Space Complexity

The space complexity of an algorithm measures the amount of memory that the algorithm needs to execute correctly. An algorithm will require memory to store any data structures that it uses and, if the algorithm is recursive, it will also require memory to store the execution stack. To illustrate how to compute the space complexity of an algorithm, let us consider the following algorithm that computes the height of a tree.

Algorithm height(r)
In: Root r of a tree
Out: Height of the tree

```
if r is a leaf then return 0
else {
   max_height ← 0
   for each child r_i of r do {
      h ← height (r_i)                    (A_1)
      if h >max_height then max_height ← h
   }
   return max_height + 1
}
```

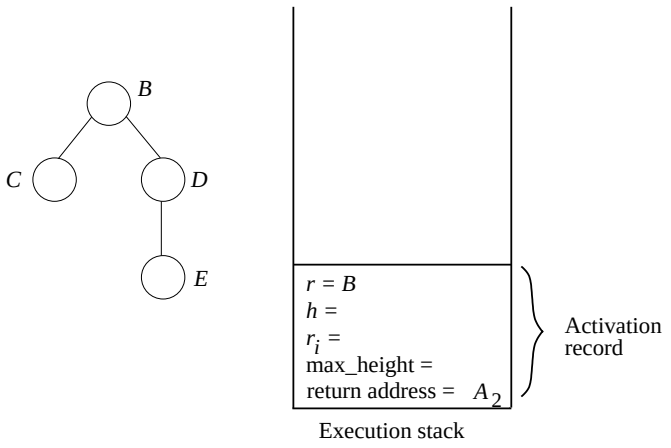
The above algorithm needs memory to store the input tree. If we assume that each node requires a constant amount c of space, then if the tree has n nodes the amount of space needed to store it is cn, which is $O(n)$.

Since the algorithm is recursive, part of the memory of the computer has to be used to store the *execution stack* (also called *call stack*, *control stack*, *run-time stack*, *machine stack*, or just "the stack"). Every time that a recursive call is made, an *activation record* (also called an *activation frame*, a *stack frame* or just a *frame*) is created at the top of the execution stack. An activation record stores the values of the parameters of the algorithm, the values of the local variables, and the return address (the address of the instruction that needs to be executed once the current call is finished).

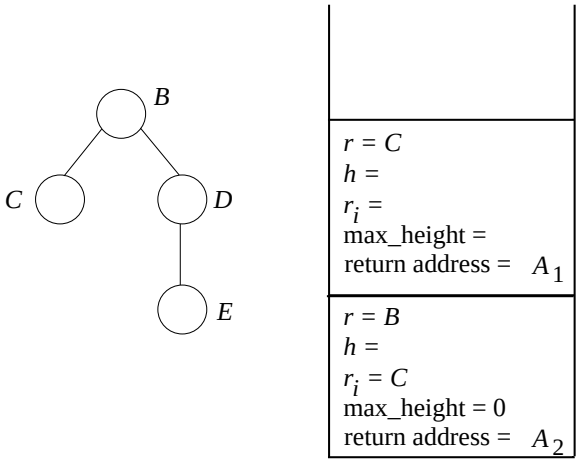
If, for example the above algorithm is invoked for the first time from the following algorithm:

```
Algorithm main {
   B ← createTree()
   h ← height(B)                    (A_2)
}
```

Then, when the initial call to height is made from algorithm main, an activation record is created in the execution stack. This activation record will store the values of the parameter r and the local variables max_height, r_i, h, and the return address. When the first call is made the value of r stored in the activation record is set to B, the root of the tree created by algorithm createTree. The return address A_2 is also stored in the activation record. At this point the execution stack looks like this:



Algorithm height then starts its execution. Since r is not a leaf, the value of max_height is set to zero and then r_i is set to the first child of r, namely node C; the recursive call is then made with C as the value of the parameter. A new activation record is created at the top of the stack; this becomes the active activation record. At this point the execution stack looks like this:

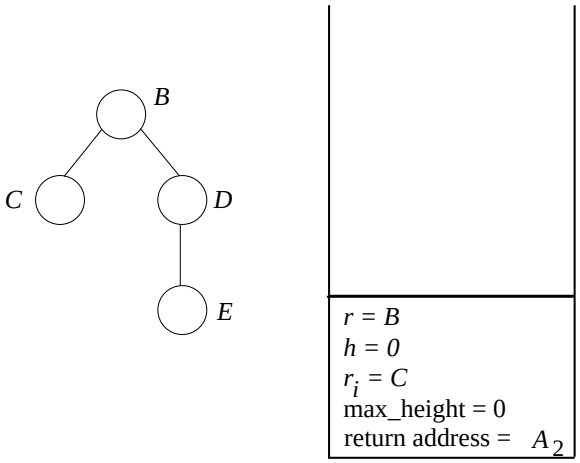


Execution stack

Note that the return address stored in the second activation record is A_1 because the call was made from algorithm height. Since node r is now a leaf, the algorithm simply returns the value 0. When a recursive call ends the activation record at the top of the execution stack is removed, but first the return address is retrieved. Since the return address stored in the last activation record is A_1, then statement

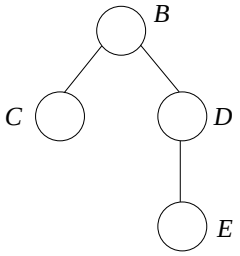
$h \leftarrow$ height (r_i)

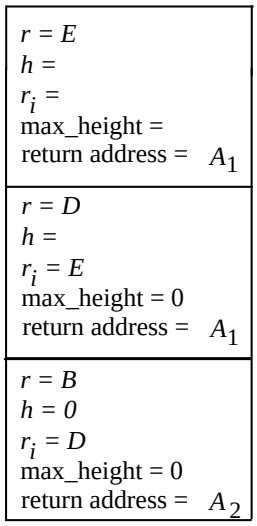
is executed next. The last recursive call returned the value 0, which is then stored in h. At this point the execution stack is this:



Execution stack

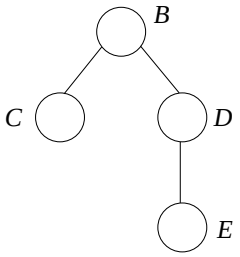
Next the algorithm sets r_i to the second child of B, namely D and then a recursive call is made. A new activation record is created at the top of the stack. Since D is an internal node, the value of max_height is set to zero and r_i is set to E in the activation record at the top of the stack. Then a new recursive call is made. The execution stack at this point looks like this:

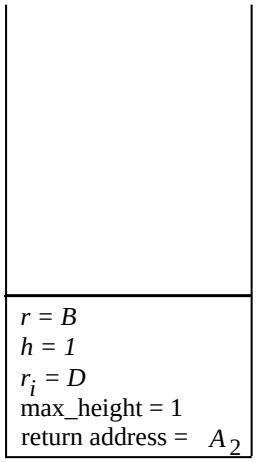




Execution stack

Since E is a leaf the algorithm returns the value 0 and the activation record at the top of the execution stack is removed. The value of h is set to 0 in the new current activation record. Since node D has only one child, the **for** loop in the second recursive call ends and the algorithm returns the value 1. The activation record at the top of the execution stack is removed and since the return address from the last call was A_1, the value 1 is stored in h and in max_height as shown below:





Execution stack

Since B does not have more children, the **for** loop ends and the algorithm finally returns the value 2. The last activation record is removed and then the execution continues in algorithm main.

Note that every time that a recursive call is made a new activation record is added to the execution stack and every time that a recursive call ends the activation record at the top of the execution stack is removed. Hence, for algorithm height, the maximum number of activation records that are simultaneously stored in the execution stack is equal to one plus the height of the input tree. Since every activation record uses a constant amount c' of space, the total space needed by the execution stack is c'(height of tree + 1) is O(heigt of tree).