Recursion is the process in which a function calls itself, either directly or indirectly. When the recursion is direct, a function calls itself directly. The function used for recursion is called a recursive function. When recursion calls a function within itself, it starts creating a call stack. Recursive problems can be solved in the same way. Recursion can be used in mathematical problems like Fibonacci numbers and calculating factorials.

If a recursive function makes multiple nested function calls, it can cause a stack overflow. A recursive function has tail recursion and non-tail recursion. Tail recursion is the last operation that is executed before returning the function. An example of tail recursion would be factorial functions and tree traversals. Non-tail recursion is when the recursive call is not the last operation before returning the function. Instead of factorial functions, non-tail recursion uses the Fibonacci sequence along with recursive algorithms.

Recursion and trees are very similar. Trees have the same structure as recursion. The recursive structure has two main components: the base case and the recursive case. In the base case, the recursion stops. In the recursive case, the function calls itself with arguments to break problems down into smaller problems.

A tree is a hierarchical data structure that consists of data organized in a tree-like format. Each element of data consists of a root node, which starts at the top of the tree and has no parent node. Down the tree's hierarchical structure, the tree has a parent node, and each parent node has child nodes along with siblings connected to the parent node. The root node is linked to the leaf nodes, which do not have a parent. The connection between two nodes in a tree is called an edge.

Even though trees are designed similarly to recursive structures, tree traversal algorithms are written recursively. For example, linked lists, arrays, and stacks can also be traversed using recursion. Some examples of recursive tree traversals are pre-order traversal, in-order traversal, and post-order traversal.

For pre-order traversal, the method first visits the node, and then it traverses from the left subtree to the right subtree. For in-order traversal, the method visits all the nodes in the left subtree, then the root node, and then visits the right subtree. For post-order traversal, the method visits all the nodes in the left subtree, visits all the nodes in the right subtree, and then visits the root node.

Recursion can be used on a binary tree to calculate the height of the tree by calling a function on its subtrees. The height is determined from the root node to a leaf node. For the nodes, the height is the number of edges from the root to the furthest leaf node. The height of a tree with just one node would be 0, while the height of a null tree would be -1. The height of the tree depends on the edge cases and the tree type.

Another way recursion can be used is by calculating the sum of nodes in a tree structure, like a binary tree. The base case for the sum of nodes would be if the node is null or empty, in which case the sum is 0. If the node is not empty or null, the recursion steps for calculating the sum of nodes involve adding the value to the node, calculating the sum of nodes in the left subtree, and then calculating the sum of nodes in the right subtree. After calculating the sum of the subtrees and the current node, the sum is then returned. When calculating the sum of nodes, the method starts from the root.

A binary search tree is a fundamental data structure. It is used for searching, insertion, and deletion, and it also depends on how the nodes are arranged. Nodes in the left subtree have smaller values than the nodes in the right subtree. Recursion is used in a binary search tree to find a specific value.

For the base case, if there is no root node and the tree has no nodes, then the value is not found, and the base case returns false. If the target value equals the root node's value, then the base case will return true. If the target value is greater than the root value, the recursive function will search the right subtree. If the target value is less than the root value, the function will search the left subtree.

With a binary search tree, each node can have up to two children: one being the right child and the other being the left child. The left child's value is less than the parent's value, while the right child's value is greater. For the base case, if the array is empty, the function returns null. The recursive function must find the middle element in the array and create a new node with that middle element as its data.

To build the left and right subsets of a binary tree, the left subtree must contain elements less than the middle element, and the right subtree must contain elements greater than the middle element. After the subtrees are recursively created, the function must return the root node. The root node ends up being the middle element, which helps balance the tree structure.

Even though recursion is essential in programming, it has its limitations. For example, whenever a recursive call adds a new stack frame to the call stack, memory is consumed. Deep recursion can lead to stack overflow errors when there are large inputs or deeply nested recursive calls. Recursive solutions can incur overhead due to function calls and returns. Debugging recursive functions can be difficult due to the execution flow through multiple function calls.

Compiling tail-recursive functions into iterative code can reduce stack usage and improve performance. In some cases, iterative solutions to recursive problems can be more time efficient.