Binary Search Tree (3A)

Young Won Lim
6/5/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Binary Search Tree (1)

Binary search trees (BST),
ordered binary trees
sorted binary trees

are a particular type of **container**:
data structures that store "items"
(such as numbers, names etc.) in memory.

They allow fast **lookup**, **addition** and **removal** of items
can be used to implement either dynamic sets of items
lookup tables that allow finding an item by its **key**
(e.g., finding the phone number of a person by name).

https://en.wikipedia.org/wiki/Binary_search_tree

Binary Search Tree (2)

keep their **keys** in sorted order
lookup operations can use
the principle of **binary search**

allowing to skip searching half of the tree
each operation (**lookup**, **insertion** or **deletion**)
takes time proportional to **log n**

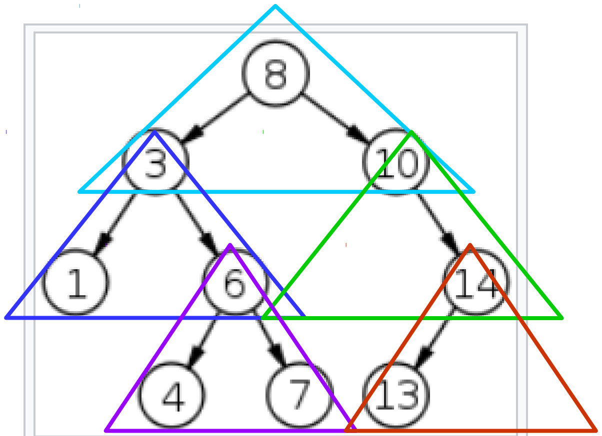
much better than the **linear time**
but slower than the corresponding operations
on **hash tables**.

https://en.wikipedia.org/wiki/Binary_search_tree

when **looking** for a **key** in a tree
or **looking** for a **place** to insert a new key,
they traverse the tree from root to leaf,
making comparisons to keys stored in the nodes
deciding to continue in the **left** or **right subtrees**,
on the basis of the comparison**.**

https://en.wikipedia.org/wiki/Binary_search_tree

Node, Left Child, Right Child



A binary search tree of size 9 and depth 3, with 8 at the root. The leaves are not drawn.

3 < 8 < 10

1 < 3 < 6

10 < 14

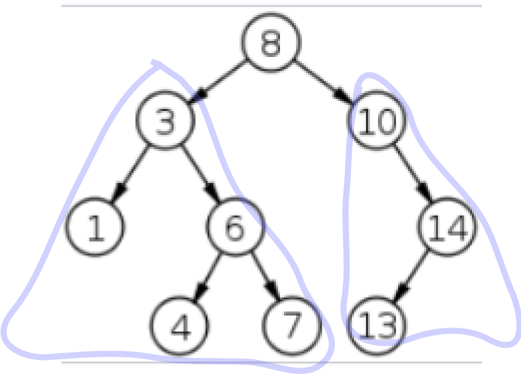
4 < 6 < 7

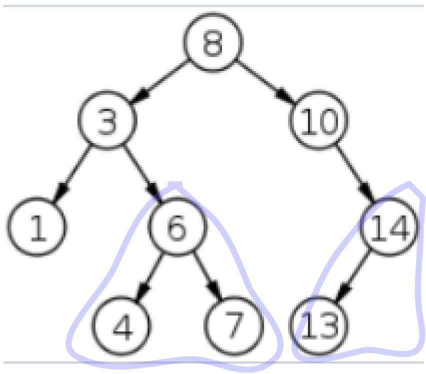
13 < 14

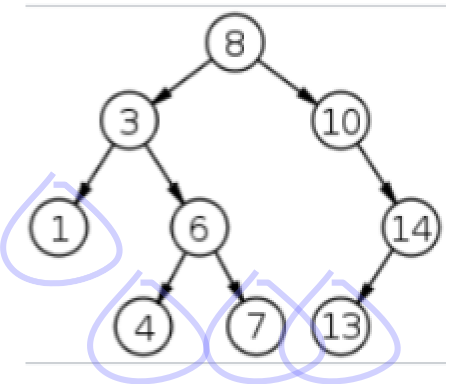
1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Subtrees





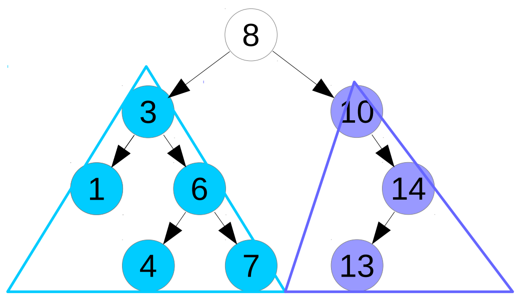


1, 3, 4, 6, 7, 8, 10, 13, 14

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

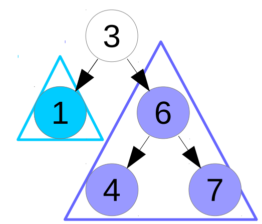
Node, Left Subtree, Right Subtree

$1, 3, 4, 6, 7 < 8 < 10, 13, 14$

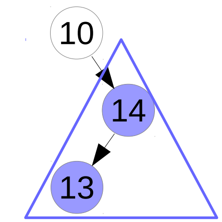


$1, \; 3, \; 4, \; 6, \; 7, \; 8, \; 10, \; 13, \; 14$

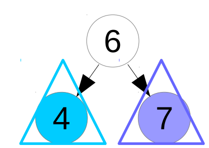
$1 < 3 < 4, 6, 7$



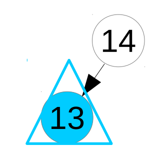
$10 < 13, 14$



$4 < 6 < 7$

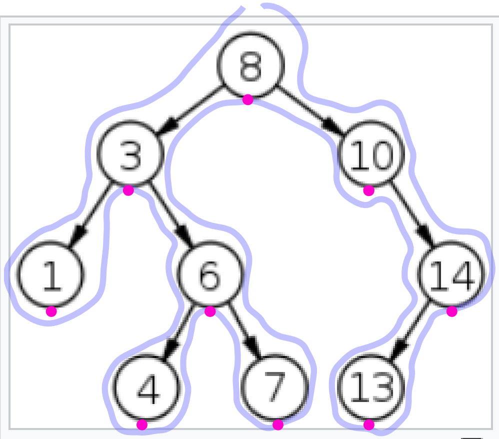


$13 < 14$



https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

In-Order Traversal

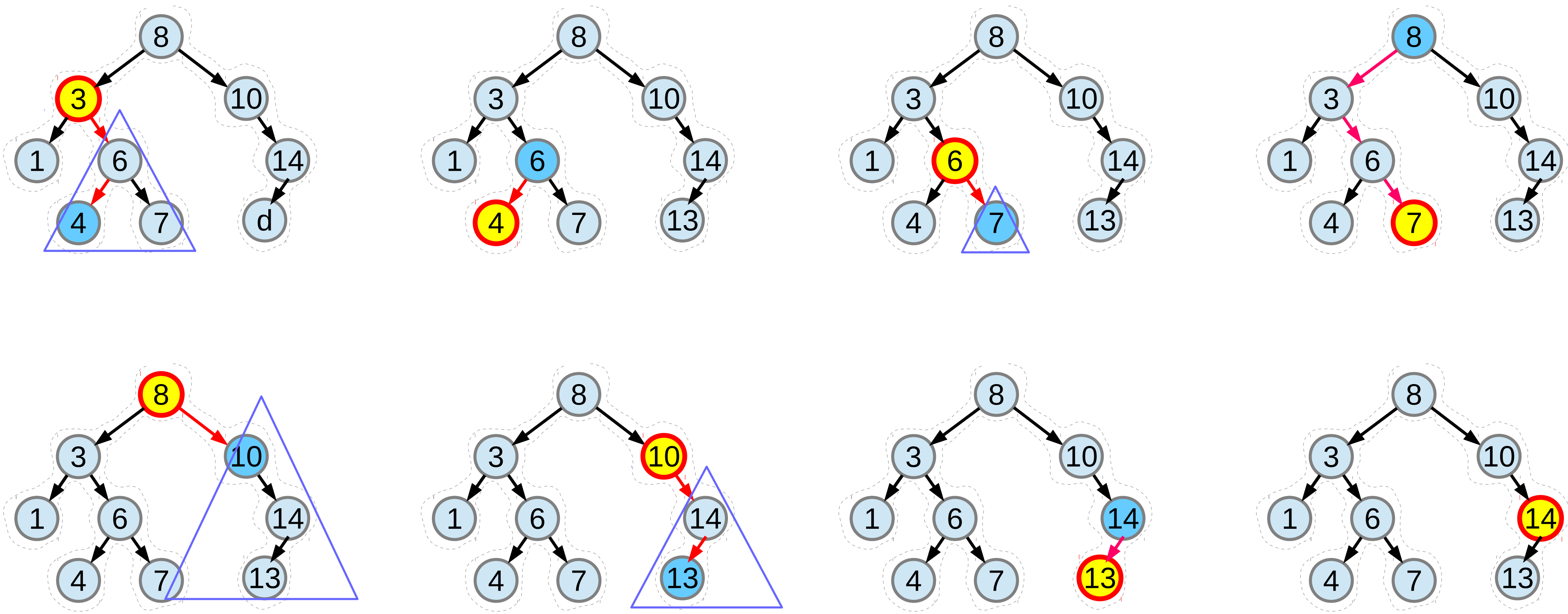


A binary search tree of size 9 and depth 3, with 8 at the root. The leaves are not drawn.

1, 3, 4, 6, 7, 8, 10, 13, 14

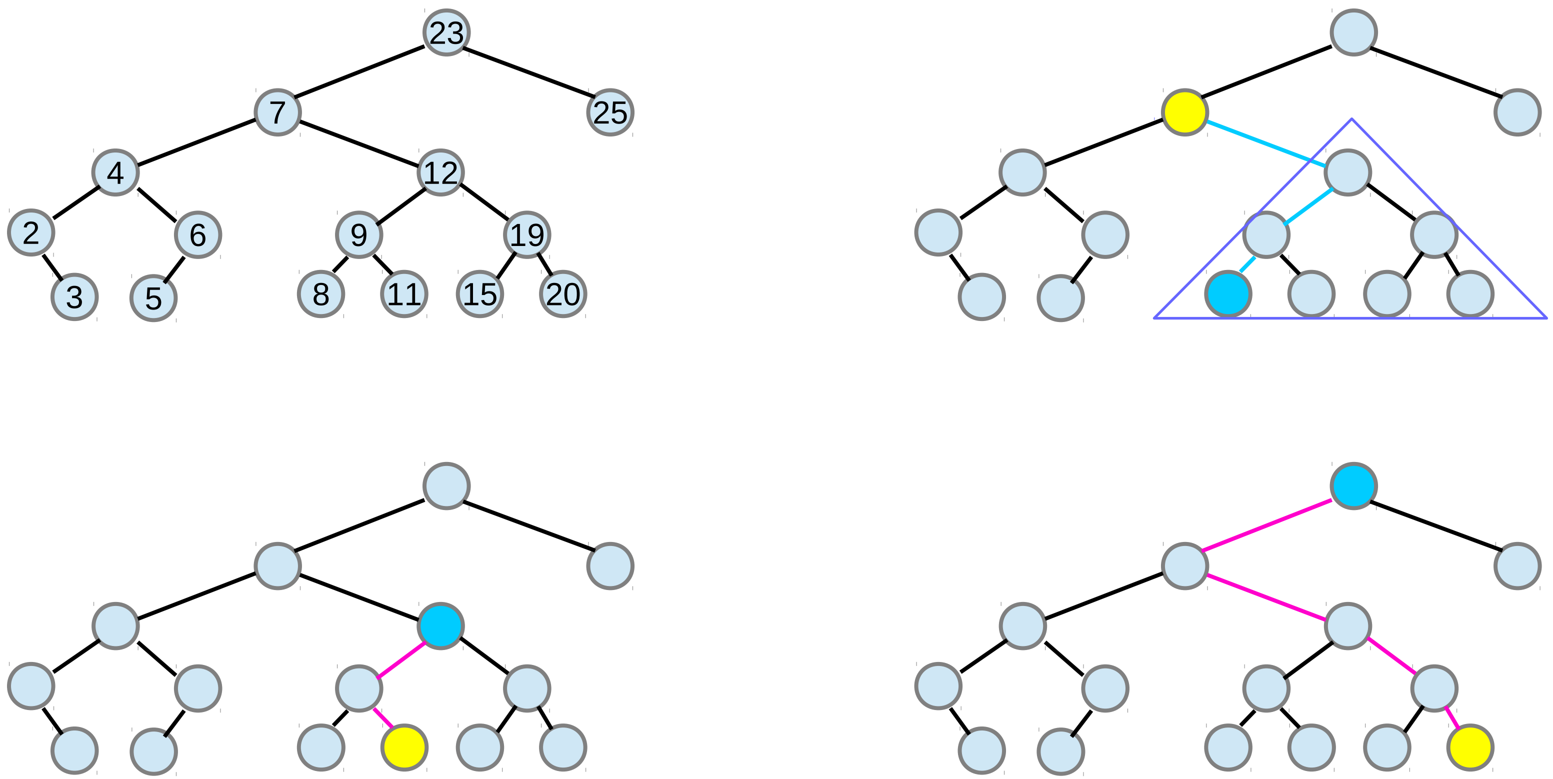
https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Successor Examples (1)



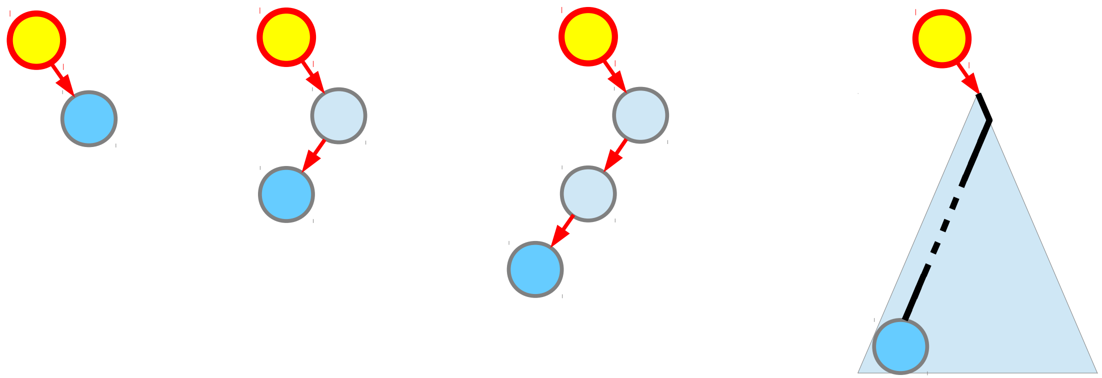
https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Successor Examples (2)

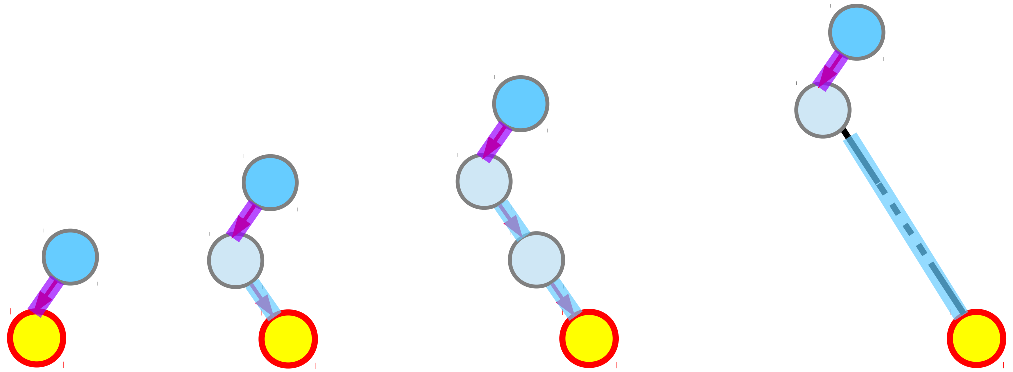


https://www.cs.rochester.edu/~gildea/csc282/slides/C12-bst.pdf

Successor Cases



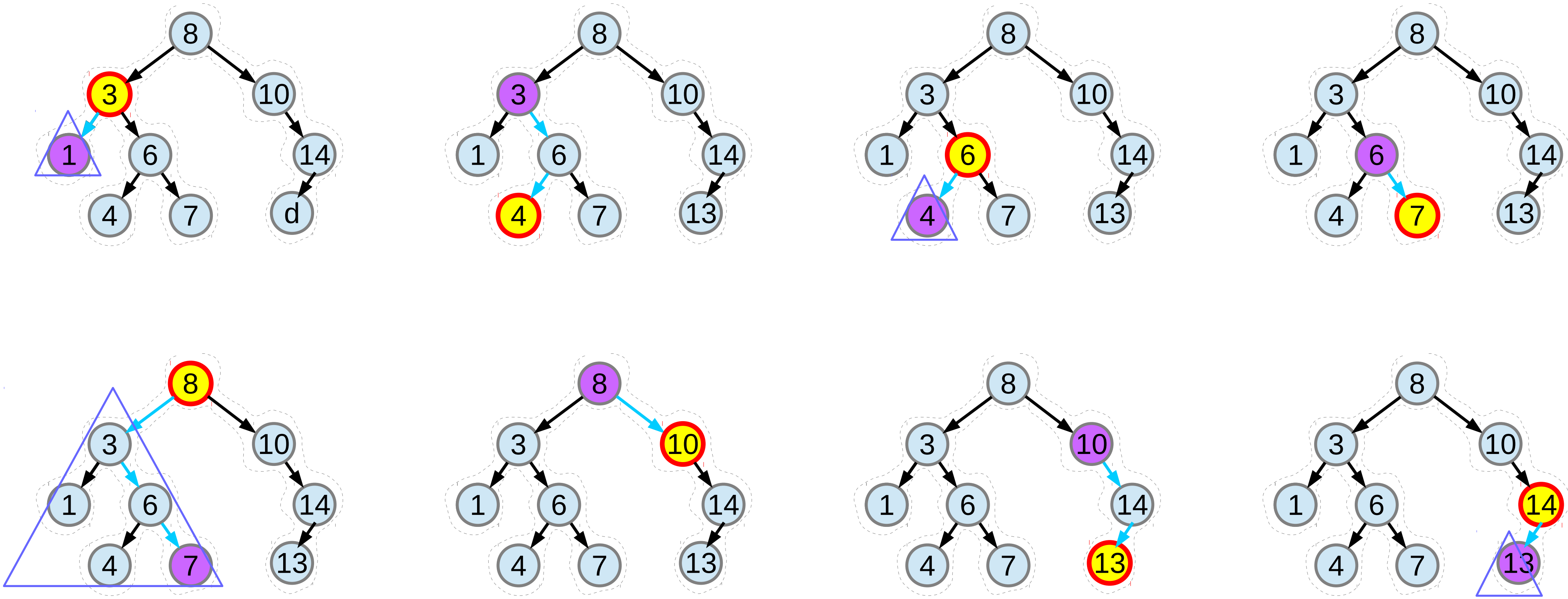
If the right child exists, then the minimum in the right subtree
– the leftmost node



the parent of the farthest node that can be reached by following only right edges backward.

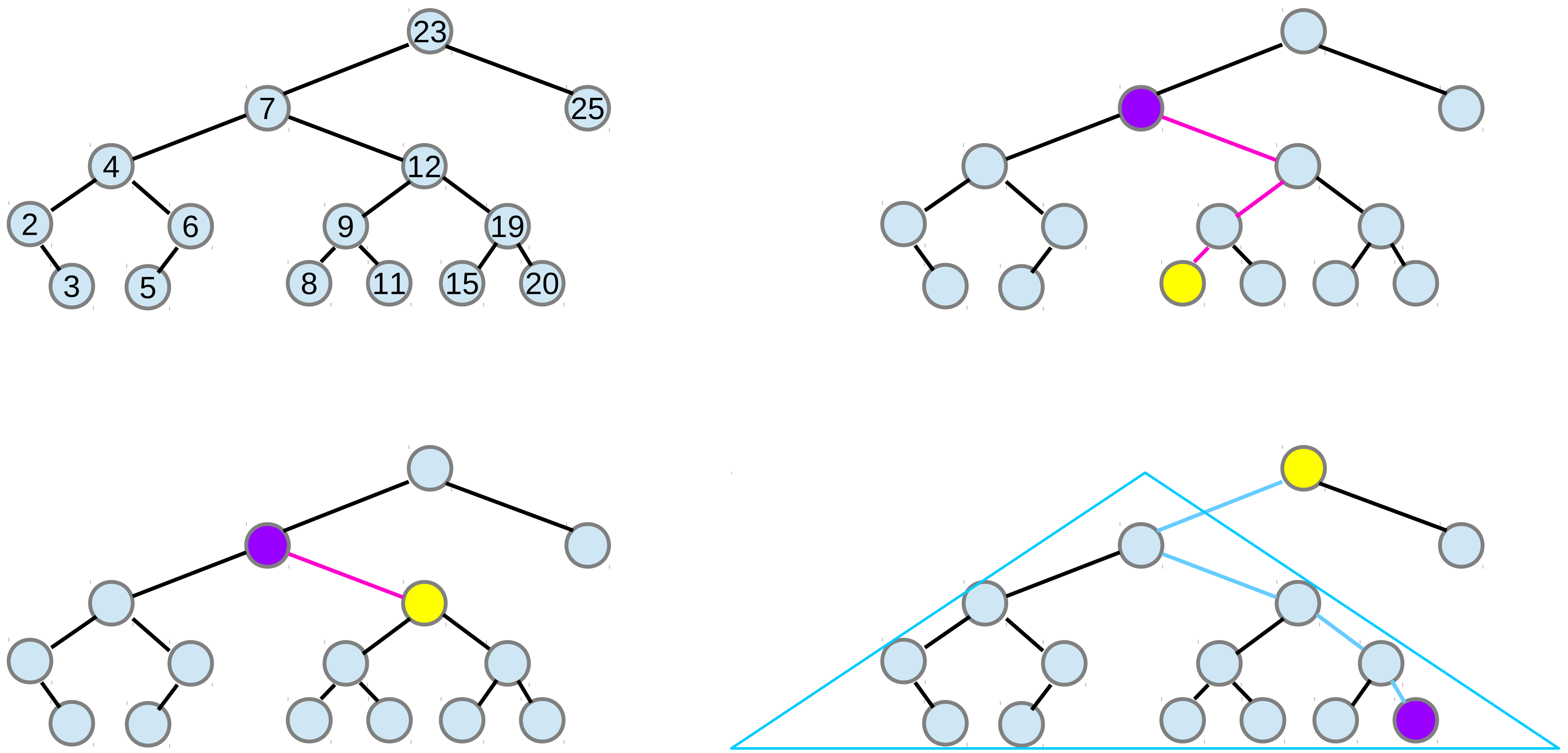
https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Predecessor Examples (1)



