Review

11.8

Tu Jiadong

Topics covered in this review may not appear in the exam.
Topics not covered in this review may appear in the exam.

Topics

- Algorithm Complexity
- Hash Table
- Binary Search Tree
- AVL Tree

Algorithm Complexity

Landau Symbols: Definition

- A function $f(n) = \mathbf{O}(g(n))$ if there exists k and c such that

$$f(n) \leq c\ g(n)$$

- A function $f(n) = \mathbf{\Theta}(g(n))$ if there exist positive k, c_1, and c_2 such that

$$c_1\ g(n) \leq f(n) \leq c_2\ g(n)$$

- A function $f(n) = \mathbf{\Omega}(g(n))$ if there exists k and c such that

$$f(n) \geq c\ g(n)$$

whenever $n > k$

Landau Symbols: Calculating limit

$$f(n) = \mathbf{o}(g(n)) \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = \mathbf{O}(g(n)) \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) = \mathbf{\Theta}(g(n)) \qquad 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

$$f(n) = \mathbf{\Omega}(g(n)) \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$$

$$f(n) = \boldsymbol{\omega}(g(n)) \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Landau Symbols

- Polynomials dominate Logarithms:

  For every $x > 0, \log n = O(n^x)$.

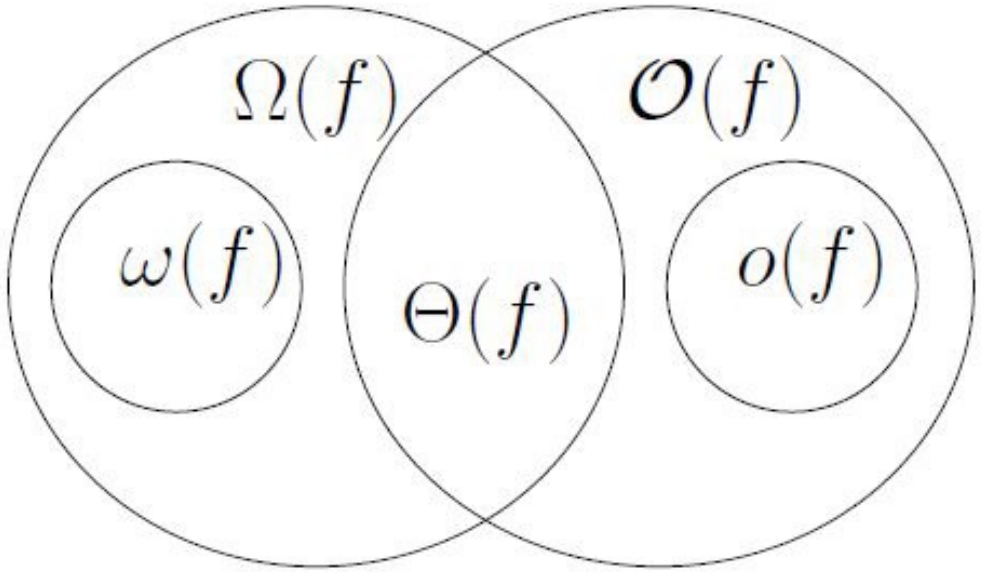
- Exponentials dominate Polynomials:

  For every $r > 1$ and every d > 0, $n^d = O(r^n)$.

$O(N^N)$

Complexity

O(N!)	Factorial
$O(2^N)$	Exponential
$O(N^3)$	Cubic
$O(N^2)$	Quadratic
O(N log N)	N x log N
O(N)	Linear
O(log N)	Logarithmic
O(1)	Constant



HW

2. $f(n) = \log(n!)$ *and* $g(n) = \log(n^n)$

$f(n) = \Theta(g(n))$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Note: You can also refer to [Stirling formula] for details.

Another way to prove: (by-definition)

$$g(n) = \log(n^n) = n\log n = \Theta(n\log n)$$

$$f(n) = \log(n!) = \sum_{i=1}^{n}\log(i) \geq \sum_{i=\frac{n}{2}}^{n}\log(i) \geq \sum_{i=\frac{n}{2}}^{n}\log(\frac{n}{2}) = (\frac{n}{2})\log(\frac{n}{2})$$

