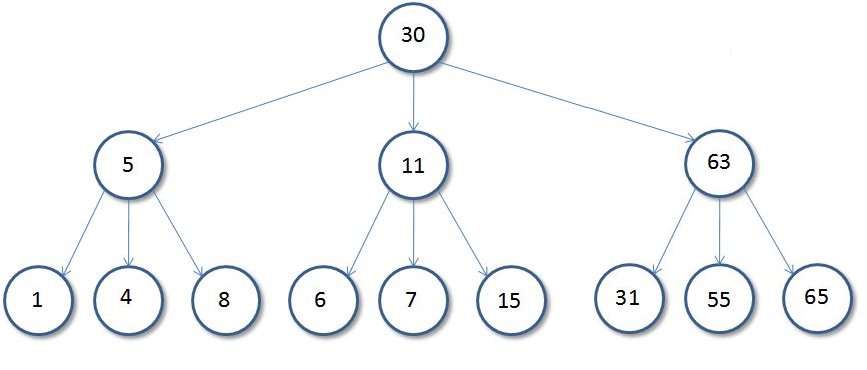
Tree Data structure and it’s Applications

* **Definition:**

A ***Tree*** is a non-linear data structure where each node is connected to a number of nodes with the help of pointers or references

* **Sample tree**



* **Basic Tree Terminologies**:

 Root: The root of a tree is the first node of the tree. In the above image, the root node is the node 30.

 Edge: An edge is a link connecting any two nodes in the tree. For example, in the above image there is an edge between node **11** and **6**.

 Siblings: The children’s nodes of same parent are called siblings. That is, the nodes with same parent are called siblings. In the above tree, nodes *5, 11, and 63* are siblings.

 Leaf **Node**: A node is said to be the leaf node if it has no children. In the above tree, node **15** is one of the leaf nodes.

 Height **of a Tree**: Height of a tree is defined as the total number of levels in the tree or the length of the path from the root node to the node present at the last level. The above tree is of height **2**.

* Note: A tree is called as Binary tree if every node has at most 2 children
* Why to use Tree

There could be several reasons why tree data structure is being used in many ways. One main reason is that you can store data in a more natural hierarchy form. Let’s take our computer’s file system as an example. Every folder has its own name, and we normally store related files or more folders inside it. When you are trying to search for a specific file or folder, you can search from a most related parent folder and start digging deeper from there. But we can also just simply type the file name in our finder, and boom! The computer does all the searching for you from the very top level of your hard drive (the root) and gives you back the result.

root  
 / \   
other home  
 / \  
 downloads projects  
 / / | \  
 ... p1 p2 p3

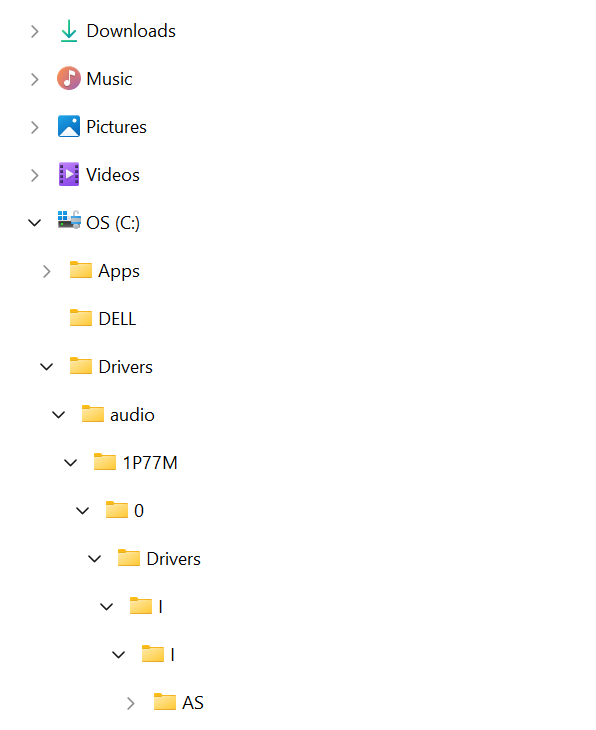
Since all nodes(data) are connected to one another with some sort of relation, they have a relatively shorter searching time (quicker than linked list and slower than arrays). We can also alter data easier and faster (quicker than arrays and slower than unordered linked lists).

