National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Fall 2022, Lab Manual – 03

Course Code: CL-2001	**Course : Data Structures - Lab**
Instructor(s) :	**Abeer Gauher, Sobia Iftikhar**

LAB - 3

RECURSION AND BACKTRACKING

Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Any method that implements Recursion has two basic parts:

- Method call which can call itself i.e. recursive
- A precondition that will stop the recursion.

Note that a precondition is necessary for any recursive method as, if we do not break the recursion then it will keep on running infinitely and result in a stack overflow.

The general syntax of recursion is as follows:

```
methodName (T parameters…)
{
   if (precondition == true)

//precondition or base condition
   {
      return result;
   }
   return methodName (T parameters…);

      //recursive call
}
```

Note that the precondition is also called base condition.

Problem-Solving Using Recursion

The basic idea behind using recursion is to express the bigger problem in terms of smaller problems. Also, we need to add one or more base conditions so that we can come out of recursion.

Example:

```java
public class lab2 {
    public static void main(String[] args) {
        int result = sum( k: 10);
        System.out.println(result);
    }
    public static int sum(int k) {
        if (k > 0) {
            return k + sum( k: k - 1);
        } else {
            return 0;
        }
    }
}
```

```
55
```

Tail and Non – Tail Recursion

Tail Recursion

We refer to a recursive function as tail-recursion when the recursive call is the last thing that function executes.With tail recursion, the recursive call is the last thing the method does, so there is nothing left to execute within the current function.
The tail recursive functions are considered better than non tail recursive functions as tail-recursion can be optimized by the compiler.
Example:

```
public class lab2 {
    public static void main(String[] args) {
        print(3);
    }
    static void print(int n)
    {
        if (n < 0)
            return;
        System.out.print(" " + n);
        print(n - 1);
    }
}
```

```
3 2 1 0
```

Non - Tail Recursion

A function is called the non-tail or head recursive if a function makes a recursive call itself, the recursive call will be the first statement in the function. It means there should be no statement or operation called before the recursive calls. Furthermore, the head recursive does not perform any operation at the time of recursive calling. Instead, all operations are done at the return time.

Example:

```
public class lab2 {
    public static void main(String[] args) {
        System.out.println(fact( n: 5));
    }
    static int fact(int n)
    {
        if (n == 0)
            return 1;

        return n * fact( n: n - 1);
    }
}
```

```
120
```

Indirect Recursion

If the function f1 calls another function f2 and f2 calls f1 then it is indirect recursion (or mutual recursion).

This is a two-step recursive call: the function calls another function to make a recursive call.

Syntax:

```
void indirectRecursionFunctionf1()
{
  // some code...

  indirectRecursionFunctionf2();

  // some code...
}

void indirectRecursionFunctionf2()
{
  // some code...

  indirectRecursionFunctionf1();

  // some code...
}
```

Example:

```java
public class lab2 {
    public static void main(String[] args) {
        int x = 12;
        func1(x);
    }

    static void func1(int a) {
        if (a > 0) {
            System.out.println(a);
            func2( b: a - 1);
        }
    }

    static void func2(int b) {
        if (b > 0) {
            System.out.println(b);
            func1( a: b - 3);
        }
    }
}
```

```
12
11
8
7
4
3
```

