Grade 7/8 Math Circles
March 23rd, 2022
Graphy Theory and Search Algorithms Extension

Introduction

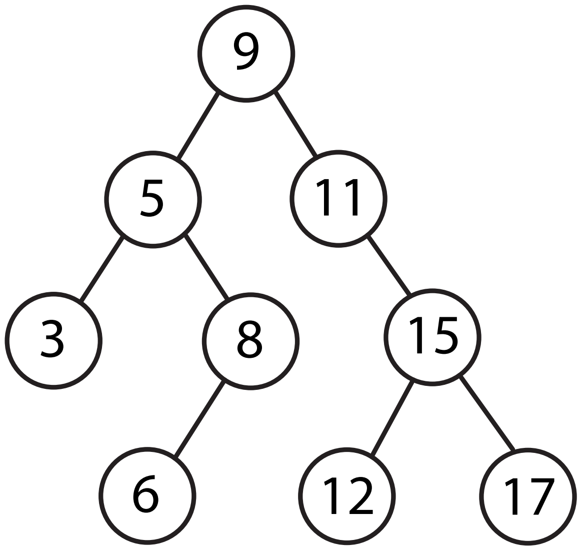
Last week we looked at graphs and two different search algorithms for graphs. Today, we will look at a type of graph that's optimized for searching.

Binary Search Trees

A **binary search tree** (or **BST**) is a type of tree graph with specific characteristics. Before we define the characteristics, we should define more basic things.

These types of graphs are called tree graphs because they have a "root" at the top and "leaves" at the bottom, with "branches" in between, like an upside down tree! In a tree graph, there are levels to the graph, where the root is the first level, and each level below is another level to the tree.

The vertices connected below a vertex are called its **children** and the vertex they're connected to would be called their **parent**. In the specific case for *binary* trees, a parent can only have 0, 1, or 2 children. Note how the word binary is used here to mean having two (or less) parts. Furthermore, a vertex can only have one parent. In the binary search tree below, the vertex 5 has two children, vertex 3 and vertex 8. We would call vertex 5 the parent of vertex 3 and vertex 8.



Now, we can consider what is special about binary search trees. As you can see in the example above, binary search trees have number labels for their vertices. The order and placement of the vertices is what's really special.

For each parent vertex, the "left" child will have a number that's *less than* the parent's number and the "right" child with have a number that's *greater than* the parent's number.

Furthermore, when we look at the whole left subtree of a vertex, the numbers will *all* be less than the vertex, and the whole right subtree will have numbers that are all greater than the vertex. Note that when we say **subtree**, it just means that we are looking at part of the tree instead of the whole tree. A *left subtree* would be like looking at the left child of a vertex like it's a root to the subtree. Similarly, a *right subtree* would be like looking at the right child of a vertex like it's a root to the subtree. For example, if we look at the right subtree of the vertex numbered 11 in the BST above, we'd treat vertex 15 like the root of the subtree and our subtree contains the vertices 15, 12, and 17.

Searching a BST

The ordering and placement of the vertices based on their numbers makes it very easy to search for a vertex with a specific number!

The search algorithm is as follows:

- Start the search by looking at the root.
- Compare the number of the vertex you're looking at to the number you are searching for.
  - If the number of the vertex is what you're looking for, we are done!
  - If the number of the vertex is *less than* the number you are searching for, look at the *right* child of the vertex. If there is no right child, the search is over and the tree does not contain the number you are looking for.
  - If the number of the vertex is *greater than* the number you are searching for, look at the *left* child of the vertex. If there is no left child, the search is over and the tree does not contain the number you are looking for.

So, when we are searching for a vertex with a specific number, either we will find it easily by using the "less than" or "greater than" rule, or we will determine that it doesn't exist in the tree.

Note that we can list the order of the vertices we look at in a similar way to what we did with BFS and DFS. We can use this to describe what steps we took in our solutions.

