.2) key $>$ the first element stored at x, and
key $\leq$ the second element stored at x

5.3) key $>$ the third element stored at x.

Case 5 and it's subcases are handled in the inductive step (where the depth of the subtree with node x as root is k+1).

So after we consider cases 1-3 in the bases, and consider cases 4-5 in the inductive step, we have proved the correctness of the insertIntoSubtree algorithm.

13) In order to prove that the number of steps executed by this insertIntoSubtree method (when it starts with it's problem's precondition satisfied) is in O(depth(x)), where x is the node is given as input and the Uniform Cost criterion is used to define the number of steps, we must find an upper bound function for this method.

Let $T_{IST}(d)$ be an upper bound for the number of steps used by the insertIntoSubtree algorithm where the depth of the subtree with input node x as root is d, where $d \in \mathbb{N}$ and $d \geq 0$.

Suppose this algorithm is executed with key $\in E$ and a non-null node x os input, where the depth of x is d.

- If $d = 0$, then we have two cases:

  Case 1) Steps 1, 2, 3, 4 are executed.

  Case 2) Steps 1, 2, 3, 5, are executed

  So if $d = 0$, then at most four steps are executed before the execution terminates.

Hilroy

- If $d = 1$, then at most seven steps are executed (steps 1, 6, 7, 8, 12, 14) before either step 15 or step 16 is executed. Both these steps will have a recursive execution with a depth of zero, and based on the value of the child of x that this recursion execution is given as input, this recursive execution might throw a No Such Element Exception.

If this exception is thrown, then it will be caught by the original execution and so steps 18 and 19 will be executed. Step 19 includes an execution of the method addLeaf, which always uses a constant number of steps.

So if $d = 1$, then at most

$$T_{IST}(d) \leq 7 + T_{IST}(0) + 2 + T_{addLeaf}(d) = 7 + 4 + 2 + T_{addLeaf}(d)$$

steps are executed, where $T_{addLeaf}(d)$ is the constant number of steps used by the addLeaf method.

- If $d \geq 2$, then at most eight steps are executed before the method terminates, either steps 1, 6, 7, 8, 12, 14, 15, 17 or steps 1, 6, 7, 8, 12, 13, 16, 17.

However, steps 15 and 16 have recursive executions with a child of x as input.

Also, there is an execution of the raiseSurplus algorithm on input x at step 17. Note that this raiseSurplus algorithm always uses a constant number of steps.

So if $d \geq 2$, then at most

$T_{IST}(d) \leq 8 + T_{IST}(d-1) + T_{RS}(d)$ steps are executed, where $T_{RS}(d)$ is the constant number of steps used by the raiseSurplus method.

So now, we get the following recurrence for $T(d)$:

$$T_{IST}(d) \leq \begin{cases} 4 & \text{if } k = 0 \\ 13 + T_{oddLeaf}(d) & \text{if } k = 1 \\ 8 + T_{IST}(d-1) + T_{RS}(d) & \text{if } k \geq 2 \end{cases}$$

Hilroy

Now, in order to prove that the number of steps executed by this method is in $O(\text{depth}(x))$, where x is the node given as input, we need to use the definition of $O(\text{depth}(x))$.

Outline of proof using the definition of $O(\text{depth}(x))$:

1) Choose values for constants c and N_0 such that $c > 0$ and $N_0 \geq 0$

2) Let n be an arbitrary integer such that $n \geq N_0$

3) Prove that $T_{IST}(n) \leq c \cdot n$

4) Since $c > 0$ and $N_0 \geq 0$ are existientally quantified and n is universally quantified, it follows that there exists constants $c > 0$ and $N_0 \geq 0$ such that $T(n) \leq c \cdot n$ for all integers n where $n \geq N_0$.

After step 4, we have successfully proved that the number of steps executed by the insertIntoSubtree method is in $O(\text{depth}(x))$.

Additional Problems that need to be considered when solving the "Deletion from this 2-3 Tree" Problem:

Problem #1: Deletion from subtree of this 2-3 Tree

Precondition:

a) This 2-3 tree, T, satisfies the 2-3 Tree properties

b) key is a non-null input of type E

c) x is a non-null node in T

Postcondition:

a) If the input key does not belong to the set represented by the subtree with root x, then a No Such Element exception is thrown.

b) If x is a leaf whose value is equal to the key, then an Element Found Exception is thrown.

c) If x is an internal node and the key belongs to the subset represented by the subtree T with root x, then

- The leaf that contains the same value as the key is removed
- Either T satisfies the 2-3 Tree properties, or

T satisfies the "Modified Tree" properties and x is now an internal node with one child.

Hilroy

Problem #2: "Move Problem Node Up by One" problem

Precondition: T is a 2-3 tree that contains an internal node that

a) T is a 2-3 tree that has an internal node x.

b) T either has:

- exactly one internal node x, where one of the children of x has only one child.
- no internal nodes that have only one child.

c) The children of T are internal nodes.

Postcondition:

a) T is modified so that T still satisfies the modified Tree Properties and either:

- x has one child, or
- there is no internal node of T with either one or four children, so that T is a 2-3 tree.

Problem #3: "Delete leaf from Subtree Problem"

Precondition: T is a 2-3 tree, x is an internal node of T whose children are leaves, key is a non-null element that is stored in one of the leaves of T.

Postcondition: The leaf that contains the value of key is deleted, and the tree T still satisfies the "Modified Tree" properties. The subset represented by T does not contain the value of key, and either x has one children or T is a 2-3 tree.

Hilary

Problem #4: "Fix the root of the subtree Problem".

Precondition: The 2-3 tree T satisfies the "Modified 2-3 Tree" properties and it's root has exactly one child.

Postcondition: T now satisfies the 2-3 Tree properties, so it's root does not have only one child.

Problem #5: "Delete the first and only element from the tree" Problem.

Precondition: The 2-3 tree T only contains a single leaf, and this leaf is the root of T. Key is an input of type E.

Postcondition: The 2-3 tree is now empty. The leaf has been deleted from this tree.

Figure A: The Delete method (solves problem #1)

```
Public void delete (E key) throws NoSuchElementException {
1)      if (root == null) {
2)          throw new NoSuchElementException
        }
3)      else if (root.element != null) {
4)          deleteFirstElement (key)
        }
        else {
5)          deleteFromSubtree (key, root)
6)          fixRoot ()
        }
}
```

Figure B: The deleteFirstElement method:

```
Private void deleteFirstElement (E key) {
1)      root = null
}
```

Figure B: The deleteFromSubtree Method (solves Problem #1)

```
private void deleteFromSubtree (E key, node x) {
1)   if (x is a leaf) {
2)     Set e to be the element of E stored at x
3)     if (e is equal to the input key) {
4)       Throw an ElementFoundException
       } else {
5)       Throw a NoSuchElementException
       }
     }else {
       try {
6)       if (x has two children) {
7)         if (key ≤ x.firstMax) {
8)           deleteFromSubtree (key, x.firstChild)
           } else {
9)           deleteFromSubtree (key, x.secondChild)
         } else {
10)        if (key ≤ x.firstMax) {
```

Hilroy

```
11)                deleteFromSubtree(key, x.firstChild)
12)            } else if (key <= x.secondMax) {
13)                deleteFromSubtree(key, x.secondChild)
               } else {
14)                deleteFromSubtree(key, x.thirdChild)
               }
           }
15)        raiseSingle(x);
16)    } catch (ElementFoundException ex) {
17)        deleteLeaf(key, x)
       }
   }
}
```

Hilroy

Figure C: The deleteLeaf Method (solves problem #5)

```
Private void deleteLeaf (E key, Node x) {
1)   if (x has two children) {
2)       if (key ≤ x.firstMax) {
3)           x.firstMax = x.secondMax
4)           x.firstChild = x.secondChild
5)           x.secondMax = null
6)           x.secondChild = null
         } else {
7)           x.secondMax = null
8)           x.secondChild = null
         }
9)       x.numberChildren = 1
     } else {
10)      if (key ≤ x.firstMax) {
11)          x.firstMax = x.secondMax
12)          x.firstChild = x.secondChild
13)          x.secondMax = x.thirdMax
14)          x.secondChild = x.thirdChild
15)          x.thirdMax = null
16)          x.thirdChild = null
17)      } else if (key ≤ x.secondMax) {
18)          x.secondMax = x.thirdMax
19)          x.secondChild = x.thirdChild
```

Hilroy

```
20)             x.thirdMax = null
21)             x.thirdChild = null

        } else {

22)             x.thirdMax = null
23)             x.thirdChild = null
        }
24)     x.numberChildren = 2
    }
}
```

Figure 0: The fixRoot Method (Solves problem # 4)

```
private void fixRoot2 ( ) {

1)    if ( (root.element != null) and (root.numberChildren == 1)) {

2)        root = root.firstChild

3)        root.parent = null
    }
}
```

Milroy

All the related problems and their algorithms have been outlined in the pages above.

- The algorithm that solves the "Deletion from This 2-3 Tree", called the delete method, is quite similar in structure to the insert method. Both methods deal with the cases when the root is null, the root is a leaf, and the root is an internal node.
- However, unlike the insert method, the delete method throws a noSuchElementException for the first case, executes the deleteFirstElement for the second case, and executes the deleteFromSubtree method for the third case.

  Both methods also use an additional method that "fixes" the root and ensures the tree is a proper 2-3 tree.
- Both insertion and deletion involve some "problem node" that is identified and moved up to the root of the tree.

  For deletion, the "problem node" is the internal node that has only one children, and this node is dealt with by the moveSingle method.

Milroy

Sketch of Proof of Correctness of deleteFromSubtree algorithm:

Claim: Suppose a 2-3 tree T satisfies the 2-3 Tree properties. Suppose key is a non-null input of type E, and x is a non-null node in T.

If the input key does not belong to the set represented by the subtree with root x, then a NoSuchElement exception is thrown.

If x is a leaf whose value is equal to the key, then an ElementFound Exception is thrown.

If x is an internal node and the key belongs to the subset represented by the subtree with root x, then

- The leaf that contains the same value as the key is removed
- Either T satisfies the 2-3 Tree properties, or T satisfies the "Modified Tree" properties and x is now an internal node with one child.

Proof: We will prove this by strong induction on n, where n is the depth of the subtree with node x as the root, and this non-null node x and key $\in E$ is used as input for this algorithm.

The cases when $n=0$ and $n=1$ will be considered in the basis.

- Inductive Step: Let k be an integer such that $k \geq 1$.

It is necessary and sufficient to use the following inductive hypothesis to prove the following inductive claim.

Inductive Hypothesis: Let m be an integer such that $0 \leq m \leq k$.

The rest of the inductive hypothesis is almost the same as the claim of this proof, with the only difference being that the depth of x is m.

Inductive Claim: The inductive claim is almost the same as the claim of this proof, with the only difference being that the depth of x is $k+1$.

- The cases that are handled in this proof:

1) x is a leaf and stores the same value as key. This case is handled in the basis, when $n=0$

2) x is a leaf that stores a different value than the key. This case is handled in the basis, when $n=0$.

3) x is an internal node with leaves as children. This case is handled in the basis, when $n=1$

Hilbroy

4) x is an internal node, with two children (that are also internal nodes). Some further subcases arise;

4.1) key $\leq$ the first element stored at x.

4.2) key $>$ the first element stored at x

Case 4 and it's subcases are handled in the inductive step, where the depth of the subtree with node x as root is k+1. Then the depth of the subtrees with the children of x as root is k, and since both subcases involve recursive executions with key and a child of x as input, the Inductive Hypothesis can be applied to show that for both subcases the execution terminates and the postcondition is satisfied when that happens.

5) X is an internal node, with three children (that are all internal nodes). Some further subcases arise:

5.1) key $\leq$ first element stored at x.

5.2) key $>$ the first element stored at x, and

key $\leq$ the second element stored at x.

5.3) key > the third element stored at x.

The subcases of case 5 are handled in the inductive step, where the depth of the subtree with node x as root is k+1.

Then the depth of the subtree with the children of x as root is k, and since all three subcases involve recursive executions with key and a child of x as input, the Inductive Hypothesis can be applied to show that for the three subcases, the execution terminates and the post condition is satisfied. When that happens.

So after we establish cases 1-3 in the basis, and use the inductive hypothesis to prove that cases 4-5 terminate with the post condition satisfied, the inductive clame becomes established and thus the inductive step becomes complete.