Backtracking

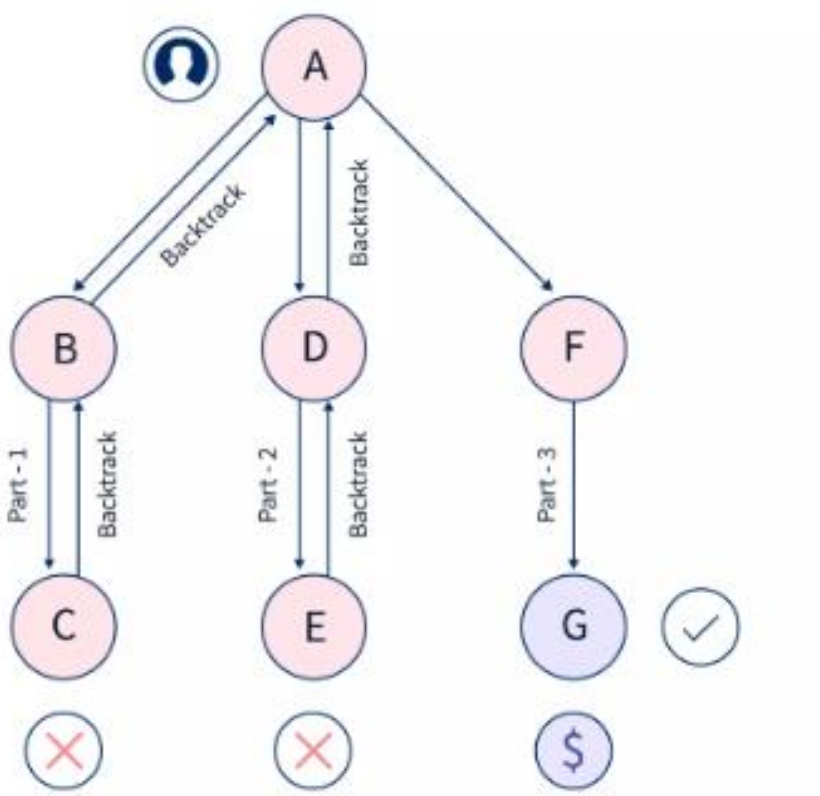
A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output.The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

Backtracking Algorithm

Backtrack(x)
  if x is not a solution
    return false
  if x is a new solution
    add to list of solutions
  backtrack(expand x)

To understand this clearly, consider the given example. Suppose you are standing in front of three roads, one of which is having a bag of gold at it's end, but you don't know which one it is.

- Firstly you will go in Path 1 , if that is not the one, then come out of it, and go into Path 2 , and again if that is not the one, come out of it and go into Path 3 .
- So, let's say we are standing at 'A' and we divided our problem into three smaller sub-probelms 'B', 'D' and 'F'.
- And using this sub-problem, we have three possible path to get to our solution -- 'C', 'E', & 'G'. So, let us see how can we solve the problem using backtracking.



1. **Choose the 1st path: A--> B--> C => Not found our feasible solution => C--> B--> A (backtrack and move back to A again)**
2. **After backtracking & Choosing the 2nd path: A--> D--> E => Not found our feasible solution => E--> D--> A (backtrack and move back to A again)**
3. **Backtrack & Choose the 3rd path: A--> F--> G => Got our solution**

In backtracking we try to solve a subproblem, and if we don't reach the desired solution, then undo whatever we did for solving that subproblem, and try solving another subproblem, till we find the best solution for that problem(if any).

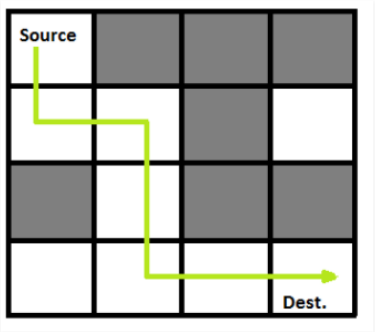
Difference between Recursion and Backtracking
There is no major difference between these two in terms of their approach, except for what they do:

- Backtracking uses recursion to solve it's problems. It does so by exploring all the possiblities of any problem, unless it finds the best and feasible solution to it.
- Recursion occurs when a function calls itself repeatedly to split a problem into smaller sub-problems, until it reaches the base case.
- So, we may say recursion is a part of backtracking itself. However, there is no base case in backtracking to stop it. The backtracking mechanism will not stop as long as it finds an optimal solution to the problem.

Lab Tasks:

Task#1:
Sort The Unsorted Numbers with both tail recursive and Normal recursive approach
Sample Input and Output
Given array is
12 11 13 5 6 7 Sorted array is 5 6 7 11 12 13

Task#2:
A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. You start from source and have to reach the destination. You can move only in two directions: forward and down.



In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

A. Design the function with recursive approach to find the number of existing destination path in the above.

Task#3:
You are required to write a program that uses indirect/mutual recursion to print the first fifteen numbers. Create two functions that invoke each other using indirect recursion.

Task#4:
a. Generate the following sequence with recursive approach

1 , 3 , 6 , 10 , 15 , 21 , 28 . . . .

b. Write a indirect recursive code for the above task with same approach as defined in the above sample code of In-Direct Recursion

Task#5:
You are given a string you need to print all possible strings that can be made by placing spaces (zero or one) in between them through backtracking.
Input: str[] = "FAST"
Output: FAST
FAS T
F AST
F A S T
FA ST
FAS T