

# Homework 3 - CS317 Spring 2019

Matt Fletcher

April 16, 2019

1. The goal of dynamic programming is to create polynomial time solutions to problems by holding solutions to previous subproblems in a table to be re-used in future uses of that subproblem. A problem is said to have overlapping subproblems if it can be broken down into subproblems which are used multiple times. Optimal substructure is when the globally optimal solution can be constructed from locally optimal solutions to subproblems.

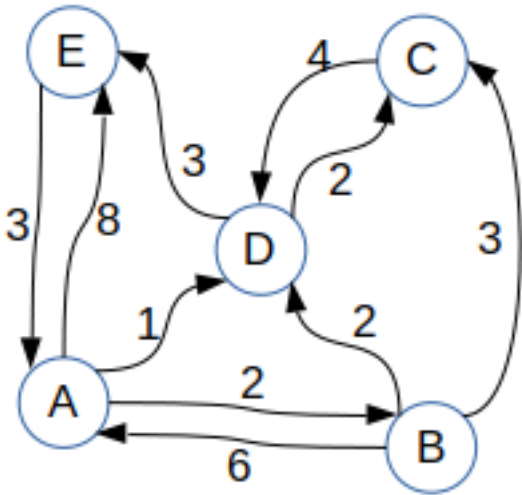
The four steps to dynamic programming approach are:

- (a) Characterize the structure of an optimal solution.
- (b) Recursively define the value of an optimal solution by expressing the solution of the original problem in terms of optimal solutions for smaller problems.
- (c) Bottom up computation, compute the value of an optimal solution in a bottom-up fashion by using a table structure.
- (d) Construct the optimal solution by expressing the solution of the original problem in terms of optimal solutions for smaller subproblems.

- 
2. (a) Step 1 is finding the optimal substructure and using it to construct an optimal solution to the subproblems.

- 
3. TODO finish in spreadsheet.

- 
4. Begin by visualizing the graph.



Now, write the beginning table.

	A	B	C	D	E
A	0	2	$\infty$	1	8
B	6	0	3	2	$\infty$
C	$\infty$	$\infty$	0	4	$\infty$
D	$\infty$	$\infty$	2	0	3
E	3	$\infty$	$\infty$	$\infty$	0

Update the table by seeing if any paths are shortened by going through A.

	A	B	C	D	E
A	0	2	$\infty$	1	8
B	6	0	3	2	14
C	$\infty$	$\infty$	0	4	$\infty$
D	$\infty$	$\infty$	2	0	3
E	3	5	$\infty$	4	0

- 
5. The knapsack problem is as follows: "Given  $n$  items of weights  $w_i$  and values  $v_j$ , and a knapsack of capacity  $W$ , find the most valuable subset of the items that fit into the knapsack. The dynamic approach to solving the knapsack problem consists of breaking the set of all the subsets of the first  $i$  items that fit the knapsack of capacity  $J$  into 2 categories: those that do not include the  $i^{th}$  item and those that do.
- 

6. Class P problems are those that have a polynomial time solution. This includes problems such as sorting a list or finding the largest element. Problems in class NP have non-polynomial time solutions but are able to be checked in polynomial time. For example, the traveling salesman problem can only be solved in non-polynomial time. The P-NP problem asks whether every problem which has a solution that can be checked in polynomial time also has a solution that can be solved in polynomial time. If P is shown to be equal to NP, then one major effect will be that modern RSA encryption will be easily able to be broken, given that RSA encryption relies on a solution only being able to be found in NP time.
- 

7. The backtracking technique is stopping a particular branch when a solution using that approach is no longer possible. In the N-Queens problem,  $n$  queens must be placed on an  $n \times n$  board such that no 2 queens are in the same row, column, or diagonal. To use backtracking, stop attempting to solve a board when it is impossible for the next queen to be placed.

Q	-	-	-
—	\		
—		\	
—			\

Q	-	-	-
—	\	Q	-
—	/	\	\
—		—	\

Not enough squares remain. Go back and try the next square for queen 2.

Q	-	-	-
—	\	-	Q
—		\	—
—	/		\

Q	-	-	-
—	\	-	Q
—	Q	\	—
—	/	\	\

Not enough squares remain. Go back to the first queen and try the next square.