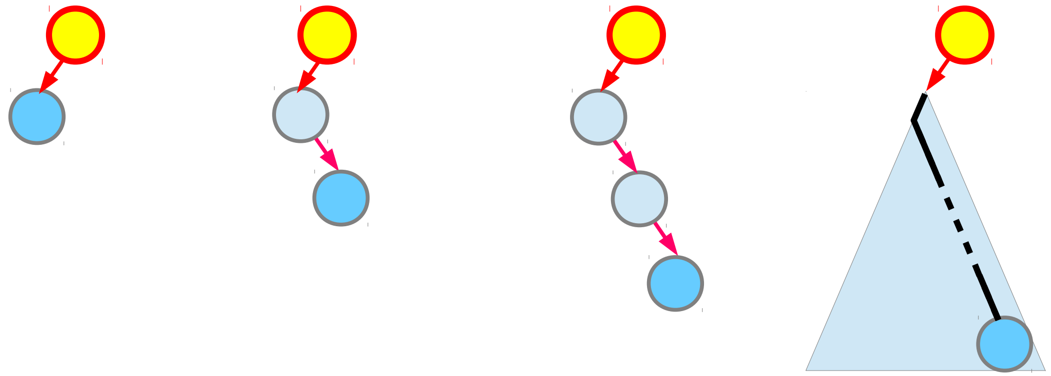
https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Predecessor Examples (2)

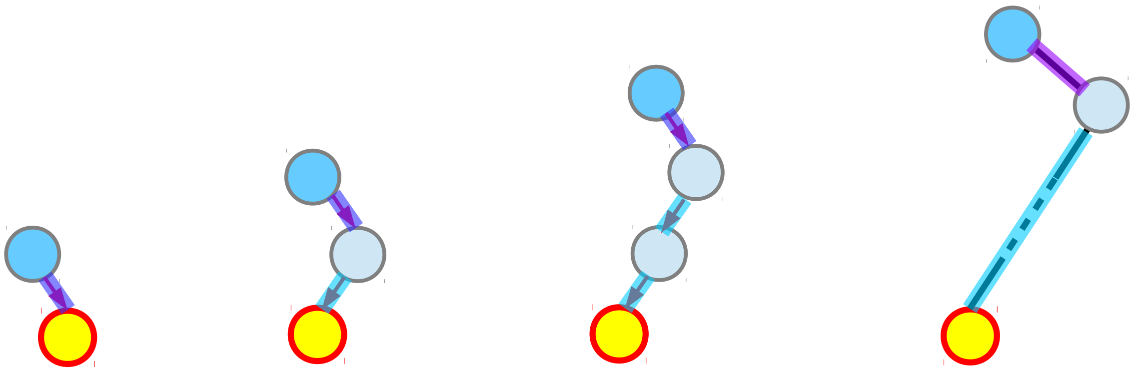


https://www.cs.rochester.edu/~gildea/csc282/slides/C12-bst.pdf

Predecessor Cases



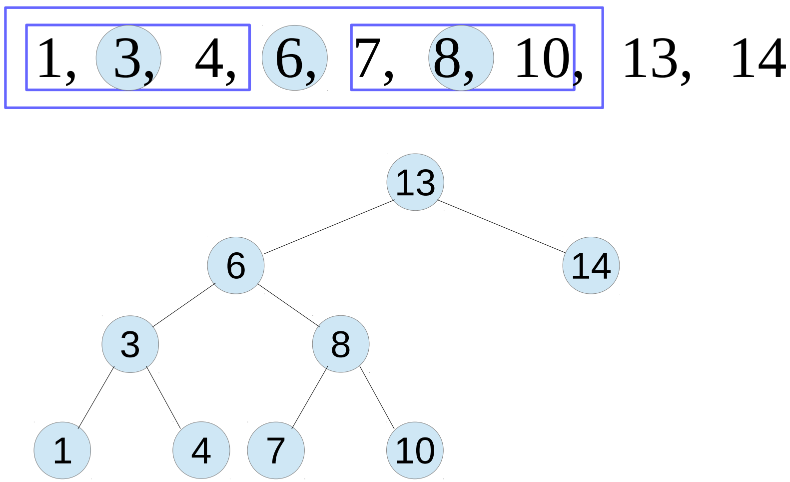
If the left child exists, then the maximum
in the left subtree
– the rightmost node