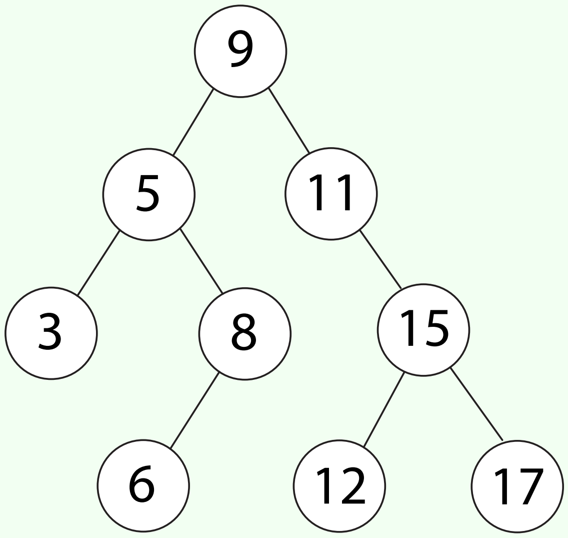
Example

Let's see how the algorithm works by searching for the following in the BST from earlier. For each, state the order that the vertices were looked at.

a) A vertex numbered 3.

b) A vertex numbered 12.

c) A vertex numbered 7.



(a) We start by looking at the root, vertex 9. Since $3 < 9$, we look at the left child, which is vertex 5. Since $3 < 5$, we again look at the left child, which is vertex 3 this time. We were looking for a vertex numbered 3, so we are done! We looked at the vertices in the order $9, 5, 3$.

(b) We start by looking at the root, vertex 9. Since $12 > 9$, we look at the right child, which is vertex 11. Since $12 > 11$, we again look at the right child, which is vertex 15. Now, since $12 < 15$, we look at the left child, which is vertex 12. We were looking for a vertex numbered 12, so we are done! We looked at the vertices in the order $9, 11, 15, 12$.

(c) We start by looking at the root, vertex 9. Since $7 < 9$, we look at the left child, which is vertex 5. Since $7 > 5$, we look at the right child, which is vertex 8. Since $7 < 8$, we look at the left child, which is vertex 6. Since $7 > 6$, we would look at the right child of vertex 6, but no right child exists. Therefore, there is not a vertex numbered 7 in this BST. We looked at the vertices in the order $9, 5, 8, 6$.

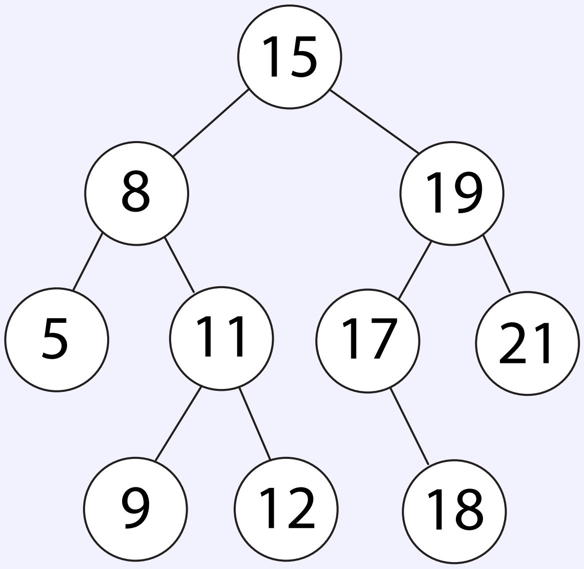
Exercise 1

Search for the following in the BST below. For each, state the order that the vertices were looked at.

a) A vertex numbered 11.

b) A vertex numbered 16.

c) A vertex numbered 5.



Exercise 1 Solution

(a) We start by looking at the root, vertex 15. Since $11 < 15$, we look at the left child, which is vertex 8. Since $11 > 8$, we look at the right child, which is vertex 11. We were looking for a vertex numbered 11, so we are done! We looked at the vertices in the order $15, 8, 11$.

(b) We start by looking at the root, vertex 15. Since $16 > 15$, we look at the right child, which is vertex 19. Since $16 < 19$, we look at the left child, which is vertex 17. Since $16 < 17$, we would look at the left child of vertex 17, but no left child exists. Therefore, there is no vertex numbered 16 in this BST. We looked at the vertices in the order $15, 19, 17$.

