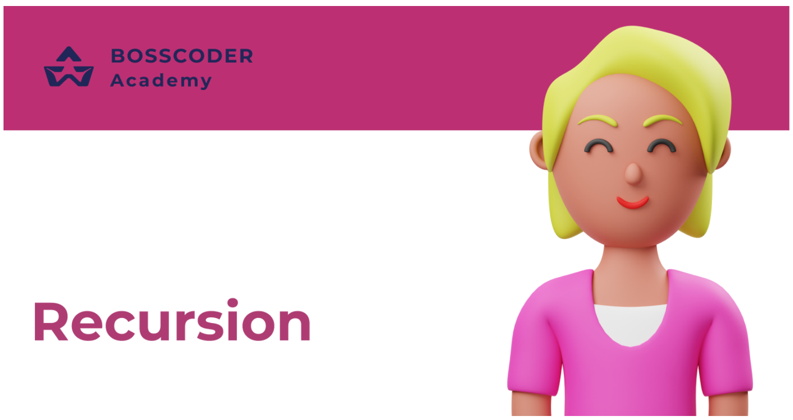
BOSSCODER ACADEMY

UPSKILL WITH US

Recursion

Gyanu Mayank

17th February, 2023



Agenda.
1. Basics of Recursion and How it works
2. Problems Based on Recursion

Recursion
Recursion is when a function calls itself again and again until it reaches a specified stopping condition.
Each recursive function has two parts:
1. **Base Case:** The base case is where the call to the function stops i.e., it does not make any subsequent recursive calls
2. **Recursive Case:** The recursive case is where the function calls itself again and again until it reaches the base case.

To solve a problem using recursion, break the recursion. i.e. make a **recurrence relation** for that problem. When a function is called, the state of that function is saved in a stack. Each recursive call pushes a new stack frame. When the base case is reached the stack frames start popping from the stack until the stack becomes empty.

Example

1. [Multiplication of two numbers using Recursion]

Given two numbers x and y find the product using recursion.

Example

Input: x=5, y=6

Output: 30

Approach

The multiplication of a number is nothing but repeated addition. So, the approach could be to recursively add the bigger of the two numbers (M and N) to itself until we obtain the required product. Let’s assume that M >= N. Then according to this approach, we recursively add ‘M’ to itself, ‘N’ times.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int mult(int x, int y)

{
    if (x < y)
        return mult(y, x);
```

```
    else
        return 0;
}

int main()

{
    int m, n;

    cin >> m >> n;

    cout << mult(m, n);

    return 0;
}
```

Time Complexity**:** O(min(x,y))
Space Complexity: O(min(x,y)), The extra space is used in the recursive call stack.

2. **Modular exponentiation**
Given three numbers a, b and c, we need to find (a^b) % c
Example
Input: a=2,b=4,c=10
Output: 6
Approach
The approach is based on the below properties.
1. (m * n) % p has a very interesting property:
(m * n) % p =((m % p) * (n % p)) % p
2. if b is even:
(a ^ b) % c = ((a ^ b/2) * (a ^ b/2))%c ? this

(a ^ b) % c = (a * (a ^(b-1))%c

3. If we have to return the mod of a negative number x whose absolute value is less than y: then (x + y) % y will do the trick.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int modPow(int a, int b, int c)
{

    if (a == 0)
        return 0;
    if (b == 0)
        return 1;

    long y;
    if (b % 2 == 0)
    {
        y = modPow(a, b / 2, c);
        y = (y * y) % c;
    }

    else
    {
        y = a % c;
        y = (y * modPow(a, b - 1, c) % c) %
    }

    return (int)((y + c) % c);
}

int main()
```

BOSSCODER ACADEMY UPSKILL WITH US

```
    int a, b, c;
    cin >> a >> b >> c;

    cout << modPow(a, b, c);

    return 0;
}
```

Time Complexity: O(logn)
Space Complexity: O(logn)

Problems

1. **The string is palindrome or not**
   Given a string **S**, check if it is a palindrome or not. (A palindrome is a string which is the same from forward and backwards.)
   Example
   Input: S = "abba"
   Output: 1
   Approach
   The idea of a recursive function is simple
   1. If there is only one character in the string return true.
   2. Else compare the first and last characters and recur for the remaining substring.
   Implementation

```
#include <bits/stdc++.h>
using namespace std;

bool isPalRec(string str,
              int s, int e)
```

BOSSCODER ACADEMY UPSKILL WITH US

```
    if (s == e)
        return true;

    if (str[s] != str[e])
        return false;

    if (s < e + 1)
        return isPalRec(str, s + 1, e - 1);

    return true;
}

bool isPalindrome(string s)
{
    int n = s.size();

    if (n == 0)
        return true;

    return isPalRec(s, 0, n - 1);
}
int main()

{
    string s;
    cin >> s;
    if (isPalindrome(s))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

i-ith index are equal or not. If they are not equal return false and if they are equal then continue with the recursion calls.

```
#include <bits/stdc++.h>
using namespace std;

bool isPalindrome(string s, int i){

    if(i > s.size()/2){
       return true ;
    }

    return s[i] == s[s.size()-i-1] && isPal

}

int main()

{
    string s;
    cin >> s;
    if (isPalindrome(s,0))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(n)

given string

Given a string S, the task is to write a program to print all permutations of a given string

Example

Input: S = “ABC”

Output: “ABC”, “ACB”, “BAC”, “BCA”, “CBA”, “CAB”

Approach-1 Using Backtracking

Backtracking is an algorithmic strategy for recursively solving problems by attempting to develop a solution gradually, one step at a time, and discarding any solutions that do not satisfy the problem's criteria at any point in time.

We’ll define a function generatePermutaionsHelper(Str, l, r). This function will generate the permutations of the substring starting from index “l” and ending at index “r”.Calling the above function, generatePermutaionsHelper(Str, l, r).

If “l” is equal to “r”, a new permutation is found. Insert this string in the “ans” list. Else, continue to iterate on the string from “l” to“r”.Let “i” denote the current index. Swap Str[l] and Str[i] to fix the “ith” character on the index ”l”.

Call generatePermutaionsHelper(Str, l + 1, r) to get the permutation of the rest of the characters. Now, backtrack and swap Str[l] and Str[i] again. In the end, we’ll have the list “ans” having all the permutations of the given string.

Implementation

BOSSCODER ACADEMY UPSKILL WITH US

```
void generatePermutationsHelper(string &str
{

    if (l == r)
    {
        ans.push_back(str);
        return;
    }
    for (int i = l; i <= r; i++)
    {
        swap(str[l], str[i]);
        generatePermutationsHelper(str, l +
        swap(str[l], str[i]);
    }
}

int main()
{

    vector<string> ans;
    string str;
    cin >> str;

    int l = 0;
    int r = str.size() - 1;

    if (str.length() == 0)
    {
        cout << "No Permutations Possible!!
    }
    else
        generatePermutationsHelper(str, l,
    for (int i = 0; i < ans.size(); i++)
    {
```

```
    return 0;
}
```

Approach-2 Avoid Repetition Using Backtracking

Create a recursive function and pass the input string and a string that stores the permutation (which is initially empty when called from the main function). If the length of the string is 0, print the permutation. Otherwise, run a loop from i = 0 to N: Consider S[i], to be a part of the permutation. Remove this from the current string and append it to the end of the permutation. Call the recursive function with the current string which does not contain S[i] and the current permutation.

Implementation

```
#include<bits/stdc++.h>
using namespace std;

void permute(string s, string answer)
{
    if (s.length() == 0) {
        cout << answer << endl;
        return;
    }
    for (int i = 0; i < s.length(); i++) {
        char ch = s[i];
        string left_substr = s.substr(0, i)
        string right_substr = s.substr(i +
        string rest = left_substr + right_s
        permute(rest, answer + ch);
    }
```

```
int main()
{
    string s;
    cin>>s;
    string ans = "";
    if(s.length()==0)
    {
        cout<<"No Permutations Possible!!";
    }
    else
        permute(s, ans);
    return 0;
}
```

Time Complexity: O(N * N!) i.e. there are N! permutations and it requires O(N) time to print a permutation.
Space Complexity: O(|S|)

BOSSCODER ACADEMY UPSKILL WITH US

for

Read

Date: 12th May, 2023

and System Design

Read

Date: 5th May, 2023

View All blogs

Helping ambitious learners upskill themselves & shift to top product based companies.

Lets hear all about it.

Who are we

About us
Blog
Attend a FREE Event
Privacy Policy
Terms & Condition
Pricing and Refund Policy

Contact Us

Email: ask@bosscoderacademy.com

Follow us on

LinkedIn
Youtube
Instagram
Telegram
Reviews on Quora