Data Structures [R1UC308B]

Module-III: Recursion

Dr. Subhash Chandra Gupta

GALGOTIAS UNIVERSITY

School of Computer Science and Engineering
Plat No 2, Sector 17A, Yamuna Expressway
Greater Noida, Uttar Pradesh - 203201

July 31, 2025

Contents

Recursion

The process in which a function calls itself directly or indirectly is called **recursion** and the corresponding function is called a **recursive function**. Using recursive algorithm, certain problems can be solved quite easily.

Need of Recursion:

- Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.
- It has certain advantages over the iteration technique which will be discussed later.
- A task that can be defined with its similar subtask, recursion is one of the best solutions for it. For example; The Factorial of a number.

Properties of Recursion:

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- Base condition is needed to stop the recursion otherwise infinite loop will occur.

Algorithmic Steps:

The algorithmic steps for implementing recursion in a function are as follows:

1 - Define a base case: Identify the simplest case for which the solution is known or trivial. This is the stopping condition for the recursion, as it prevents the function from infinitely calling itself.

2 - Define a recursive case: Define the problem in terms of smaller subproblems. Break the problem down into smaller versions of itself, and call the function recursively to solve each subproblem.

3 - Ensure the recursion terminates: Make sure that the recursive function eventually reaches the base case, and does not enter an infinite loop.

4 - Combine the solutions: Combine the solutions of the subproblems to solve the original problem.

Types of Recursions:

- Recursion are mainly of two types depending on whether a function calls itself from within itself or more than one function call one another mutually.
- The first one is called **direct recursion** and another one is called **indirect recursion**.

Direct Recursion

When a function calls itself from within itself is called direct recursion. These can be further categorized into four types:

- **Tail Recursion:** If a recursive function calling itself and that recursive call is the last statement in the function then it's known as Tail Recursion.
  - After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

```java
// Recursion function
static void fun(int n)
{
  if (n > 0)
  {
    System.out.print(n + " ");

    // Last statement in the function
    fun(n - 1);
  }
}
```

- **Head Recursion:** If a recursive function calling itself and that recursive call is the first statement in the function then it's known as Head Recursion.
  - There's no statement, no operation before the call.
  - The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

```java
// Recursive function
static void fun(int n)
{
    if (n > 0) {

        // First statement in the function
        fun(n - 1);

        System.out.print(" "+ n);
    }
}
```

- **Tree Recursion:** If a recursive function calling itself for one time then it's known as Linear Recursion. Otherwise if a recursive function calling itself for more than one time then it's known as Tree Recursion.

```java
// Recursive function
static void fun(int n)
{
    if (n > 0) {
        System.out.print(" "+ n);

        // Calling once
        fun(n - 1);

        // Calling twice
        fun(n - 1);
    }
}
```

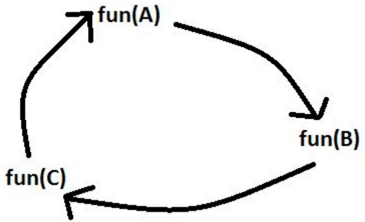
- **Nested Recursion:** In this recursion, a recursive function will pass the parameter as a recursive call. That means recursion inside recursion.

```
static int fun(int n)
{
    if (n > 100)
        return n - 10;

    // A recursive function passing parameter
    // as a recursive call or recursion
    // inside the recursion
    return fun(fun(n + 11));
}
```

Indirect Recursion

In this recursion, there may be more than one functions and they are calling one another in a circular manner.



In the above diagram fun(A) is calling for fun(B), fun(B) is calling for fun(C) and fun(C) is calling for fun(A) and thus it makes a cycle.

Removal of recursion

By replacing the selection structure with a loop, recursion can be eliminated. A data structure is required in addition to the loop if some data needs to be kept for processing beyond the end of the recursive step. A simple string, an array, or a stack are examples of data structures.

There are a few ways to remove recursion from code, including:

- Iteration: Wrap your algorithm in a loop, pushing and popping a custom call stack at the start and end of each iteration.
- Macro expansion: This technique can eliminate recursion, but the depth of recursion is limited by the number of macro invocations.

- Refactoring: In Python, you can refactor the code using a series of small, careful refactorings to remove a single recursion.
- Stack: You can use a stack to store a representation of the operations that need to be performed.
- Generalization: Generalize the function definition.
- Computation traces: Study the computation traces of the function.

Iteration and recursion with examples

- Linear Search

```java
public static int LSIterative(int []arr, int n,int item) {
    for(int i=0;i<n;i++) {
        if(arr[i]==item) {
            System.out.println("Item "+item+" is at index "+i+"
            System.out.println("Have a good day, bye bye!");
            return i;
        }
    }
    System.out.println("Item "+item+" is not in the array");
    System.out.println("Have a good day, bye bye!");
    return -1;
}
```

Figure: Linear Search Iterative

```java
public static int LSRecursive(int []arr, int n,int item) {
    if(n<=0) {
        System.out.println("Item "+item+" is not found in the array")
        System.out.println("Have a good day, bye bye!");
        return -1;
    }
    else if (arr[n-1]==item) {
        System.out.printf("Item %d is at index %d in the array\n",ite
        System.out.println("Have a good day, bye bye!");
        return n-1;
    }
    else return LSRecursive(arr,n-1,item);
}
```

Figure: Linear Search Recursive

- Fibonacci Numbers

```
//Using iteration Method
static void Fibo(int N)
{
    int num1 = 0, num2 = 1;
    for (int i = 0; i < N; i++) {
        System.out.print(num1 + " ");
        int num3 = num2 + num1;
        num1 = num2;
        num2 = num3;
    }
    System.out.println("\nEnd of iteration");
}
```

Figure: Fibonacci Numbers Iterative

```
//Using recursion Method
static int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

Figure: Fibonacci Numbers Recursive

```java
//iteration
System.out.println("Fibonacci Series of " + n +
        " numbers is: ");
System.out.println("\nStart of iteration");
Fibo(n);

//Recursion
System.out.println("\nStart of recursion");
for (int i = 0; i < n; i++) {
     System.out.print(fib(i) + " ");
}
System.out.println("\nEnd of recursion");
```

Figure: Fibonacci Function Call

- Tower of Hanoi

```
static void towerOfHanoi(int n, char from_rod,
                         char to_rod, char aux_rod)
{
    if (n == 0) {
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    System.out.println("Move disk " + n + " from rod "
                        + from_rod + " to rod "
                        + to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

Figure: Tower of Hanoi

Trade-off between iteration and recursion

The trade-offs between iteration and recursion in programming include:

- Speed: Iteration is generally faster than recursion.
- Memory: Recursion requires more memory than iteration.
- Code complexity: Recursion can lead to simpler, more readable code, while iteration can result in more complex code.
- Time complexity: Recursion has higher time complexity than iteration.
- Approach: Recursion follows a divide and conquer approach, while iteration follows a sequential execution approach.
- Suitability: Recursion is better for tasks that can be described naturally in a recursive way, while iteration is better for loops.
- Optimization: It can be difficult to optimize recursive code.

Thank you

Please send your feedback or any queries to
subhash.chandra@galgotiasuniversity.edu.in