and

$$f(n) = \log(n!) = \sum_{i=1}^{n}\log(i) \leq n\log n$$

hence,

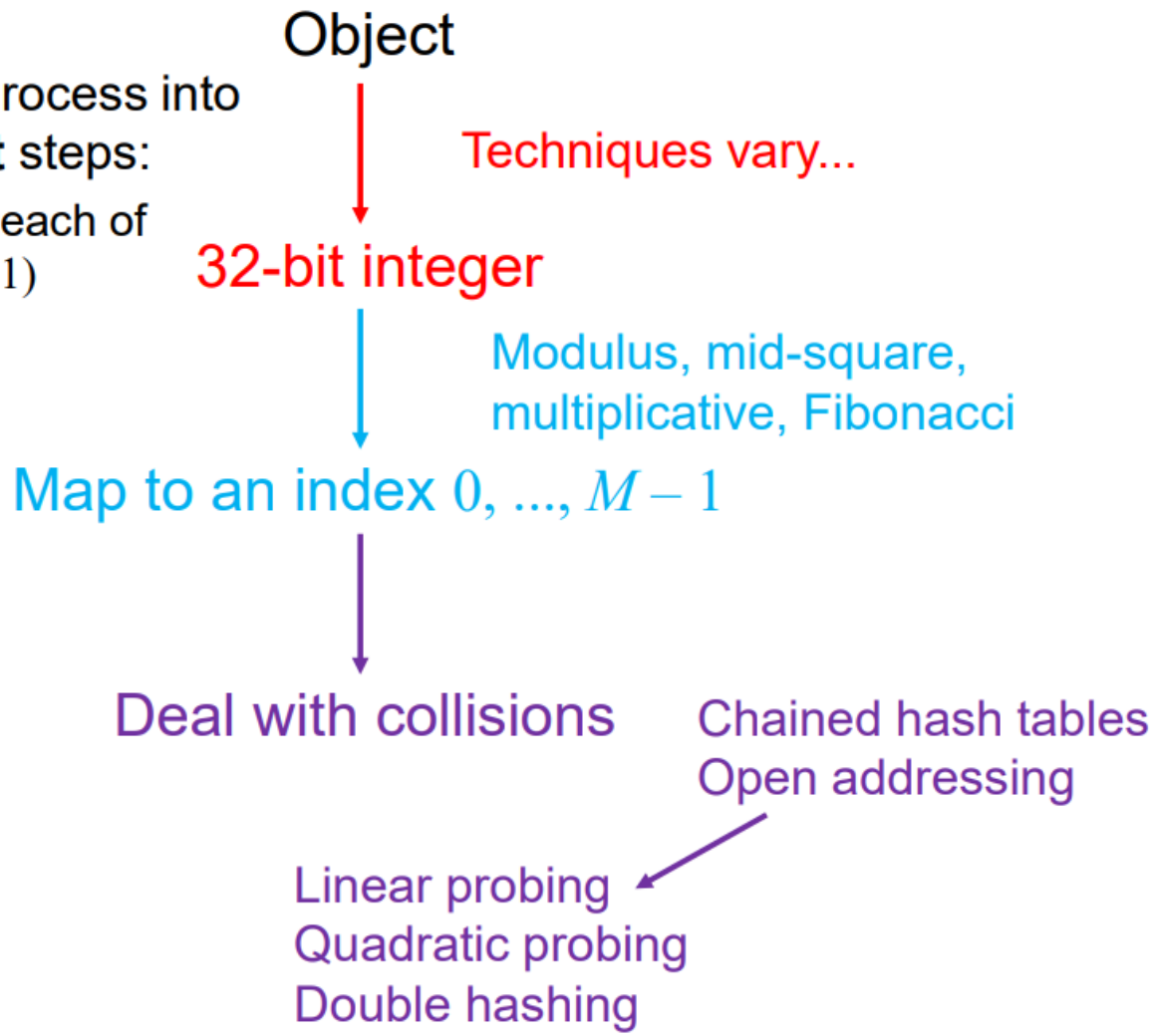
$$f(n) = \Theta(n\log n) = \Theta(g(n))$$

Hash Table

Hash Process

We will break the process into three **independent** steps:

- We will try to get each of these down to $\Theta(1)$



Linear Probing vs Quadratic Probing

Key: k, Hash function: h(k), Table size: m

Example:

- Example Probing Sequence (quadratic): $h(k) + 1^2, h(k) + 2^2, h(k) + 3^2, h(k) + 4^2, \ldots$
- Example Probing Sequence (linear): $h(k) + 1, h(k) + 2, h(k) + 3, h(k) + 4, \ldots$

Formally:

The i^{th} probe position for a value k be given by the function

$$h(k, i) = h(k) + c_1 i + c_2 i^2 \pmod{m}$$

Note: If $c_2 \neq 0$, it is quadratic probe. If ($c_2 = 0$, $c_1 = 1$), it degrades to a linear probe

Performance for Open Addressing

n: # of keys in the table
m: size of the table

- **Theorem:** Under the uniform hashing assumption, a single insertion/search/deletion has expected time cost (average number of probes) of $O(1/(1-\alpha))$, where $\alpha = n/m$ is the load factor.

Proof: for details refer to the text book

X : # of probes required for an unsuccessful search

p : is the probability of finding an empty slot at each probe(trial)

trial

$$\Rightarrow P(x) = p(1-p)^{x-1}$$

$$\Rightarrow E(X) = \sum xP(x) = \frac{1}{p} = 1/(1-\alpha)$$

HW2

1: (2'+1'+2'+2'+2') Hash Collisions

Consider a hash table consisting of $M = 11$ slots, and suppose we want to insert integer keys $A = [43, 23, 1, 0, 15, 31, 4, 7, 11, 3]$ orderly into the table. We give the hash function $h1()$ in the form of pseudocode as below:

```
int h1(int key) {
    int x = (key + 7) * (key + 7);
    x = x / 16;
    x = x + key;
    x = x % 11;
    return x;
}
```

Question 1. *Suppose that collisions are resolved via chaining. If there exists collision when inserting a key, the inserted key(collision) is stored at the end of a chain. In the table below is the hash table implemented by array. Insert all the keys into the table below. If there exists a chain, please use → to represent a link between two keys.*

Index	0	1	2	3	4	5	6	7	8	9	10
Keys	31→4	43→15	23	0		1			7	11→3	

Question 2. *What is the load factor λ?*

$\lambda = \frac{10}{11}$

Question 3. *Suppose that collisions are resolved via linear probing. The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (do not need to write the home slot again) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:*

Key Value	43	23	1	0	15	31	4	7	11	3
Home Slot	1	2	5	3	1	0	0	8	9	9
Probe Sequence					2,3,4		1,2,3,4,5,6			10

Question 4. *Suppose that collisions are resolved via a kind of quadratic probing, with the probe function as below:*

$$(k^2 + k)/2$$

This means that if the key is hashed to the i-th slot, we will check the $(i + (k^2 + k)/2)$-th slot for the k-th probing. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (do not need to write the home slot again) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	43	23	1	0	15	31	4	7	11	3
Home Slot	1	2	5	3	1	0	0	8	9	9
Probe Sequence					2,4		1,3,6			10

Final Hash Table:

Index	0	1	2	3	4	5	6	7	8	9	10
Keys	31	43	23	0	15	1	4		7	11	3

Question 5. *Suppose that we are using a hash table with m slots, where $m > 1$. Recall that in open addressing we store an element in the first empty slot among $A_0(k), A_1(k), \cdots, A_{m-1}(k)$. In this problem,*

$$A_i(k) = (h(k) + i^2 + i) \bmod m$$

where $h(k)$ is some hash function.

Prove that the probe sequence will always include at most $(m+1)/2$ distinct slots.