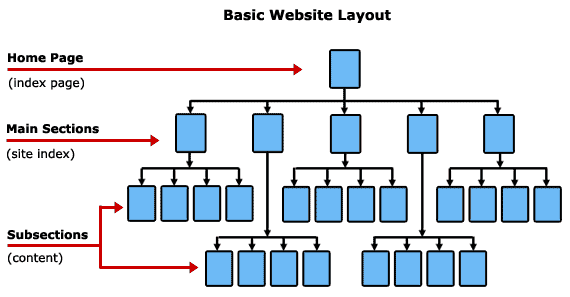
* **Types of tree data structures**

There are different types of tree data structures and they all have their specific usage. In this article, I would like to talk about five different types of tree data structures.

1. General tree
2. Binary tree
3. Binary search tree
4. AVL tree
5. Red-Black tree

* Application of Tree

1. The files and folders that you see in your windows explorer are stored in the tree format. In the below image you can say that ‘My Computer’ is the root; Local Disk (C), Local Disk (D), and Local Disk (E) are basically the parent nodes and files inside them are leaf nodes. This allows faster traversal of the nodes while jumping from one node to another.
2. The layout of a webpage is designed in the tree structure. In the below diagram, the homepage or index page is our root node, main sections/ site index are their child nodes, which again are parents to multiple other child nodes (subsections).



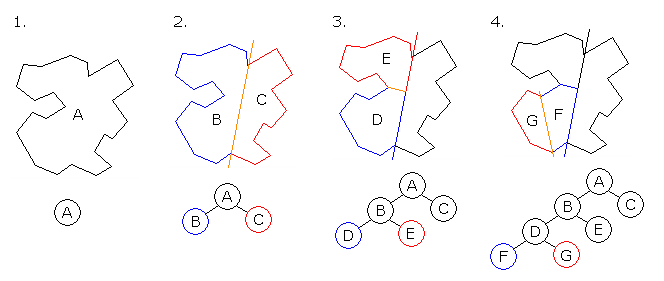
1. Games

Most of us have either played or watched someone play a 3D video game. A typical scenario involves an avatar controlled by the player wandering around in a virtual environment and interacting with its surroundings. Such games usually contain many virtual entities, representing different objects. In order to have some degree of realism, the user’s avatar should at least interact with these entities as if they were solid objects (instead of passing through them).

For this purpose, collision detection is necessary. It can easily be seen that as the number of entities grows larger, naively checking for collisions with every object becomes computationally infeasible. As a matter of fact, most of these checks are redundant, as the objects and the avatar are too far away.

**Space partitioning trees recursively divide the space into smaller and smaller cells until a specific cell size is reached.** Leaf nodes correspond to regions of the virtual environment and contain the objects that are currently within it. Thus, checking for collisions boils down to finding the cell where the avatar currently is, and checking for collisions with objects only in the neighbouring cells.

**This process massively reduces the number of calculations and allows real-time collision detection.** Examples of space partitioning trees include [quadtrees](https://en.wikipedia.org/wiki/Quadtree) (for dividing a 2d space) and [octrees](https://en.wikipedia.org/wiki/Octree#:~:text=An%20octree%20is%20a%20tree,three%2Ddimensional%20analog%20of%20quadtrees.) (for 3d spaces):



1. Data base

**Databases are an integral part of any application.** As time passes, applications become more data-hungry and having a convenient, easily accessible place to store data is very important.

Unfortunately, storage devices are generally slow, often requiring mechanical parts to move in order to access them, like in the case of traditional magnetic hard drives. As such, it is far too inconvenient to do exhaustive searches inside a database, in order to find a specific piece of information. As such, we would like to find the exact memory block where our data entry lies, so we can access it directly.

Fortunately, databases internally contain access structures called indexes, that do just that!

**Database indexes are usually based on a special tree structure, called**[**B+ Trees**](https://www.baeldung.com/cs/b-trees-vs-btrees)**.** Given a query key, we want to access a memory block that contains the data entry that satisfies the query. B+ trees divide the possible key values into intervals, the intervals becoming smaller and smaller as we traverse down the tree.

Leaf nodes contain pointers to the memory blocks that contain the data entries within their interval. Pointers to the previous and next leaves also guarantee easy access between them.

In order to access a data entry that satisfies a query, we traverse the tree and find the leaf node whose interval contains the query key. We then follow the pointer of that leaf node and retrieve the appropriate memory block from the storage device.

This is, of course, an oversimplified version of the process. Database indexes have other needs they must accommodate, adding to the complexity of the problem. However, B+ trees form the foundation, upon which this mechanism is built:

