CC-213L

Data Structures and Algorithms

Laboratory 02

Recursion and Backtracking

Version: 1.0.0

Release Date: 27-10-2024

Department of Information Technology
University of the Punjab
Lahore, Pakistan

Contents:

Learning Objectives:
- Understand what's recursion
- Trace recursive functions
- Write recursive relations
- Solve complex recursion problems
- Understand what is backtracking
- Practice backtracking
- Solve problems including backtracking

Resources Required:
- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:
- In this Lab, you are **NOT** allowed to discuss your solution with your classmates, even not allowed to ask how s/he is doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course / Lab Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Muhammad Tahir Mustafvi	bcsf20m018@pucit.edu.pk

Background and Overview:

Recursion:

Recursion is a powerful programming technique where a function calls itself to solve a problem by breaking it into smaller, more manageable sub-problems.

Every recursive function must have at least two cases: the recursive case and the base case. The base case is a small problem that we know how to solve and is the case that causes the recursion to end.

Base Case:

At this point, the problem is small enough that we can answer it without breaking it down into the smaller sub-problems. For example, you know what the answer of 1! (Factorial) is. The base case in recursion is like the stop sign for a recursive function. It tells the function when to stop calling itself and start giving answers.

Recursive Case:

The recursive case is where a function calls itself with a smaller or simpler version of the problem. It's the heart of recursion, breaking down complex tasks into smaller steps. This process continues until the base case is reached, at which point the recursion reverses and combines the results to solve the original problem. It's like solving a big puzzle by solving smaller pieces first.

Brief description of types of recursions:

Direct Recursion: A function calls itself directly.
Indirect Recursion: Functions call each other in a circular manner.
Linear Recursion: A function makes only one recursive call.
Tree Recursion: A function makes multiple recursive calls.
Tail Recursion: Recursive call is the last operation in the function.
Non-Tail Recursion: Recursive call isn't the last operation.
Nested Recursion: A function calls itself with an altered argument.

Steps to Keep in Mind When Solving a Recursion Problem:

Identify the Base Case
Determine the simplest case(s) where the problem can be solved directly.

Define Recursive Formulas
Identify and express the problem in terms of smaller, similar subproblems. Formulate the recursive formulas to solve these subproblems.

Combine Subproblems' Results
Combine or aggregate the results obtained from solving the subproblems to obtain the final solution.

And keep in mind - Understanding recursion requires a lot of practice :).

Recursive Trace

Before we move towards some logical recursive functions, let's practice recursive trace a little bit with the following snippets. Get your pen and paper and try to figure out what would be the output of the following functions with the help of recursion tree. Don't do it on compiler at first but later for verification.

1. Assume the value of N to be 4.

```
3   void f1(int N)  // dual recursion
4   {
5       if (N <= 1)
6       {
7           return;
8       }
9       cout << N;
10      f1(N - 1);
11      f1(N - 1);
12  }
```

Figure 1 (Recursion Example 1)

Explanation:

In the above recursive function, every function will call two times itself in its body. In line 9 the current value of N will be displayed. In line 10 function calls itself by passing argument 1 less than its parameter. Same for line number 11.

2. Assume the value of N to be 4.

```
3   void f2(int N)  // dual recursion
4   {
5       if (N <= 1)
6       {
7           return;
8       }
9
10      f2(N - 1);
11      cout << N;
12      f2(N - 1);
13  }
```

Figure 2 (Recursion Example 2)

Explanation:

In line number 11 the value of parameter of recursive function will be displayed. In line 10 and 12 function calls itself recursively for value of N less than its parameter value.

3. Assume the value of N to be 5.

```
3   void f3(int N)  // dual recursion
4   {
5       if (N <= 1)
6       {
7           return;
8       }
9
10      f3(N - 1);
11      f3(N - 1);
12      cout << N;
13  }
```

Figure 3 (Recursion Example 3)

Some Logical Recursive Functions Examples

Go through the following algorithms and you will get some idea of how to write the recursive function.