We can easily get that $A_i(k) = A_{m-i-1}(k)$. When m is even, the distinct slots is $m/2$ and when m is odd, the distinct slots is $(m+1)/2$.

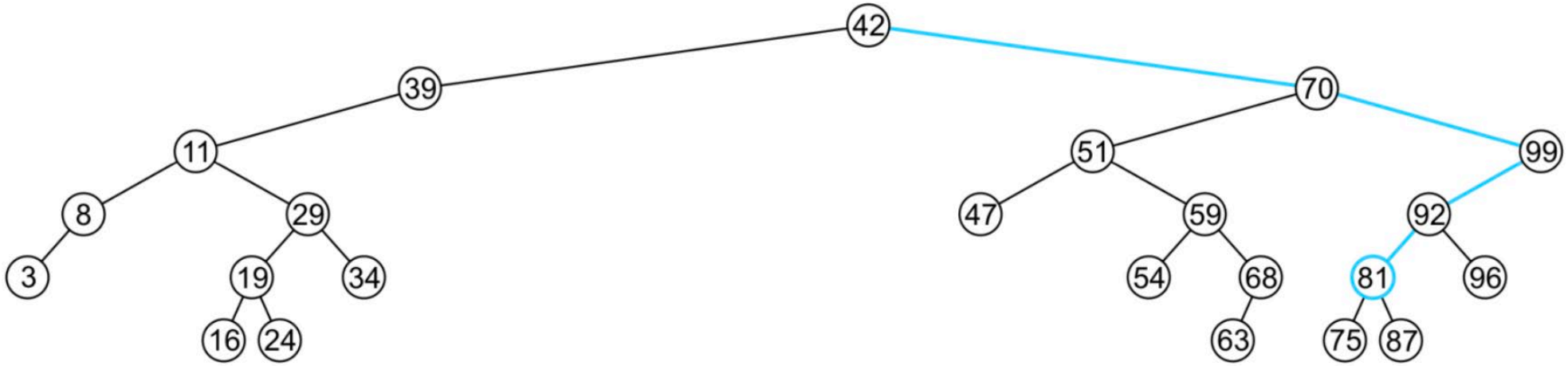
$$\begin{aligned} A_{m-i-1}(k) &= (h(k) + (m-i-1)^2 + m - i - 1) \bmod m \\ &= ((h(k) + i^2 + i) \bmod m + (m^2 - 2m(i+1) + m) \bmod m) \bmod m \\ &= (h(k) + i^2 + i) \bmod m \\ &= A_i(k) \end{aligned}$$

∴ $A_0(k), A_1(k), \cdots, A_{m-2}(k), A_{m-1}(k)$

Binary Search Tree

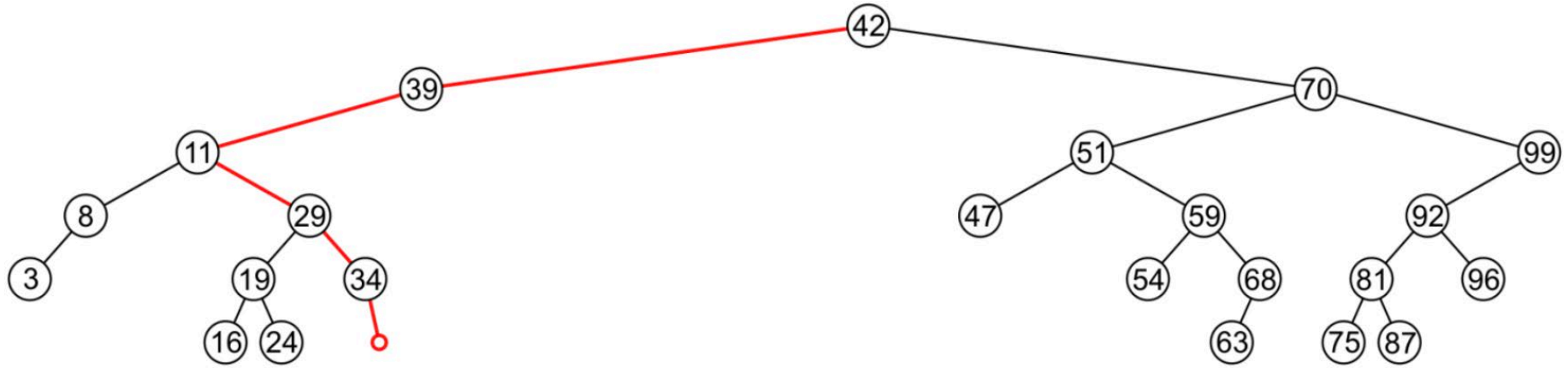
Search/Find

- Examine the root node and if we have not found what we are looking for:
  - If the object is less than what is stored in the root node, continue searching in the left sub-tree
  - Otherwise, continue searching the right sub-tree