-	Q	-	-
/	—	\	
	—		\
	—		

-	Q	-	-
/	—	\	Q
	—	/	\
	—		—

-	Q	-	-
/	—	\	Q
Q	—	/	\
—	—		—

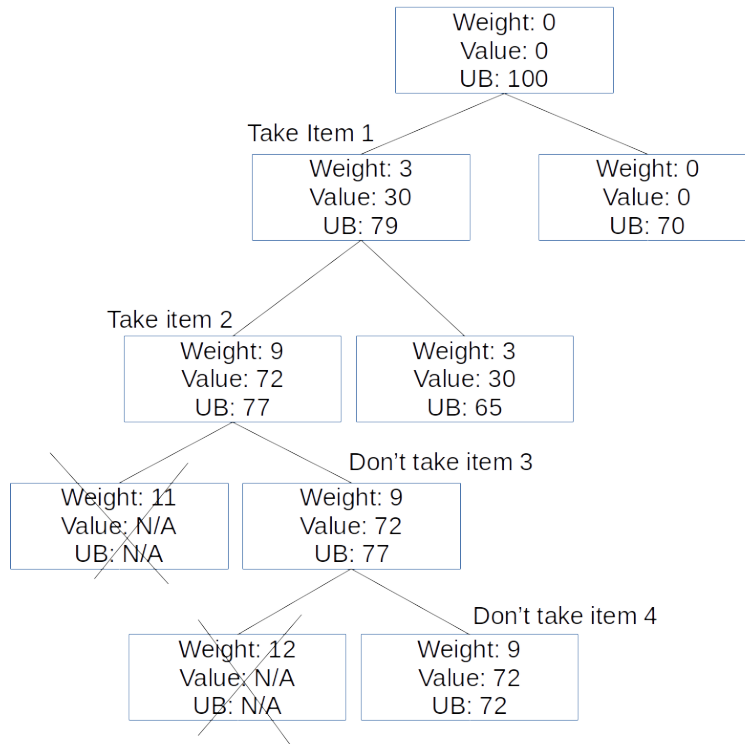
-	Q	-	-
/	—	\	Q
Q	—	/	\
—	—	Q	—

As all 4 queens have been placed, and no 2 are in the same row, column, or diagonal, this is the final solution to the problem.

\_\_\_\_\_

8. Begin by sorting the table on the weight/value ratio.

Item	Weight	Value	Val/Weight
1	3	30	10
2	6	42	7
3	2	10	5
4	3	15	5

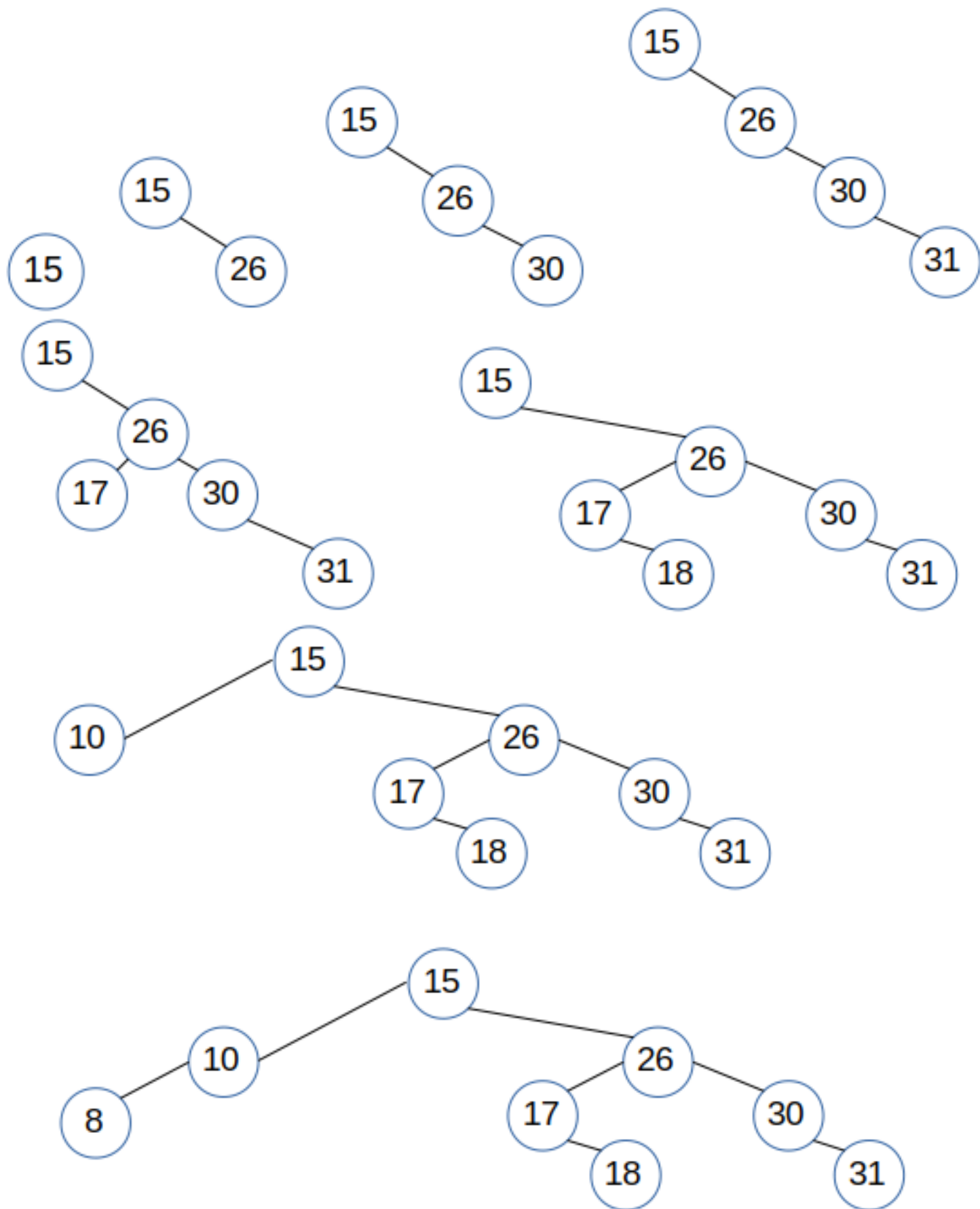


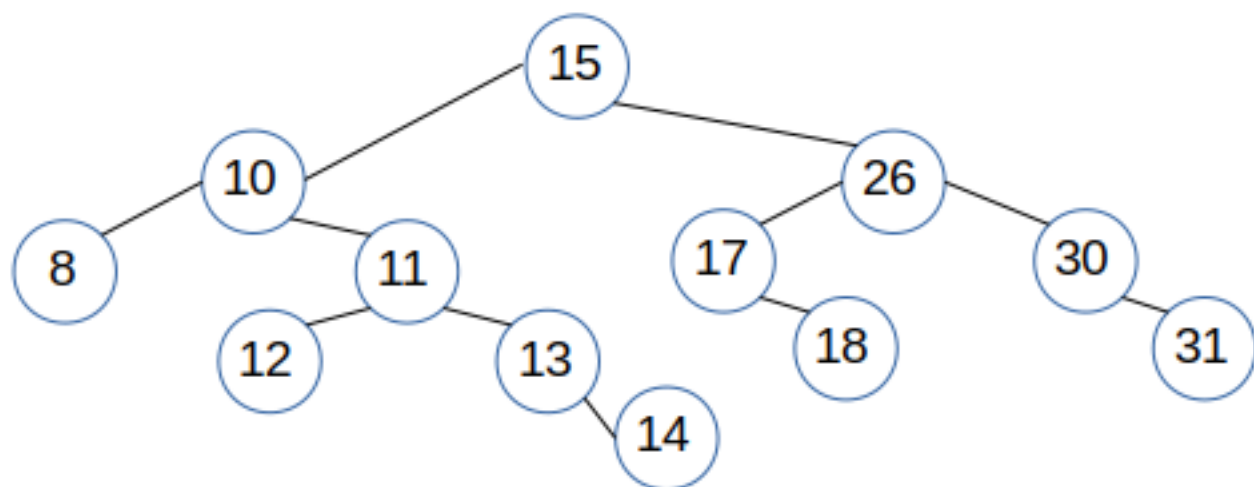