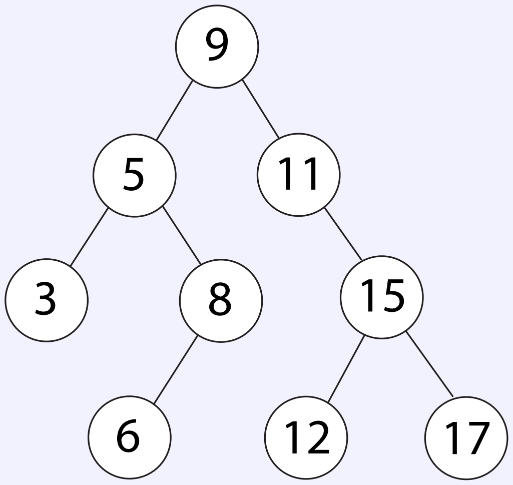
(c) We start by looking at the root, vertex 15. Since $5 < 15$, we look at the left child, which is vertex 8. Since $5 < 8$, we look at the left child again, which is vertex 5. We were looking for a vertex numbered 5, so we are done! We looked at the vertices in the order $15, 8, 5$.

Exercise 2

Instead of searching for a vertex with a specific number, imagine we want to add a vertex with a number that doesn't already exist in the tree.

a) What type of algorithm should we use for this? (Hint: Would one similar to the search algorithm work?)

b) Add a vertex numbered 7 to the following BST from earlier. What order did we look at the vertices and what does the BST look like afterwards?



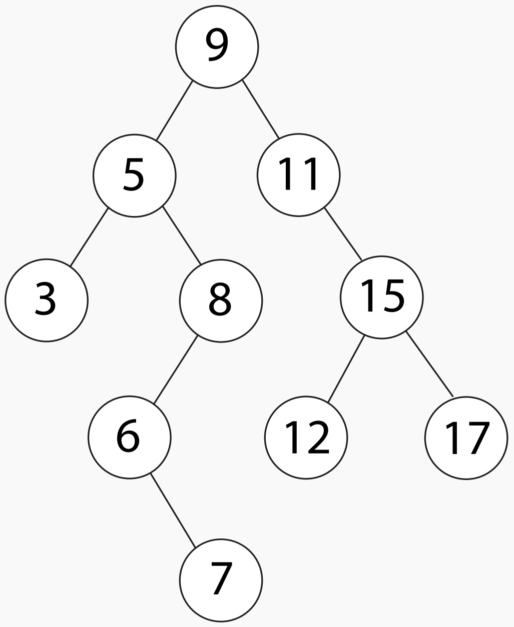
c) Add a vertex numbered 10 to the same BST from earlier. What order did we look at the vertices and what does the BST look like afterwards?

Exercise 2 Solution

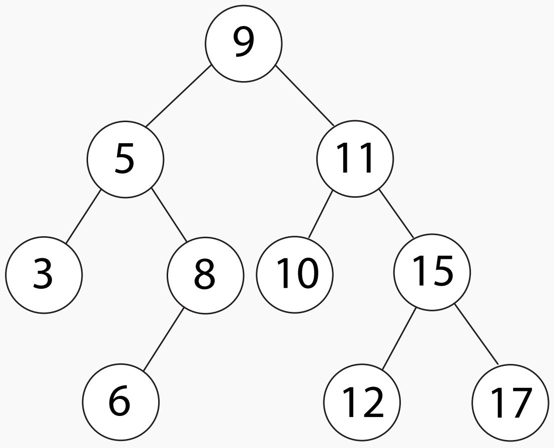
(a) We would follow the same steps as the search algorithm until we get to a non-existent child. This will always happen because we will never be adding a vertex with a number that already exists in the tree. Then, we simply add the vertex as the non-existent child, since that's where it should be placed according to the algorithm.

(b) We start by searching until we find a non-existent child, looking at the root first, which is vertex 9. Since $7 < 9$, we look at the left child, which is vertex 5. Since $7 > 5$, we look at the right child, which is vertex 8. Since $7 < 8$, we look at the left child, which is vertex 6. Since $7 > 6$, we would look at the right child of vertex 6, but no right child exists.

Therefore, now that we have discovered which child the search leads us to, we put the vertex numbered 7 in this place. So, now vertex 6 has a right child, vertex 7. We looked at the vertices in the order $9, 5, 8, 6$ and the BST will look like the following:



(c) We start by searching until we find a non-existent child, looking at the root first, which is vertex 9. Since $10 > 9$, we look at the right child, which is vertex 11. Since $10 < 11$, we would look at the left child of vertex 11, but no left child exists. Therefore, we put the vertex numbered 10 as its left child. We looked at the vertices in the order $9, 11$ and the BST will look like the following:


9
5
11
3
8
10
15
6
12
17