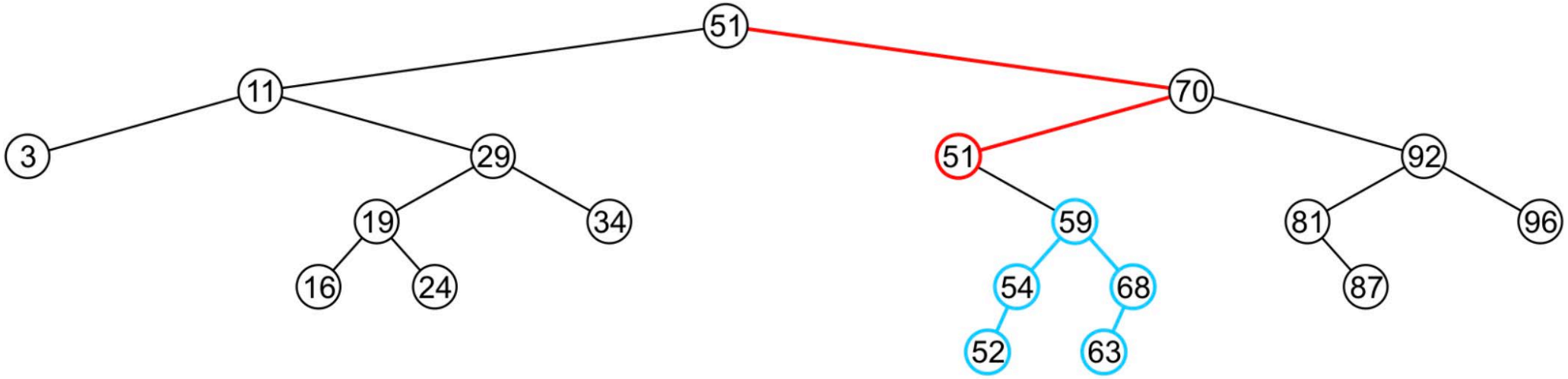
Insert

- Step through the tree like Search
  - If we find the object already in the tree, we will return
  - Otherwise, we will arrive at an empty node
  - The object will be inserted into that location



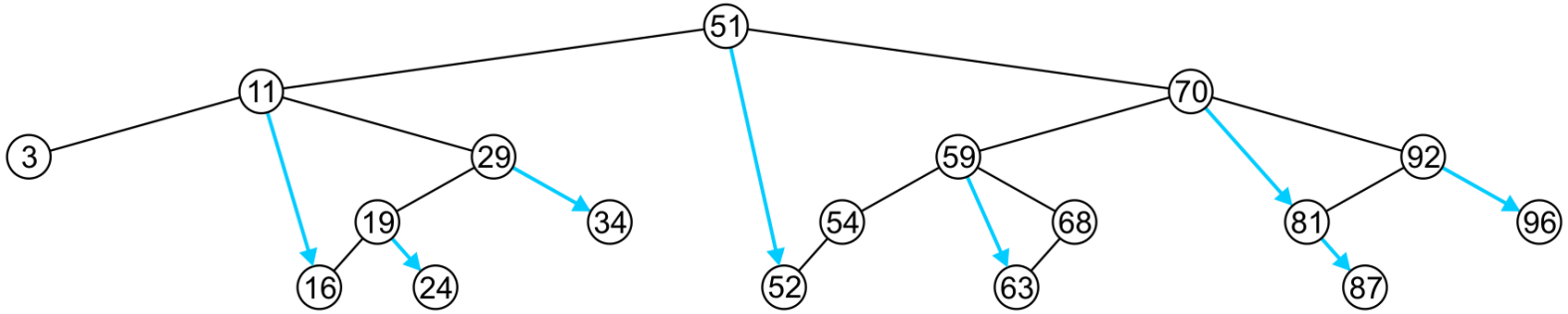
Erase

- For 0 children node, erase it directly.
- For 1 children node, promote the sub-tree associated with the child
- For 2 children node:
  - Replace it with the minimum object in the right sub-tree.
  - Erase that object (recursively) from the right sub-tree.

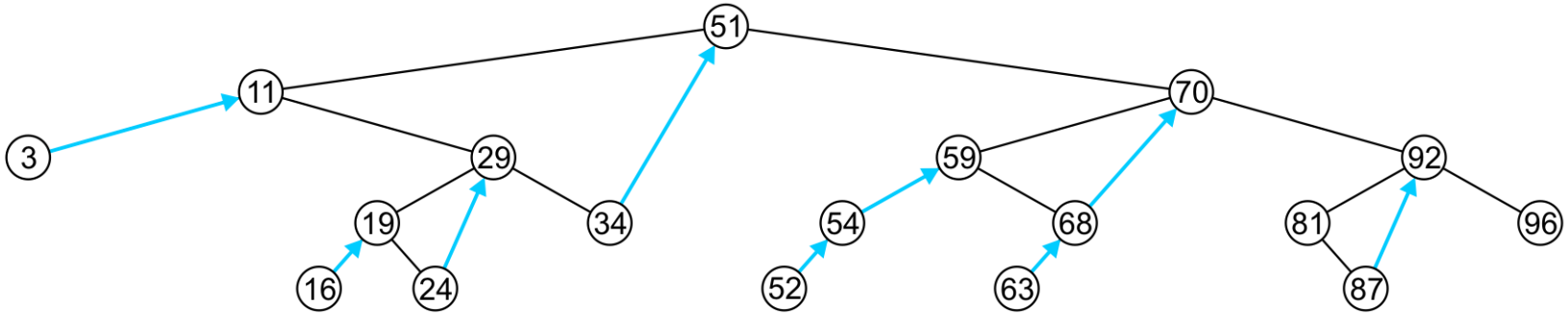


Next

- If the node has the right sub-tree, find the min in the right sub-tree:



- If it does not have the right sub-tree, find the first larger object (if any) that exists in the path from the node to the root:



AVL Tree

Definition

A binary search tree is said to be AVL balanced if:

- The difference in the heights between the left and right sub-trees is at most 1, and
- Both sub-trees are themselves AVL trees

Properties

- Maximum possible number of nodes in AVL tree of height h : $2^{(h+1)}-1$
- Minimum possible height of AVL Tree using n nodes : $\lfloor log_2 n \rfloor$
- Minimum number of nodes in AVL Tree of height h is given by a recursive relation: $N(h)=N(h-1)+N(h-2)+1$