1. Print numbers up to **n** in ascending order - non-tail recursion

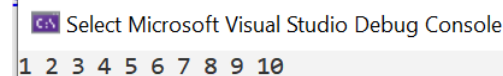```
1   #include <iostream>
2   using namespace std;
3   void printNumbers(int n)
4   {
5   //Base case :Stop the recursion when n reaches 0
6   if (n == 0)
7   return;
8   // Recursive call to print numbers from 1 to n-1
9   printNumbers(n - 1);
10
11   // Print the current number
12
13   cout << n << " ";
14   }
15   int main()
16   {
17   printNumbers(10);
18   }
```

Figure 4 (Recursion Example 4)

**Explanation:**

In line 6 there is a base case. When number reaches zero function should return to its calling statement and control will move to the next statement. Otherwise in line 9 we have called recursive function again for value of N-1 and so on for next recursive call. In line 13 value of n is also displayed that will be passed to that function at calling time as argument.

**Output:**

Select Microsoft Visual Studio Debug Console

1 2 3 4 5 6 7 8 9 10

Figure 5(Output)

2. Print numbers in descending order:

```
1    #include <iostream>
2    using namespace std;
3    void printNumbersDescending(int n)
4    {
5        //Base case :Stop the recursion when n reaches 0
6        if (n == 0)
7            return;
8
9        // Print the current number
10
11       cout << n << " ";
12
13       // Recursive call to print numbers from n to 1
14       printNumbersDescending(n - 1);
15   }
16   int main()
17   {
18       printNumbersDescending(10);
19   }
```

Figure 6 (Descending Print)

Explanation:

In line 6 there is a base case. When a number reaches zero, the function should return to its calling statement and control will move to the next statement. In line 11 the value of n is also displayed that will be passed to that function at calling time as argument. In line 14 we have called recursive function again for value of N-1 and so on for next recursive call.

Output


```
10 9 8 7 6 5 4 3 2 1
```

Figure 7 (Output)

3. Factorial of a number.

```cpp
1    #include <iostream>
2    using namespace std;
3    int factorial(int n)
4    {
5        if (n == 0 || n == 1)    // base case
6            return 1;  // we know the answers for the base cases
7        int sub_problem = factorial(n - 1); // recursive case
8        int answer = n * sub_problem;   // combine results
9        return answer;
10   }
```

Figure 8 (Factorial)

Explanation:

This is an example of linear, on-tail recursion. If we call this function for n = 4
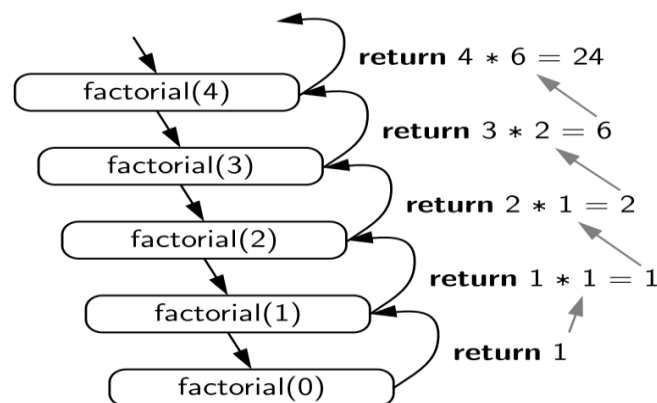

Figure 9 (Factorial Recursion Trace)

4. Nth Fibonacci number:

```cpp
1    #include <iostream>
2    using namespace std;
3    int fib(int n)
4    {
5        if (n <= 1)
6            return n;
7        return fib(n - 1) + fib(n - 2);
8    }
```

Figure 10 (Fibonacci Series)

Explanation:

You already know that Fibonacci series is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …, so to print any number at a specific index, say n = 4, i.e. 4th index.
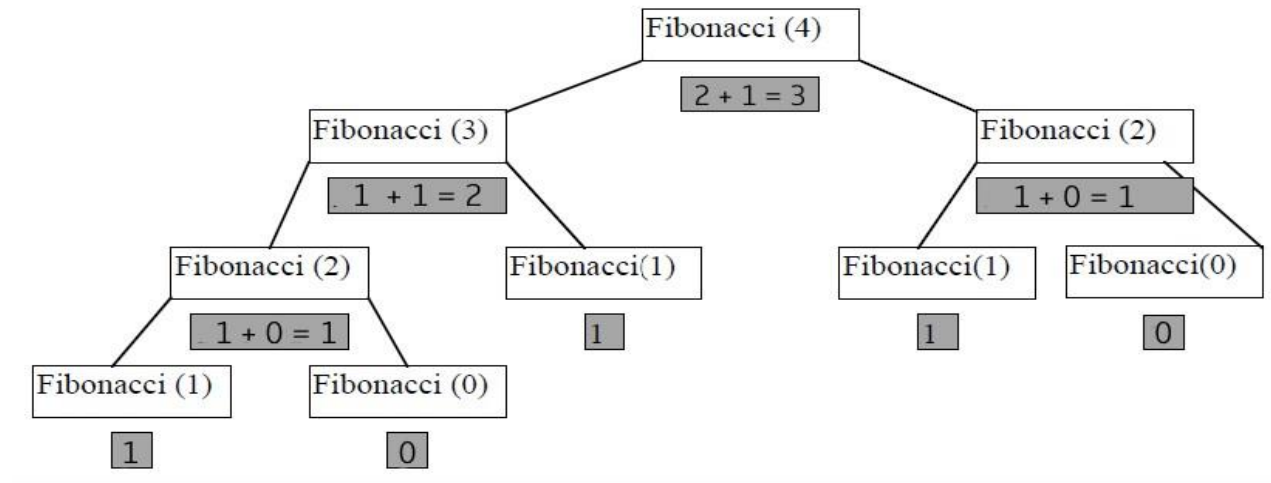
Figure 11(Fibonacci Series Trace)

5. Function to find the minimum element in an array:

```cpp
int findMinRec(int* A, int size)
{
    // Base case: if there is one element in, it`s the minimum
    if (size == 1)
        return A[0];
    int this_element = A[size - 1];
    int sub_problem = findMinRec(A, size - 1); //Recursive case
    int answer = min(this_element, sub_problem); // combine result;
    return answer;

}
```

Figure 12 (Find Min Number)

Activities:

Pre-Lab Activities

Now that you have gone through some recursive functions, put your understanding on test and try to solve the following problems.

Task 01 Search Array:

Write a recursive function to return the index of a specific element in array. Function should return -1 if element is not present in array.
int search (int* arr, int element, int array_size);

Task 02 Power Find:

Write the recursive function to calculate the power of a number.
int pow (int number, int power);

Task 03 Num Digits:

Write the recursive function to count the number of digits in a number.
int number_of_digits(int number);

Task 04 String Reverse:

Given a string s, the function should return the reverse of the string.
string reverse (string s, int len);

Task 05: Binary Search

Given a sorted array of size n and target element. Perform binary search using recursion and return the index of element.
int binary_search (int* arr, int left, int right , int target);

Task 06: Binary Search with a twist

Given a **sorted** array of size n and a target element, implement binary search without using explicit left and right pointers. Instead, trim the array at each step, adjusting the pointer and size to narrow down the search range.
int binary_search (int* arr, int size, int target);

 arr: Pointer to the current subarray.
 size: Size of the current subarray.
 target: Element to search for.

In-Lab Activities:

Task 01 Dry run:

Dry run the following recursive function for binomial (**8**,**4**) and find the output. Also, show the recursive trace.

```cpp
int binomial(int n, int m)
{
        if (n == m || m == 0)
                return 1;
        else
                return binomial(n − 1, m) + binomial(n − 1, m − 1);
}
```

Task 02 Sum Squares:

Write a recursive function to calculate the sum of squares of natural numbers up to a given number n.
int squares_sum(int n);
Example:
Input: 3
Output: 14
Explanation: $1^2 + 2^2 + 3^2 = 14$

Task 03 Find GCD:

Write recursive function to find the GCD of two numbers. The greatest common divisor (GCD) of two or more numbers is the greatest common factor number that divides them, exactly. It is also called the highest common factor (HCF). For example, the greatest common factor of 15 and 10 is 5, since both the numbers can be divided by 5. 15/5 = 3. 10/5 = 2.
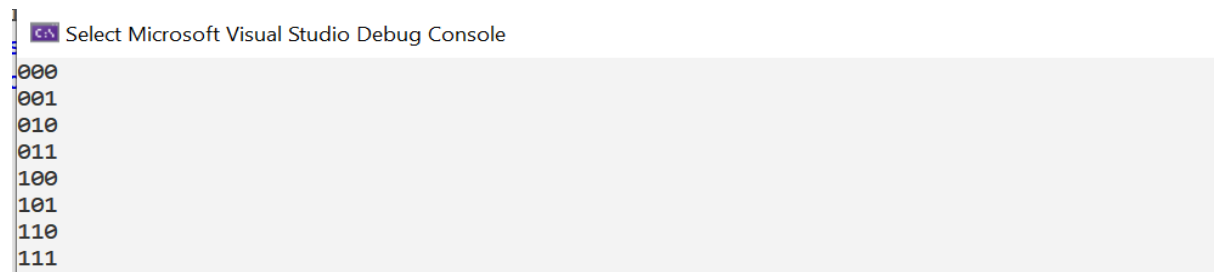
Task 04 Print Bitsrings:

Write a function to print all the possible bitstrings of length n. Function call will be made from the main function like this:

```cpp
1    #include <iostream>
2    using namespace std;
3    void func(string s, int len)
4    {
5
6    }
7    int main()
8    {
9        func("", 3);
10       return 0;
11   }
```

Figure 13 (Print Bit string)

Output:

```
Select Microsoft Visual Studio Debug Console
000
001
010
011
100
101
110
111
```

Figure 14 (Output)

Task 05: Rectangle Print

Write a recursive function that should print pattern like this. This function should take number of rows and argument and print this rectangle.

```
Enter the number of rows: 5
*****
****
***
**
*
```

Figure 15 (Pattern Print)

Post-Lab Activities

Task 01 Rat in a Maze:

Problem Description:

In a distant land, there is a maze with a complex network of paths. Inside this maze lives a curious little rat who is determined to explore its depths and reach a destination at the opposite end. However, the maze is full of obstacles—certain paths are blocked, making it impossible for the rat to pass through them. The rat can only move in two directions: to the right or downward. It must navigate the maze by finding a path from its starting point in the top-left corner to its destination at the bottom-right corner.
The challenge is to help the rat find one unique path through the maze, if it exists. If the path does not exist, it is important to indicate that as well. Your task is to determine the path and mark it on the maze.

Problem Statement:

You are given a 2D grid (maze) of size n x n where each cell can either be open (1) or blocked (0). The rat starts at the top-left corner of the grid, i.e., position (0, 0), and its goal is to reach the bottom-right corner of the grid at position (n-1, n-1). The rat can only move right or down at any step.

Function:

void rateInMaze(int maze, int x, int y, int** sol) where:**
1. **maze**: A 2D integer array of size n x n representing the maze. Cells marked with 1 indicate open paths, and cells marked with 0 represent blocked paths.
2. **x, y**: Integers representing the coordinates of the destination cell in the maze.
3. **sol**: A 2D integer array of size n x n initialized to zero. The function should update the sol array to indicate the unique path found by the rat, where sol[i][j] is set to 1 for cells that are part of the path and 0 otherwise.

Constraints:
- The rat can only move right or down from any cell.
- If a path exists, the function should return the solution array with the path marked.
- If no path exists, the function should leave the solution array unchanged (all zeros).

Submissions:
- For In-Lab Activity:
    - Save the files on your PC.
    - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
    - Submit the .c file on Google Classroom and name BITF21MXXX_Lab05-Pre_Lab.zip and BITF21MXXX_Lab05-Post_Lab.zip Pre-Lab and Post-Lab. Tasks should be named like Task01 and Task02 etc.

Evaluations Metric:
- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:**   **[25 marks]**
    - Task 01: Search Array   [05 marks]
    - Task 02: Power Find   [05 marks]
    - Task 03: Num Digits   [05 marks]
    - Task 04: String Reverse   [05 marks]
    - Task 05: Binary Search   [05 marks]
- **Division of In-Lab marks:**   **[35 marks]**
    - Task 01: Dry Run   [05 marks]
    - Task 02: Sum Square   [05 marks]
    - Task 03: Find GCD   [05 marks]
    - Task 04: Bit String print   [10 marks]
    - Task 05: Pattern Print   [10 marks]
- **Division of Post-Lab marks:**   **[15 marks]**
    - Task 01: Tower of Hanoi   [15 marks]

References and Additional Material:
- Recursion
  https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/

Lab Time Activity Simulation Log:
- Slot – 01 – 02:15 – 02:30:   Class Settlement
- Slot – 02 – 02:30 – 03:00:   In-Lab Task 01
- Slot – 03 – 03:00 – 03:30:   In-Lab Task 02
- Slot – 04 – 03:30 – 04:00:   In-Lab Task 03
- Slot – 05 – 04:00 – 04:30:   In-Lab Task 04
- Slot – 06 – 04:30 – 05:00:   In-Lab Task 05
- Slot – 07 – 04:45 – 05:00:   Discussion on Post-Lab Task