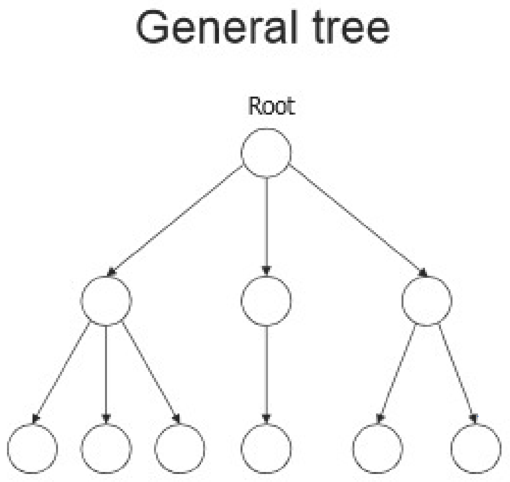
(1) Tree

- So far we have learned about linear data structures, such as arrays and linked lists.
    ○

- A tree is not a linear data structure
    ○

Hierarchical data structure.

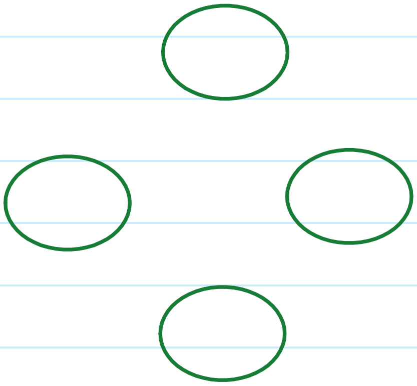


Each element is called a node.

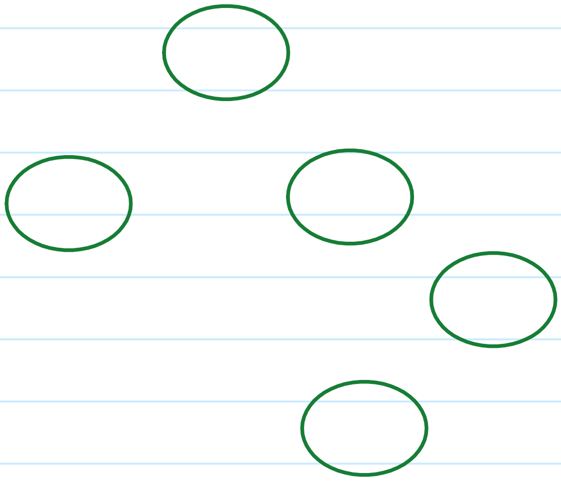
- Connected by *edges*
- Parent/child relationship
- Cannot have cycles
- Cannot have disconnected parts

https://entcheva.github.io

Not a tree:

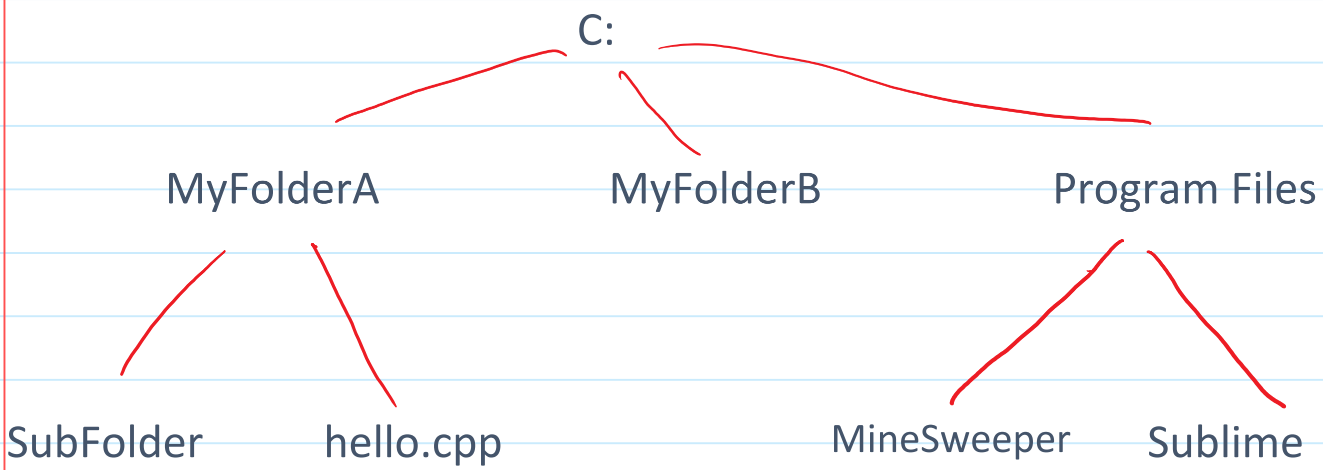


Not a tree:



(2) Tree Usage in Your Personal Computer

File structure:



C:

MyFolderA

MyFolderB

Program Files

SubFolder

hello.cpp

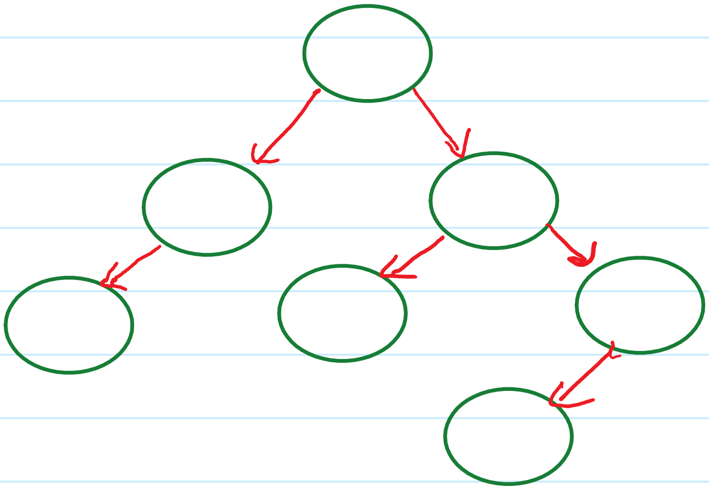
MineSweeper

Sublime

(3) Binary Tree

A binary tree is a special case of a general tree:

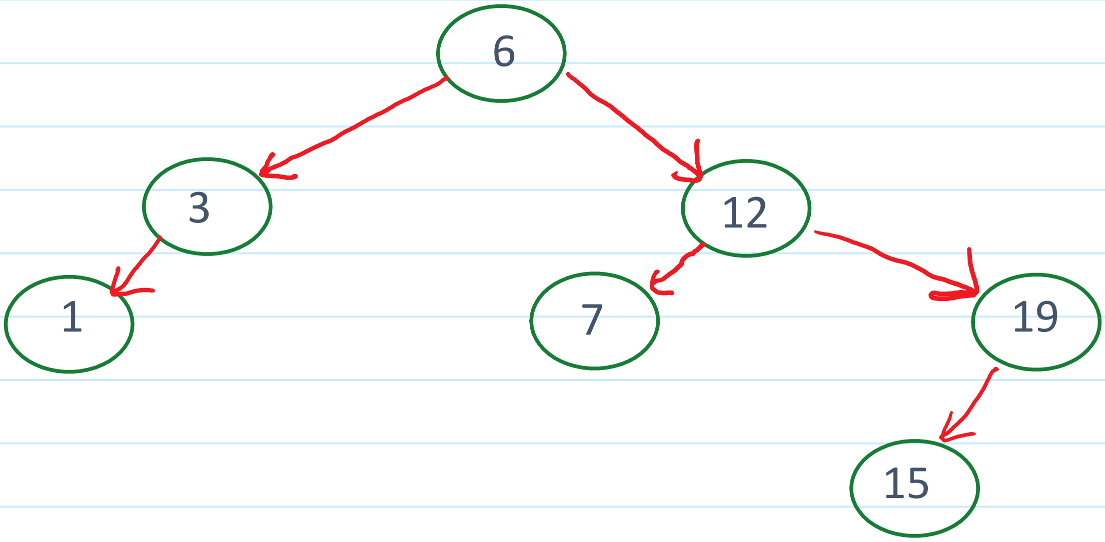
- in a binary tree, each node has exactly 2 children
    - in terms of implementation a node that has no children actually has two null children



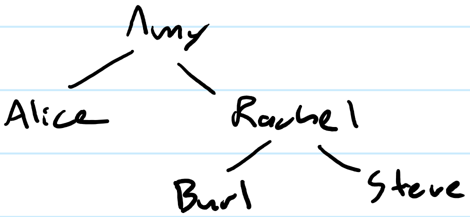
(4) Binary Search Tree (BST)

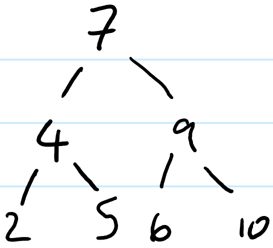
A special case of a Binary Tree, in which the data is ordered. For any node in the tree:

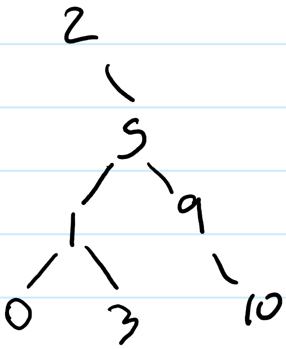
1. the nodes in the left subtree have key values less than the node value
2. nodes in the right subtree will have key values greater than or equal than the parent node value



Could apply to string keys:

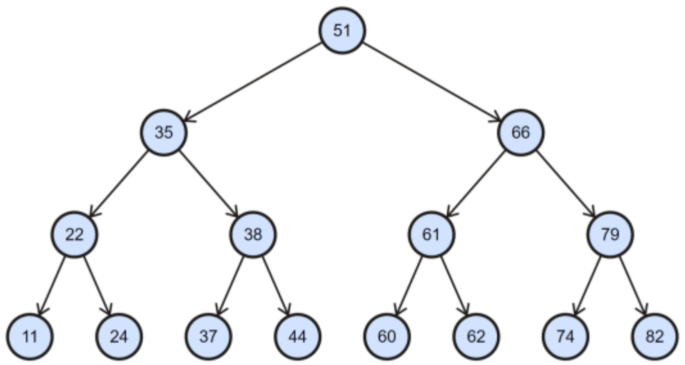


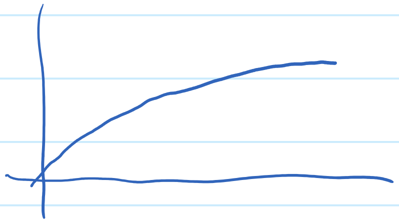




(5) BST height vs N

Consider the following BST:





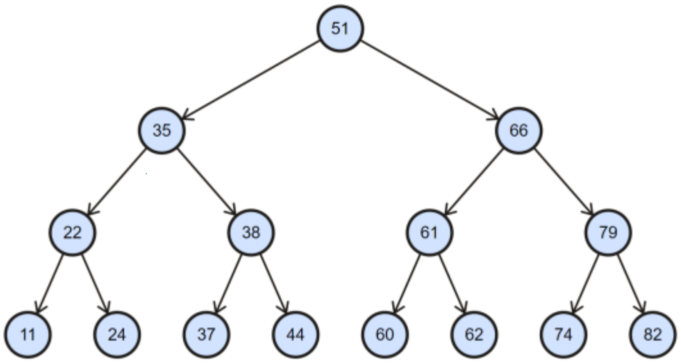
How many nodes are in the above tree?

The height of a tree is the number of edges between the root to the deepest leaf node. h = 3

How can we write h in terms of N?

(6) Searching a BST

Consider the following BST:



Say, we have a search method implemented for our BST class in C++.

We call the method:

`searchBST(44)`

Recall doing a *search* on a list (array or LL):

- *check for equality at each node. If !=, go to the next node*
- *therefore, it takes N operations to do a search on a linear list*

With a BST, we check for equality first. Then we *decide* whether to visit the left child or the right child, based on whether search-key is less than or greater than the current node's key:

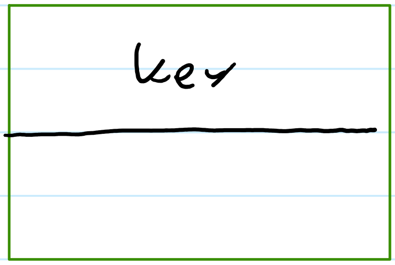
```
if( search->key == crawler->key )
    return crawler

while(key not found)
    if search->key  < crawler->key
        traverse to the left child
    else
        traverse to the right child

return crawler
```

(7) Back to BST: Node and ADT

```
struct Node{
   type key;
   Node * leftChild;
   Node * rightChild;
};
```



note: some implementations also include a parent pointer in the struct definition.

BST ADT:

```
private:
   Node root

   Node insertHelper(Node, value)
   Node searchHelper(Node, value)
   Node deleteHelper(Node, value)

   Node getMinValue(Node)
   node getMaxValue(Node)

   void destroySubtree(Node)

public:
   init()
   insert(value)
   search(value)
   delete(value)
   disp()
   deleteTree // destructor
```

(8) Building a BST

```
if (newKey < currentKey)
   traverse to left child (LC)
else
   traverse to right child (RC)
```

Example: given array A, populate a BST starting with A[0]:

A = {4,2,3,6,0,9,5}

Example:

A = {2,4,3,6,0,9,5}