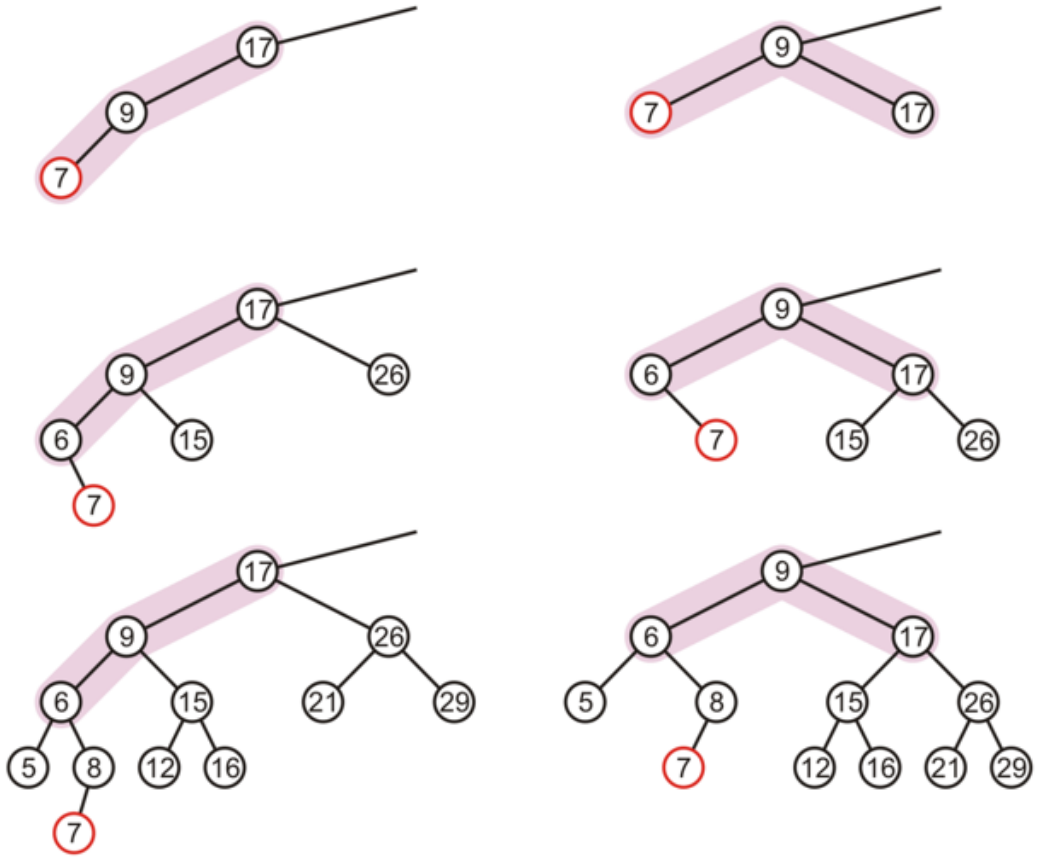
$N(h) = \Theta(\phi^h)$, $\phi = \frac{1+\sqrt{5}}{2}$

- Maximum possible height of AVL Tree using n nodes, is calculated using recursive relation: $N(h)=N(h-1)+N(h-2)+1$

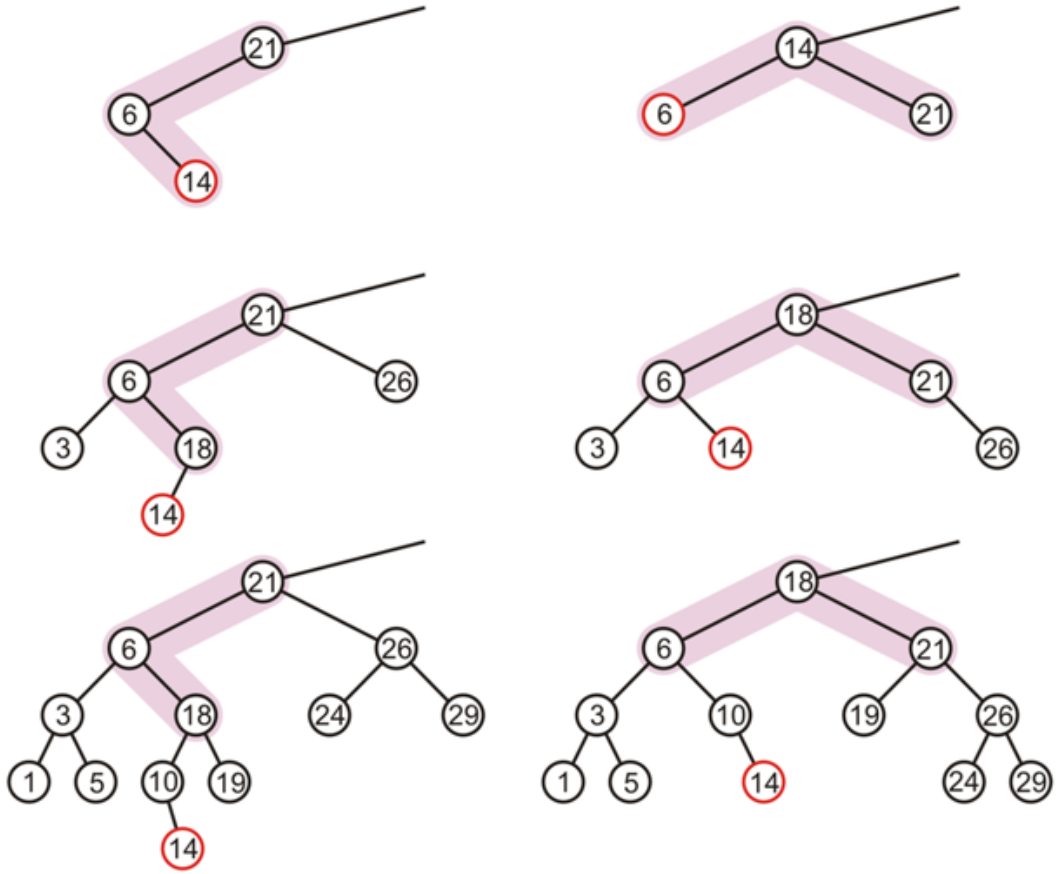
Example: What is the maximum height of any AVL tree with 15 nodes?

N(4) = N(3) + N(2) + 1 = 12, N(5) = N(4) + N(3) + 1 = 20

Maintain Balance: LL/RR



Maintain Balance: LR/RL



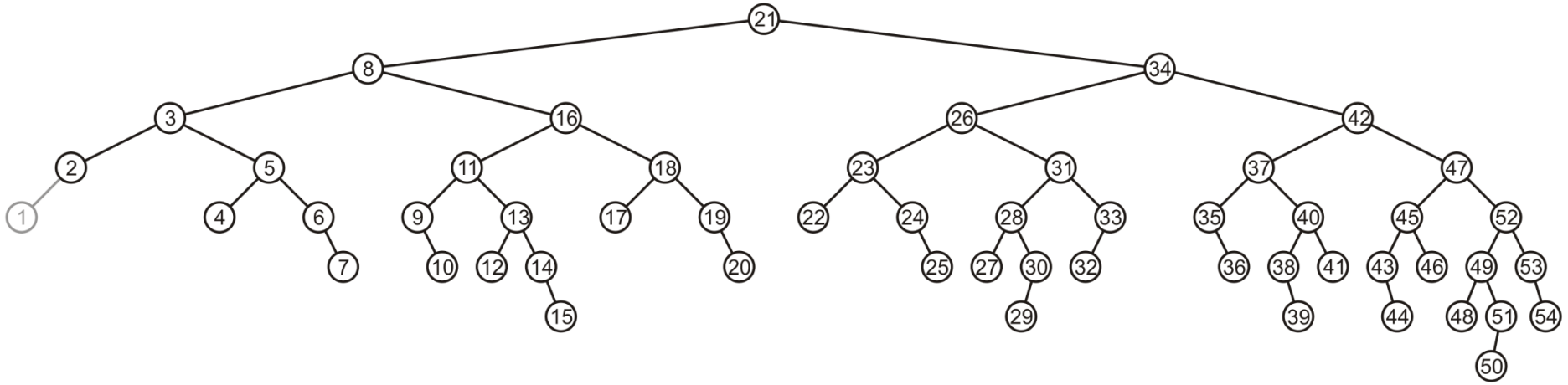
Deletion

Removing a node from an AVL tree may cause more than one AVL imbalance

- Like insert, erase must check if it caused an imbalance
- Unfortunately, it may cause O(h) imbalances that must be corrected
  - Insertions will only cause one imbalance that must be fixed

Deletion

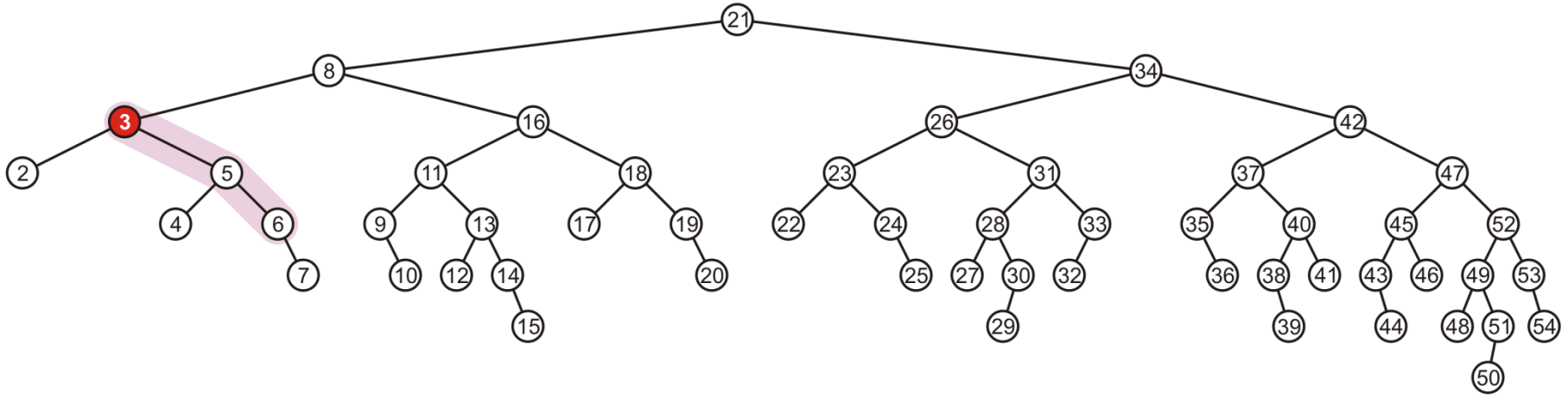
Suppose we erase the front node: 1



Deletion

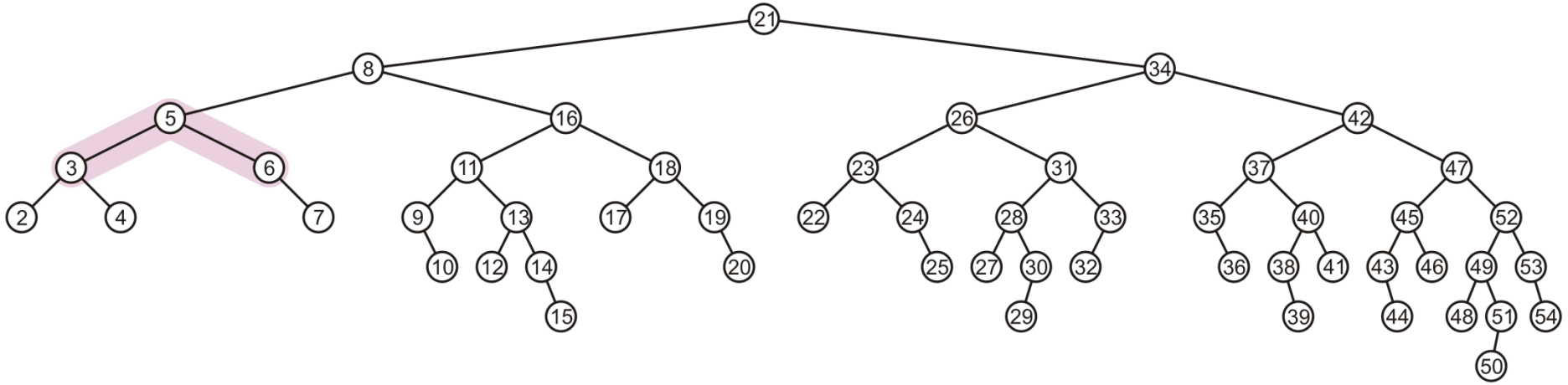
While its previous parent, 2, is not unbalanced, its grandparent 3 is

- The imbalance is in the right-right subtree



Deletion

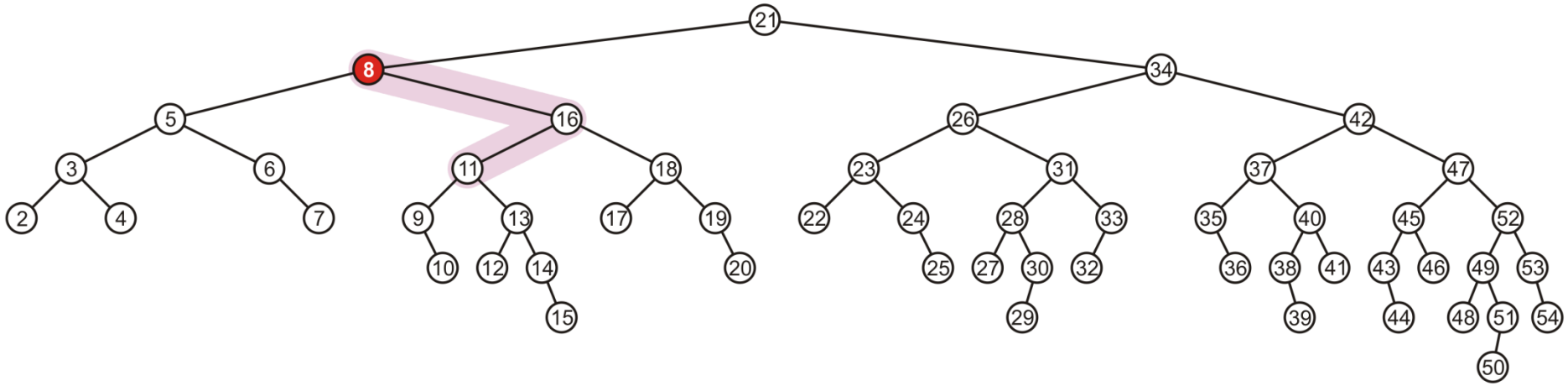
We can correct this with a simple balance



Deletion

Recursing to the root, however, 8 is also unbalanced

- This is a right-left imbalance



Time Complexity

Search

- Same way as BST: O(h). But for AVL, h = $\Theta(\ln(n))$
- The time complexity is O(ln(n))

Insertion

- May require one correction to maintain balance
- Each correction requires $\Theta(1)$ time
- The total time is $O(h) + O(1) = O(\ln(n))$

Deletion

- May require O(h) corrections to maintain balance
- Each correction require $\Theta(1)$ time
- Height h is $\Theta(\ln(n))$
- The total time is $O(\ln(n)) + O(\ln(n)) = O(\ln(n))$

1: (12') Multiple Choices

Each question has one or more correct answer(s). Select all the correct answer(s). For each question, you get 0 point if you select one or more wrong answers, but you get 1 point if you select a non-empty subset of the correct answers.

Note that you should write you answers of section 1 in the table below.

Question 1	Question 2	Question 3	Question 4	Question 5	Question 6
C	BD	BD	ACD	A	~~ACD~~ ABCD .

C **Question 1.** *Which of the followings are true?*

(A) For a min-heap, in-order traversal gives the elements in ascending order.

(B) For a min-heap, pre-order traversal gives the elements in ascending order.

(C) For a BST, in-order traversal gives the elements in ascending order.

(D) For a BST, pre-order traversal gives the elements in ascending order.

BD **Question 2.** *For a Binary Search Tree (BST), which of the followings are true?*

(A) If we erase the root node with two children, then it will be replaced by the maximum object in its right sub-tree.

(B) The cost for erasing the root node who has two children is $O(n)$.

(C) In a BST with N nodes, it always takes $O(\log N)$ to search for an specific element.

(D) For a BST, the newly inserted node will always be a leaf node.

BD **Question 3.** *Suppose we want to use Huffman Coding Algorithm to encode a piece of text made of characters. Which of the following statements are true?*

(A) Huffman Coding Algorithm will compress the text data with some information loss.

(B) The construction of binary Huffman Coding Tree using priority queue has time complexity $O(nlogn)$, where n is the size of the character set of the text.

(C) When inserting nodes into the priority queue, the higher the occurrence/frequency, the higher the priority in the queue.

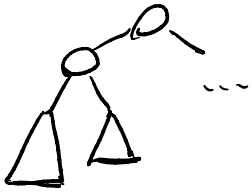
(D) The Huffman codes obtained must satisfy prefix-property, that is, no code is a prefix of another code.

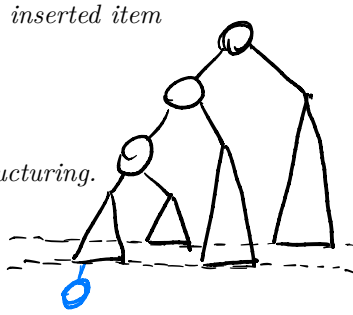
Question 4. *Which of the following statements are true for an AVL-tree?*

ACD *(A) Inserting an item can unbalance non-consecutive nodes on the path from the root to the inserted item before the restructuring.*

(B) Inserting an item can cause at most one node imbalanced before the restructuring.

(C) Removing an item in leaf nodes can cause at most one node imbalanced before the restructuring.





(D) Only at most one node-restructuring has to be performed after inserting an item.

A **Question 5.** *Consider an AVL tree whose height is h, which of the following are true?*

(A) This tree contains $\Omega(\alpha^h)$ *nodes, where* $\alpha = \frac{1+\sqrt{5}}{2}$.

(B) This tree contains $\Theta(2^h)$ *nodes.*

(C) This tree contains $O(h)$ *nodes in the worst case.*

(D) None of the above.

ABCD **Question 6.** *Which of the following is TRUE?*

(A) The cost of searching an AVL tree is $O(\log n)$ *but that of a binary search tree is* $O(n)$

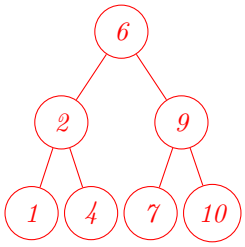
(B) The cost of searching an AVL tree is $O(\log n)$ *but that of a complete binary tree is* $O(n \log n)$

(C) The cost of searching a binary search tree with height h is $O(h)$ *but that of an AVL tree is* $O(\log n)$

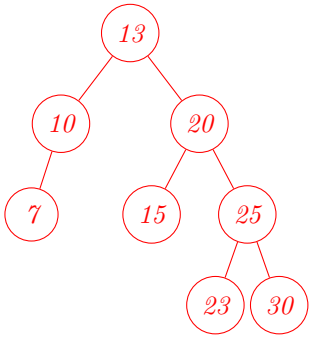
(D) The cost of searching an AVL tree is $O(nlogn)$ *but that of a binary search tree is* $O(n)$

2: (3'+3'+2'+3') BST and AVL Tree

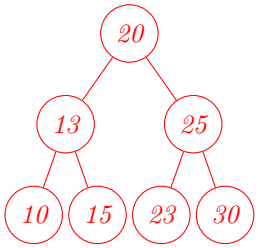
Question 7. *Draw a valid BST of minimum height containing the keys 1, 2, 4, 6, 7, 9, 10.*



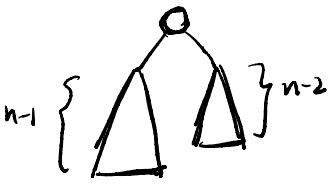
Question 8. *(1) Given an empty AVL tree, insert the sequence of integers* $15, 20, 23, 10, 13, 7, 30, 25$ *from left to right into the AVL tree. Draw the final AVL tree.*



(2) For the final AVL tree in the question (1), delete 7*. Draw the AVL tree after deletion.*



(3) For an AVL tree, define D = the number of left children - the number of right children, for the root. Then what is the maximum of D for an AVL tree with height n ?



$N_{max}(n-1)$ $-$ $N_{min}(n-2)$

$\downarrow$ $\downarrow$

$2^{n-1+1}-1=2^n-1$ $?$

$N(n)=N(n-1)+N(n-2)+1$

$N(n)+1=(N(n-1)+1)+(N(n-2)+1)$

$N(0)+1=2$ $N(1)+1=3$

$N(2)+1=5$ $N(3)+1=8$ ...

$\Rightarrow N(n)+1=Fib(n+3)$

$=\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+3}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+3}\right]$

$\Rightarrow N(n-2)=\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]-1$

3: (3'+3') Huffman Coding

After you compress a text file using Huffman Coding Algorithm, you accidentally spilled some ink on it and you found that one word becomes unrecognizable. Now, you need to recover that word given the following information:

Huffman-Encoded sequence of that word:

000101001110110100

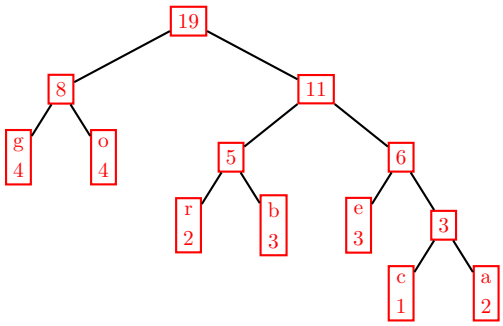
Frequency table that stores the frequency of some characters:

characters	a	b	e	g	l	o	r
frequency	2	3	3	4	1	4	2

Question 9. *Please construct the binary Huffman coding tree according to the given frequency table and draw the final tree below.*

Note: The initial priority queue is given as below. When inserting nodes into the priority queue, the priority of nodes with the same frequency follows "last in least priority".

l	a	r	b	e	g	o
1	2	2	3	3	4	4



Question 10. *Now you can "decompress" the encoded sequence and recover the original word you lost. Please write the original word below.*

googler

4: (9')Only-child

We define that the node is only-child if its parent node only have one children(Note: The root does not qualify as an only child). And we define a function for any binary tree T : $OC(T)$ = the number of only-child node.

Question 11. *(3')Prove the conclusion that for any nonempty AVL tree T with n nodes, OC(T)*$\leq \frac{1}{2}n$.

Ans: In an AVL tree just the leaves may be only-children, and therefore for every only-child in T, there existsa unique parent node that is not an only-child. The total number of only children in T is at most n/2

$A = \{$ only-child $\}$ $B = \{$ only-child's parent $\}$.

$A \cup B \subseteq T$

$\Rightarrow |A \cup B| \leq |T| = n$

$\Rightarrow |A| + |B| - |A \cap B| \leq n$

$|A| + |B| \leq n$

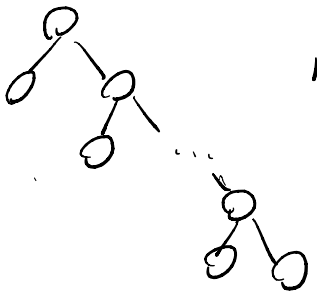
$OL(T) + OL(T) \leq n$

$OL(T) \leq \frac{n}{2}$

Question 12. *(3') For any binary tree T with n nodes, Is it true that if* $OC(T) \leq \frac{1}{2}n$ *then height*$(T) = O(\log n)$*? If true, prove it. If not, give a counterexample.*

$OL(T) = 0 \leq \frac{1}{2}n$

$n = 1 + 2h \Rightarrow h = \frac{n-1}{2} = \Theta(n)$



Question 13. *(3') For any binary tree T, Is it true that if there are* n_0 *only-children and they are all leaves, then height*$(T) = O(\log n_0)$*? If true, prove it. If not, give a counterexample.*

$h = n_0 + 1 = \Theta(n_0)$