Then we can conclude that the insert Into Subtree algorithm is correct.

Hilbury

Proof that the delete algorithm correctly solves the "Deletion from this 2-3 Tree" problem:

Claim: Suppose a 2-3 tree T satisfies the 2-3 Tree Properties. If the delete algorithm is executed with non-null input key $\in E$ as input, then the execution of the algorithm will eventually terminate and the following properties will be satisfied:

a) If the input key belongs to the subset represented by T then the key is removed from this set, which is otherwise unchanged. If the key does not belong to this subset of E, then a NoSuchElementException is thrown.

b) T still satisfies the 2-3 Tree properties.

Proof: This will be proved by strong induction on n. The cases when n=0 and n=1 will be considered in the basis.

Basis: If n=0, then the root of T is null. Then step 1 is passed, and at step 2, a NoSuchElementException is thrown (as required for this

Hilroy

case) and the algorithm terminates.

If n = 1, then the root is a leaf. So the test at step 1 is failed, and the execution moves on to and passes the test at step 3. At step 4, the deleteFirstElement algorithm is executed with Leg as input.

- As can be seen in figure B, only one step is executed in this method and the root of T is now null. The execution of deleteFirstElement algorithm then ends.

After step 4, the execution of the delete algorithm ends as well, with the tree T now null, as required for this

Inductive Step: Let k be an integer such that $k \geq 1$. It is necessary and sufficient to use the following Inductive Hypothesis to prove the following Inductive Step.

Inductive Hypothesis: Suppose a 2-3 tree T satisfies the 2-3 Tree Properties. Suppose the depth of T is m, where m is an integer such that $0 \leq m \leq k$.

If the delete algorithm is executed with non-null input

key ∈ E, then the execution will eventually terminate and the following properties will be satisfied:

a) If the input key belongs to the subset represented by T then the key is removed from this set, which is otherwise unchanged. If the key does not belong to this subset of E, then a NoSuchElementException is thrown.

b) T still satisfies the 2-3 Tree Properties.

Inductive Claim: This will be almost the same as the inductive claim, with the only difference being that the depth of T is k+1.

Now, suppose this algorithm is executed with non-null input key ∈ E, and T is a 2-3 tree that satisfies the 2-3 Tree Properties.

Suppose that the depth of T is k+1.

Since $k \geq 1$, then $k+1 \geq 2$, so the root of T is an internal node.

Then the tests at steps 1 and 3 will fail, so step 5 will be executed. Step 6 executes the deleteFromSubtree algorithm with key and the root of T as input.

Hilroy

Since we have already given an outline of the proof of correctness of this deleteFromSubtree algorithm, we know that this algorithm's execution will end.

We now have two cases:

Case 1: The input key does not belong to the set represented by T.

In this case, the deleteFromSubtree algorithm will end and throw a NoSuchElement exception.

The delete execution will recieve this execution and throw it, and then this execution will terminate, with the post condition satisfied for this case.

Case 2: The input key belongs to the set represented by T.

In this case, the deleteFromSubtree algorithm removes the leaf that contains the same value as the key, and then this algorithm terminates.

Then after this execution, T either satisfies the 2-3 Tree properties, or T satisfies the Modified 2-3 Tree properties and the root of x now has exactly one child.

So after the deleteFromSubtree method finishes executing, the execution of the delete algorithm moves on to step 6, and the fixRoot2 algorithm is executed.

- Figure D shows the algorithm of fixRoot2 algorithm.

Step 1 checks if the root is an internal node with one child. If that is the case, then step 1 is passed and the first (and only) child of the root becomes the root.

At step 3, the parent of this root is set to null, and then the execution terminates.

So after the fixRoot2 execution, if T had a root with one child, now T has a root with the correct number of children (either three or four), so now T satisfies the 2-3 Tree properties.

The delete execution then terminates.

So now the postcondition for this case has been satisfied.

Then since the algorithm terminates with the postcondition being satisfied for both cases, the inductive claim has been established and so the inductive step is complete. Hilroy

Thus, if the delete algorithm is executed with the precondition satisfied, then the execution will eventually terminate with the postcondition satisfied.