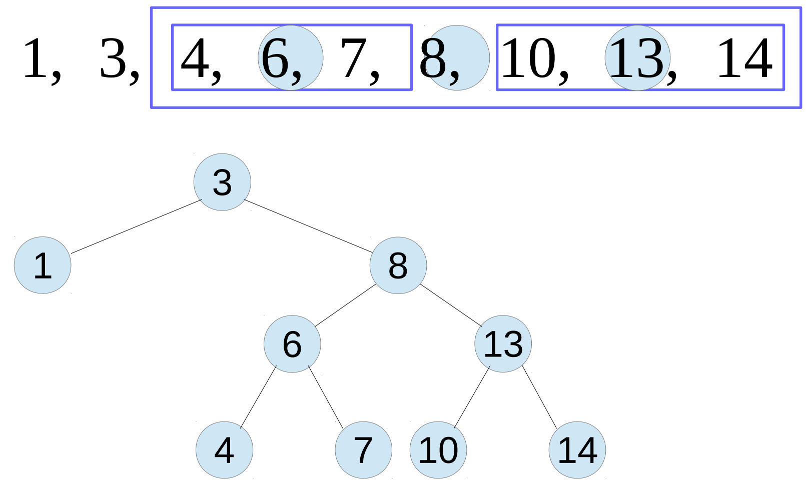


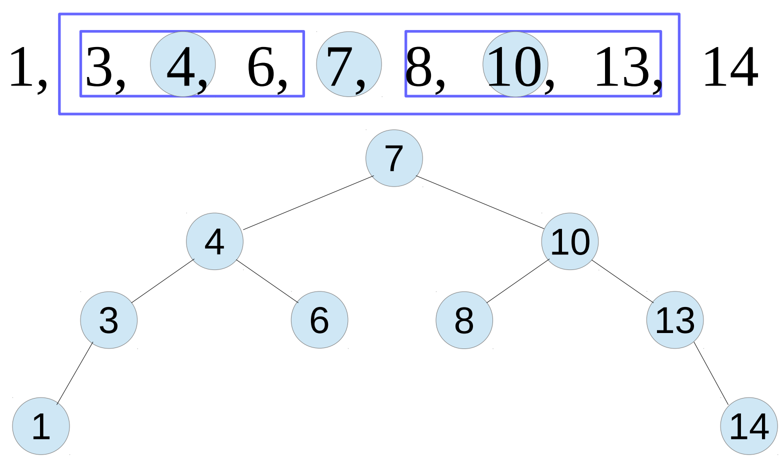
the parent of the farthest node that can be reached by following only left edges backward.

https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.html

Different BST’s with the same data

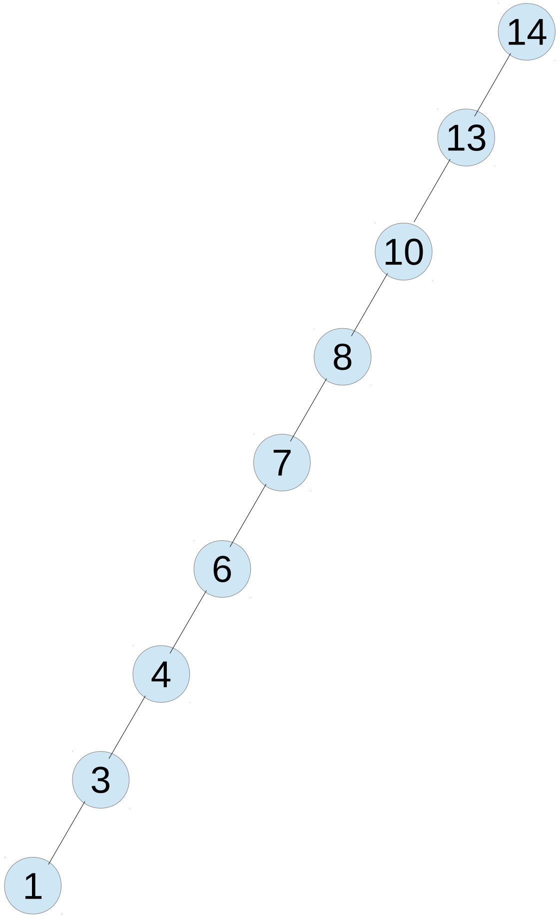




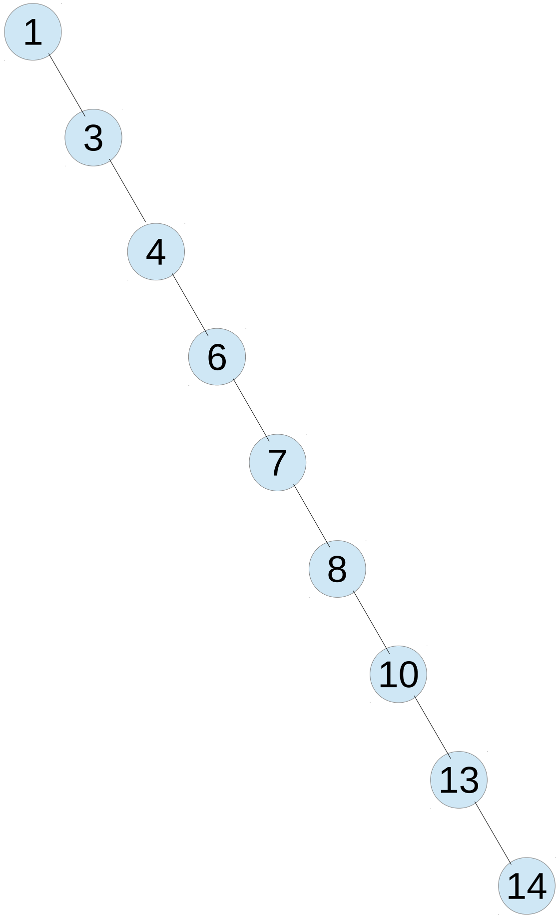


Unbalanced BSTs

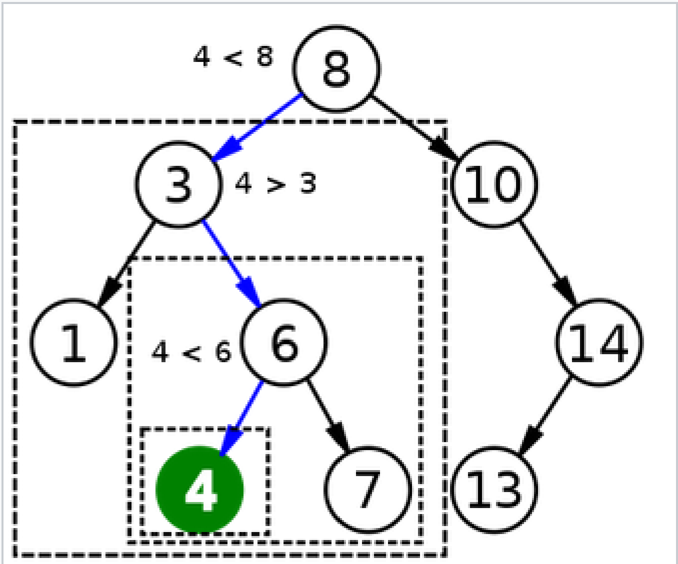
1, 3, 4, 6, 7, 8, 10, 13, 14



1, 3, 4, 6, 7, 8, 10, 13, 14



Binary Search on a Binary Search Tree



Binary search trees are searched using an algorithm similar to binary search.

1, 3, 4, 6, 7, 8, 10, 13, 14

1, 3, 4, 6, 7, 8, 10, 13, 14

1, 3, 4, 6, 7, 8, 10, 13, 14

https://en.wikipedia.org/wiki/Binary_search_algorithm

Insertion

Insertion begins as a **search** would begin;
if the key is not equal to that of the **root**,
we search the **left** or **right** subtrees as before.
Eventually, we will reach an **leaf node**
and **add** the new key-value pair
as its **right** or **left child**,
depending on the node's **key**.

In other words, we examine the **root**
and recursively insert the new node
to the **left** subtree if its key is less than that of the **root**,
or the **right** subtree if its key is greater than or equal to the **root.**

https://en.wikipedia.org/wiki/Binary_search_tree

Deletion

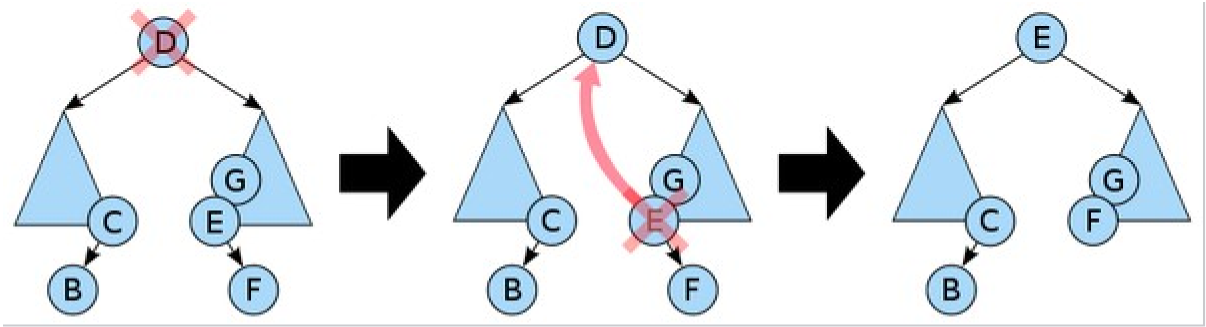
1. Deleting a **node** with no **children**:
   simply remove the node from the tree.

2. Deleting a **node** with one **child**:
   remove the node and replace it with its child.

3. Deleting a **node** with two **children**:
   call the **node** to be deleted D.
   Do not delete D.
   Instead, choose either its in-order **predecessor node**
   or its in-order **successor node** as replacement node E.
   Copy the user values of E to D
   If E does not have a **child**
      simply remove E from its previous parent G.
   If E has a **child**, say F, it is a right child.
      Replace E with F at E's parent.

https://en.wikipedia.org/wiki/Binary_search_tree

Deletion



Deleting a node with two children from a binary search tree.
First the leftmost node in the right subtree,
the in-order successor E, is identified.
Its value is copied into the node D being deleted.
The in-order successor can then be easily deleted
because it has at most one child.
The same method works symmetrically
using the in-order predecessor C.

https://en.wikipedia.org/wiki/Binary_search_tree

References

[1] http://en.wikipedia.org/
[2]