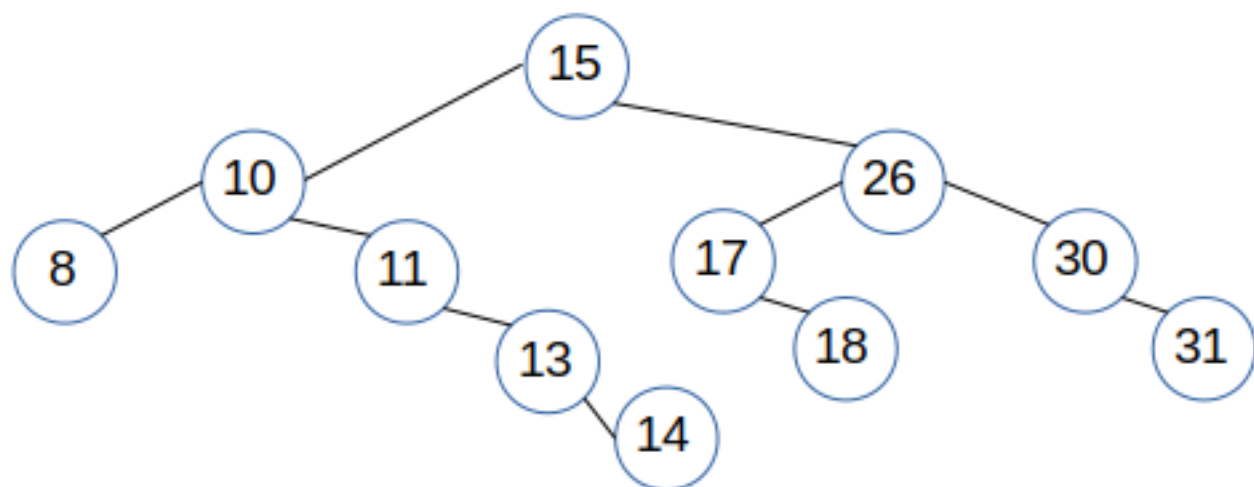
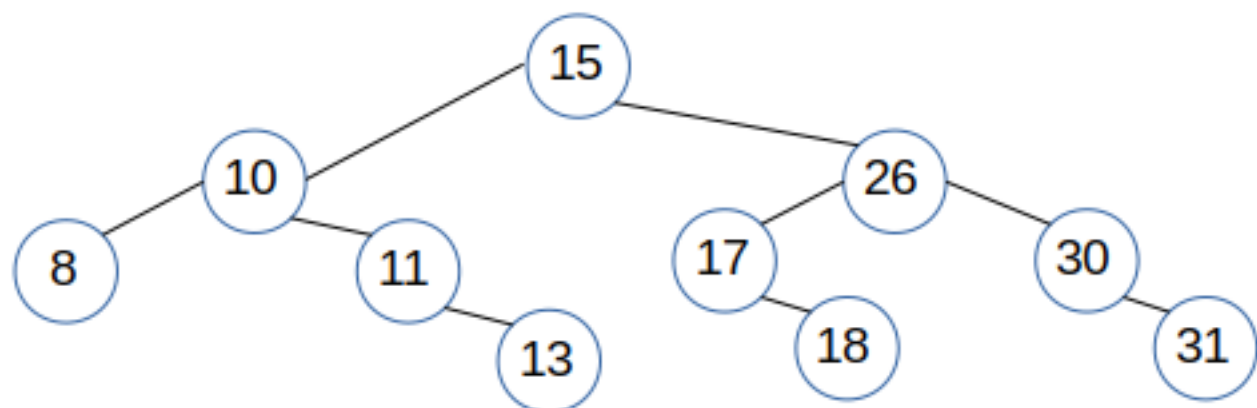
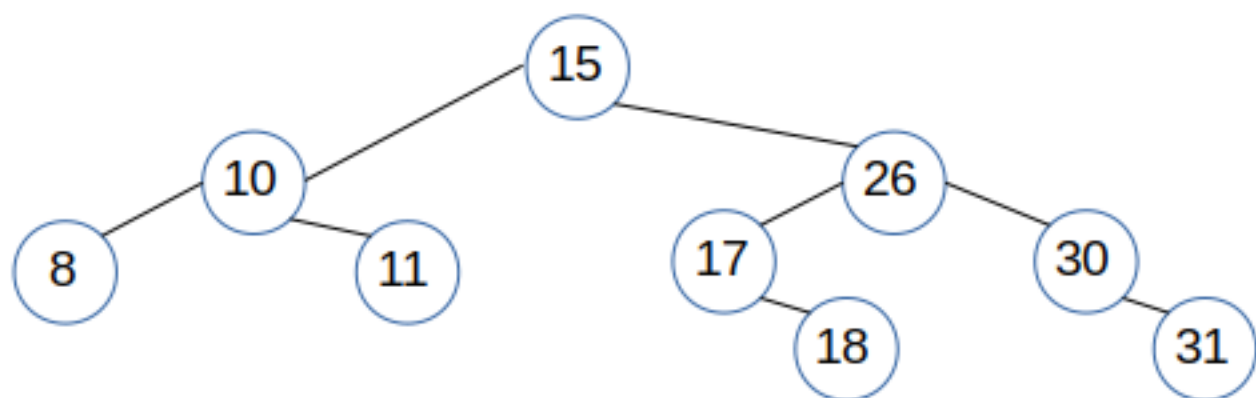
Therefore, the optimal solution takes items 1 and 2, with a value of 72.

9. The following table contains the output of a huffman encoder written in Python. TODO finish.

Char	Freq	Huffman code
'i'	7	10
't'	3	010
'o'	2	000
'n'	2	0111
'a'	2	0110
'b'	2	1101
'u'	2	1100
'h'	1	0010
'r'	1	00111
'f'	1	00110
'c'	1	11101
'l'	1	11100
'd'	1	11111
's'	1	11110

10. The following images show the construction of a binary search tree given the input list.

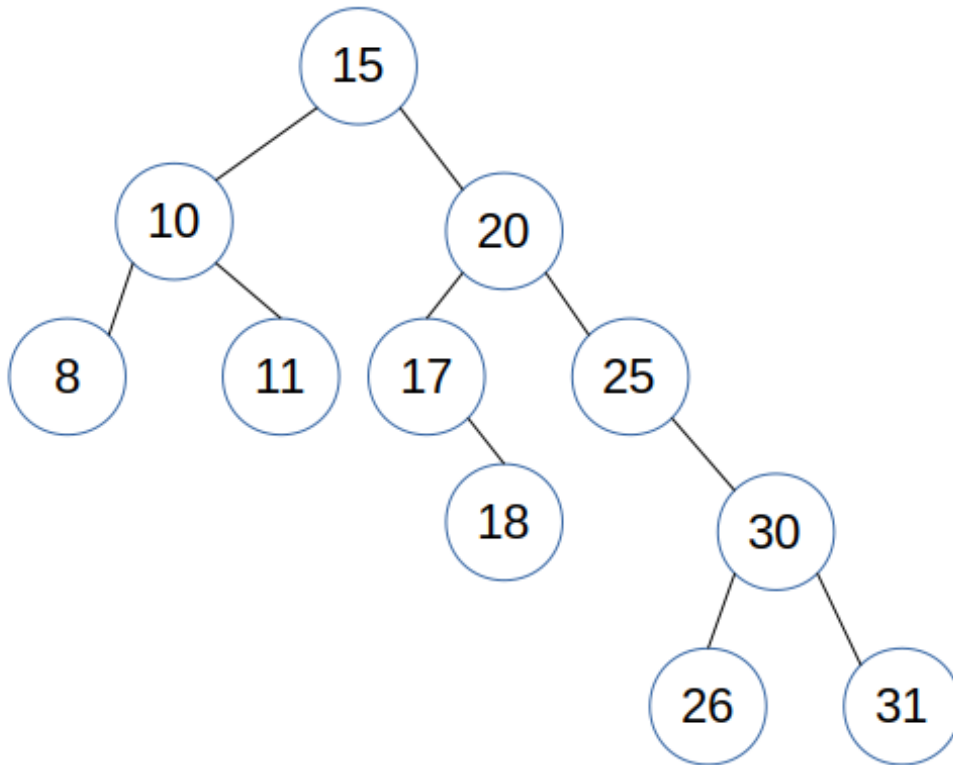




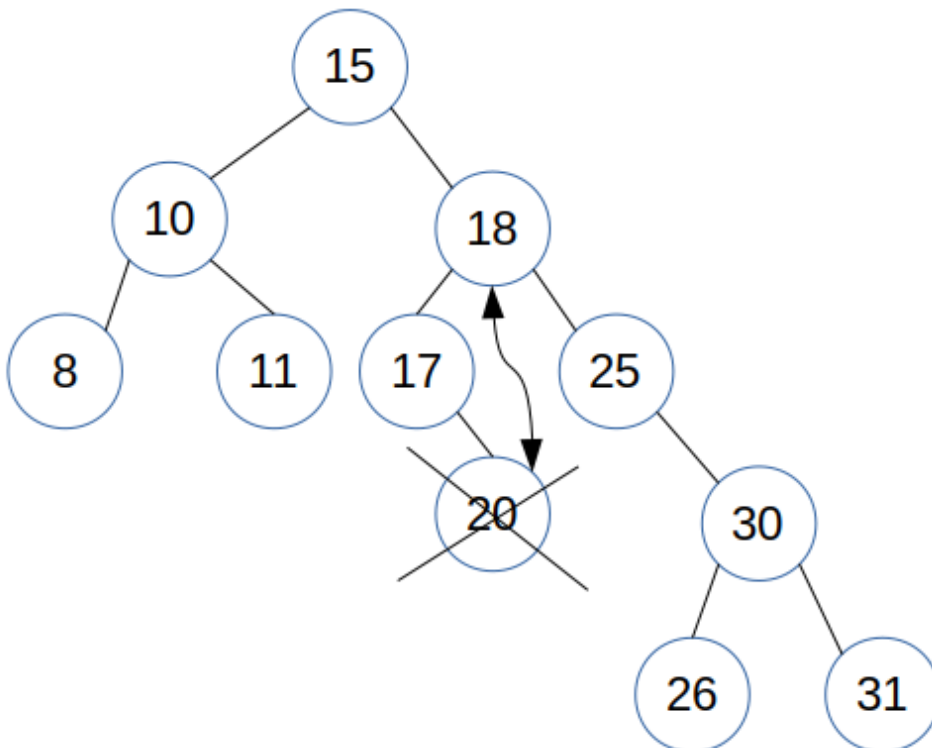
A BST insertion and deletion is least optimal when the input list is ordered. This turns the tree from a binary tree into a linked list. This changes the efficiency from  $O(\log n)$  to  $O(n)$ . A worst case input would be 1, 2, 3, 4, 5.

---

11. Here is the original tree.

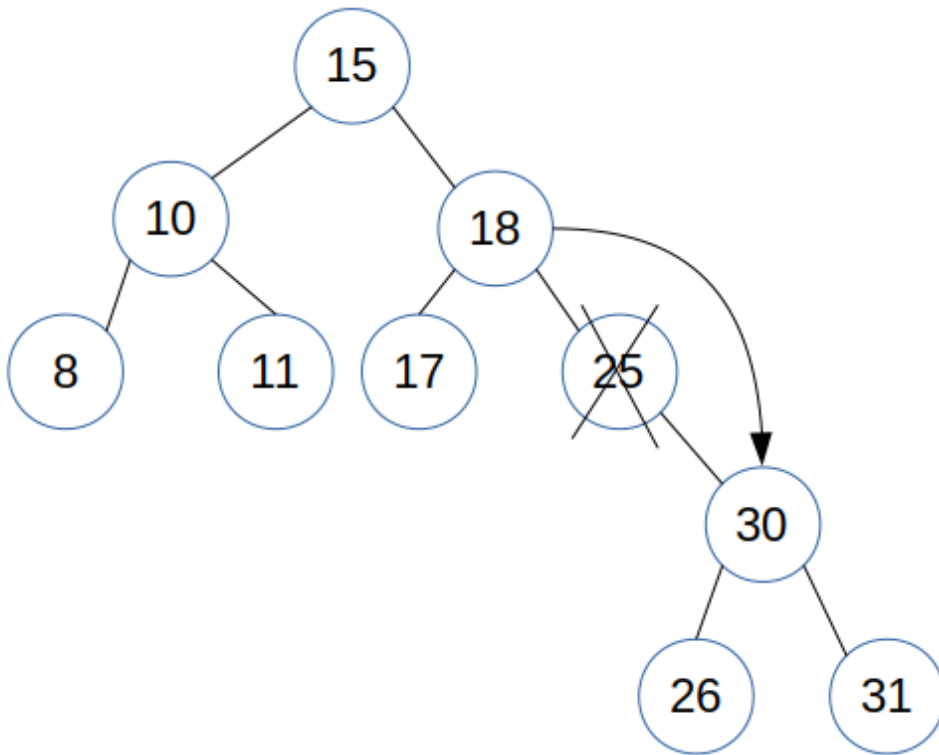


We wish to delete node #20. Because it has 2 children, we find the largest element in the left subtree, swap it with 20, and then delete that node.



Now, we wish to delete node #25. Because it only has 1 child, we simply point the parent node to point towards the

child of #25, as shown in the figure below:



In the end, the tree looks as follows:

