1 . Recursion on Subsequences

(▶ L6. Recursion on Subsequences | Printing Subsequences)

Power-Set is The Most Efficent method for generating subsequnces (▶ Power Set | Print all Subsequences)

Note : Leetcode-pattern post

```java
import java.io.*;
import java.util.*;
class Subsequence {
    public static void main (String[] args) {
        int arr[]={1,2,3};
        int n=arr.length;
        helpr(arr,n,0,new ArrayList<Integer>());

    }
    static void helpr(int arr[],int n,int i,ArrayList<Integer> list){
        if(i==n){
            System.out.println(list);
            return;
        }
        list.add(arr[i]);
            //Take the particular index into Subsequences
        helpr(arr,n,i+1,list);
        list.remove(list.size()-1);
            //Do Not Take the particular index into Subsequences
        helpr(arr,n,i+1,list);
    }
}
```

Questions On same pattern as Above Approch

2. Subsequence sum equal to k

(▶ L7. All Kind of Patterns in Recursion | Print All | Print one | Count)

```java
class Solution {
    public List<List<Integer>> SubsequenceSumEqualToK(int[] candidates, int target) {
        List<List<Integer>> ans=new ArrayList<>();
        helper(candidates,0,target,ans,new ArrayList<Integer>(),0);
        return ans;
    }
    void helper(int arr[],int i,int target,List<List<Integer>> ans,ArrayList<Integer>
temp,int sum){
        if(i==arr.length){
            if(sum==target) ans.add(new ArrayList<>(temp));
            return;
        }
        temp.add(arr[i]); sum+=arr[i];
        helper(arr,i+1,target,ans,temp,sum);
        int val=temp.remove(temp.size()-1); sum-=arr[i];
        helper(arr,i+1,target,ans,temp,sum);
    }
}
```

3. Print only 1 Subsequence Whose sum equal to K

(L7. All Kind of Patterns in Recursion | Print All | Print one | Count)

```java
import java.util.*;
import java.io.*;
class Solution {
    static public void SubsequenceSumEqualToK(int[] candidates, int target) {
        helper(candidates,0,target,new ArrayList<Integer>(),0);
    }
    static boolean helper(int arr[],int i,int target,ArrayList<Integer> temp,int sum){
        if(i==arr.length){
            if(sum==target){
              System.out.println(temp);
              return true; //condition satisfied
            }
            return false; //condition does not satisfied
        }
        temp.add(arr[i]); sum+=arr[i];
        if(helper(arr,i+1,target,temp,sum)) return true;
        int val=temp.remove(temp.size()-1); sum-=arr[i];
        if(helper(arr,i+1,target,temp,sum)) return true;
        return false;
    }
    public static void main (String[] args) {
        SubsequenceSumEqualToK(new int[]{1,2,3,4,5},6);
    }
}
```

4.Count Subsequence sum equal to k

(L7. All Kind of Patterns in Recursion | Print All | Print one | Count)

```java
import java.util.*;
import java.io.*;
class Solution {
    static public void SubsequenceSumEqualToK(int[] candidates, int target) {

        System.out.println(helper(candidates,0,target,0));
    }
    static int helper(int arr[],int i,int target,int sum){
        if(i==arr.length){
            if(sum==target){
              return 1; // if found return 1
            }
            return 0; //else return 0
        }
        sum+=arr[i];
        int l=helper(arr,i+1,target,sum); // answer from left call
        sum-=arr[i];
        int r=helper(arr,i+1,target,sum); // answer from right call
        return l+r; //return ans from left+right call
    }
    public static void main (String[] args) {
        SubsequenceSumEqualToK(new int[]{1,2,3,4,5},6);
    }
}
```

Some Approches on find combination sum equal to k

5. Combination Sum - You can pick an element any number of time :

(L8. Combination Sum | Recursion | Leetcode | C++ | Java

```java
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans=new ArrayList<>();
        helper(candidates,0,target,ans,new ArrayList<Integer>());
        return ans;
    }
    void helper(int arr[],int i,int target,List<List<Integer>> res,List<Integer> ds){
        if(i==arr.length){
            if(target==0){
                res.add(new ArrayList<>(ds));
            }
            return;
        }
        if(arr[i]<=target){ //only pick when the current element is less than or equal to target
            ds.add(arr[i]); // add curent element
                            //insted of adding current elemnt to a new variable we will try to reduce the current target to zero
            helper(arr,i,target-arr[i],res,ds); //pick, pick and pick the current element until , the target become less than the current element
            ds.remove(ds.size()-1); //remove the current element
        }
        helper(arr,i+1,target,res,ds); // does not include current element
    }
}
```

- *time complexity: 2^t(target) * k*
- *Space complexity : k(Avg. Length)*x(combinations)*

6. Combination Sum II

(L9. Combination Sum II | Leetcode | Recursion | Java | C++)

```java
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> ans=new ArrayList<>();
        Arrays.sort(candidates); //we are sorting since we need to give ans in sorted order and ignore duplicate combinations
        helper(candidates,0,target,ans,new ArrayList<Integer>());
        return ans;
    }
    void helper(int arr[],int ind,int target,List<List<Integer>> ans,ArrayList<Integer> ds){
        if(target==0){
            ans.add(new ArrayList<>(ds));
            return;
        }
        for(int i=ind; i<arr.length; i++){
            if(i>ind && arr[i]==arr[i-1]) continue; // ignore if same value occure for getting all unique combinations
            if(arr[i]>target) break; // this means we can not form combinations
            ds.add(arr[i]);
            helper(arr,i+1,target-arr[i],ans,ds);
            ds.remove(ds.size()-1);
        }
    }
}
```

- *time complexity: 2^n(for genrating all combination) * k(avg length of every combination,for adding combination to answers)*
- *Space complexity : k(Avg. Length)*x(combinations)*

Approaches on subsets sum

Subset-sum : (L10. Subset Sum I | Recursion | C++ | Java)

```java
import java.io.*;
import java.util.*;

class Solution{
    ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N){
        ArrayList<Integer> ans=new ArrayList<>();
        helper(arr,0,0,ans);
        return ans;
    }
    void helper(ArrayList<Integer> arr,int idx,int sum,ArrayList<Integer> ans){
        if(idx==arr.size()){
            ans.add(sum);
            return;
        }
        sum+=arr.get(idx);
        helper(arr,idx+1,sum,ans);
        sum-=arr.get(idx);
        helper(arr,idx+1,sum,ans);
    }
}
```

- *time complexity: 2^n*
- *Space complexity : 2^n(i did not understand why)*
-

Subset II : (L11. Subset Sum II | Leetcode | Recursion)

```java
class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        List<List<Integer>> ans=new ArrayList<>();
        Arrays.sort(nums);
        helpr(nums,nums.length,0,new ArrayList<Integer>(),ans);
        return ans;
    }
    static void helpr(int arr[],int n,int idx,ArrayList<Integer> list,List<List<Integer>> ans){
        // if(idx==n){
            ans.add(new ArrayList<>(list));
            // return;
        // }
        for(int i=idx; i<n; i++){
            if(i>idx && arr[i]==arr[i-1]) continue;
        list.add(arr[i]);
            //Take the particular index into Subsequences
        helpr(arr,n,i+1,list,ans);
        list.remove(list.size()-1);
            //Do Not Take the particular index into Subsequences
        }
    }
}
```

- *time complexity: 2^n*n(assume that every subset is near about size n)*
- *Space complexity : 2^n*k (subsets avg length k)*

Print all Permutations

Permutations :

(L12. Print all Permutations of a String/Array | Recursion | Approach - 1)

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans=new ArrayList<>();
        helpr(nums,nums.length,new ArrayList<Integer>(),ans,new boolean[nums.length]);
        return ans;
    }
      static void helpr(int arr[],int n,ArrayList<Integer> list,List<List<Integer>> ans,boolean vis[]){
        if(list.size()==n){ //when the size of aur data structure become equals to n this mean we have form out first permutation
            ans.add(new ArrayList<>(list));
            return;
        }
        for(int i=0; i<n; i++){
        if(vis[i]) continue; // if the current elemnt is used before in this permutation do not use it
        list.add(arr[i]); vis[i]=true; //when use the current element mark it as true so that we can not use it again
        helpr(arr,n,list,ans,vis);
        list.remove(list.size()-1); vis[i]=false; //while backtrack mark the used element as false so that we can use the same element in diffrent permutation
        }
    }

}
```

- *time complexity: n!(for permutations)*n*
- *Space complexity :if we ignore for ans operations then O(n)(store ds)+O(n)(map array)*

Permutations Approch 2 :

(L13. Print all Permutations of a String/Array | Recursion | Approach - 2)

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans=new ArrayList<>();
        helpr(nums,nums.length,0,ans);
        return ans;
    }
       void helpr(int arr[],int n,int idx,List<List<Integer>> ans){
        if(idx==n){ ///if index is crossing the array boundary then push the current state of permutated numbers
                        //in the array to ans
            ArrayList<Integer> ds=new ArrayList<>();
            for(int num:arr) ds.add(num);
            ans.add(new ArrayList<>(ds));
        }
        for(int i=idx; i<n; i++){ //try swapping this number with every number ahead
            swap(arr,i,idx); // swap each element till its nTh elemnt
            helpr(arr,n,idx+1,ans);//recursively call the function from next index
            swap(arr,i,idx); //swap back to original state so that when recursion call returns to this level it can explore other possibilites
        }
    }
    void swap(int arr[],int i,int j){
        int temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
    }

}
```

- *time complexity: n!(for permutations)*n*
- *Space complexity :if we ignore for ans operations .(recursion space O(n), O(n!) for returning answers*

N-Queens : (L14. N-Queens | Leetcode Hard | Backtracking)

```java
class Solution {
    public List<List<String>> solveNQueens(int n) {
        char board[][]=new char[n][n];
        List<List<String>> ans=new ArrayList<>();
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                board[i][j]='.';
            }
        }
        helper(0,board,ans);
        return ans;
    }
void helper(int row,char board[][],List<List<String>> ans){
        if(row==board.length){
            ArrayList<String> res=new ArrayList<>();
            for(char ch[]:board){
                res.add(new String(ch));
            }
            ans.add(res);
            return;
        }
        for(int col=0; col<board.length; col++){
            if(board[row][col]=='.' && isQueenSafe(board,row,col)){ //check is the queen postion is safe for placing or Not
                board[row][col]='Q'; //if safe then place thq queen
                helper(row+1,board,ans);
                board[row][col]='.'; //while backtrack remove the queen to explore other posblties
            }
        }
    }
//if(board[row][col]=='.' && left[col]==false && diagonal[row+col]==false && upper[(board.length-1)+(col-row)]==false) ← this hashing method is also can be used to check whether the queen is placed at the right position or not

    boolean isQueenSafe(char chess[][],int row,int col){ //function to check wheter placing the queen at this postion is safe or not
            for(int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
                if(chess[i][j] == 'Q')
                    return false;
            for(int i = row - 1, j = col; i >= 0; i--)
                if(chess[i][j] == 'Q')
                    return false;            }
            for(int i = row - 1, j = col + 1; i >= 0 && j < chess.length; i--, j++){
                if(chess[i][j] == 'Q')
                    return false;            }
            for(int i = row, j = col - 1; j >= 0; j--)
                if(chess[i][j] == 'Q')
                    return false;
            return true;
    }
}
```

- *time complexity:*
- *Space complexity :*

Sudoku Solver : (L15. Sudoko Solver | Backtracking)

```java
class Solution {
    public void solveSudoku(char[][] board) {
         helper(board);
    }
    boolean helper(char[][] board){
        for(int i=0; i<9; i++){
            for(int j=0; j<9; j++){
                if(board[i][j]=='.'){ //traverse the matrix and find the empty place
//                       once we find the empty place than we tried all the numbers from 1 to 9 and check that it is a valid number or not by checking the rules.
// and we find the correct number for that place than we find for the second empty place in 9 X 9 matrix.for second empty place we repeat the same process
                    for(char ch='1'; ch<='9'; ch++){
                        if(isValid(board,i,j,ch)){
                            board[i][j]=ch;
                            if(helper(board)) return true; //and after all recursive calls we got true, than we have to stop over there only and no need to search for other solutions.
                            else board[i][j]='.'; //after getting the false from solve(board) function we make all the places empty that we have filled . than try for other member for first empty place
                        }
                    }
                    return false; //and if we doesn't get any number, so we return false.
                }

            }
        }
        return true;
    }
    private boolean isValid(char[][] board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] != '.' && board[i][col] == c) return false; //check row
            if(board[row][i] != '.' && board[row][i] == c) return false; //check column
            if(board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c) return false; //check 3*3 block
        }
        return true;
    }
}
```

- *time complexity:9 ^ m (m represents the number of blanks to be filled in), since each blank can have 9 choices.*
- *Space complexity :*

M-Coloring Problem : (L16. M-Coloring Problem | Backtracking)

```java
class solve {
    // Function to determine if graph can be coloured with at most M colours
    // such
    // that no two adjacent vertices of graph are coloured with same colour.
    public boolean graphColoring(boolean graph[][], int m, int n) {
        return helper(graph,new int[n],m,n,0);
    }
    boolean helper(boolean graph[][],int col[],int m,int n,int idx){
        if(idx==n) return true;

           for(int i=1; i<=m; i++){ //trying for each colour
               if(isSafe(graph,col,i,idx)){ //is placing the colour isSafe(no adjcent node have same color) then we will color the node wich particlar color
                   col[idx]=i;
                   if(helper(graph,col,m,n,idx+1)) return true; //if this returns true this means we have sucessfully colord our graph
                   else col[idx]=0; //if it is not possible to color it with ith colour then we will get it backtrack to its orignal state to explore other possibilities
               }
           }
           return false;
       }
      boolean isSafe(boolean graph[][],int col[],int color,int idx){
               for(int i=0; i<graph.length; i++){ //checking adjcent nodes
                   if(graph[idx][i]){
                       if(col[i]==color) return false;
                   }
               }
               return true;
       }
  }
```

- *time complexity:(M^N)*N(isSafe())*
- *Space complexity :N(color array)+N(recursion space)*

Palindrome Partitioning :

(L17. Palindrome Partitioning | Leetcode | Recursion | C++ | Java)

```java
class Solution {
    public List<List<String>> partition(String s) {
        List<List<String>> ans=new ArrayList<>();
        helper(s,0,ans,new ArrayList<String>());
        return ans;
    }
    void helper(String s,int idx,List<List<String>> ans,ArrayList<String> ds){
        if(idx==s.length()){
            ans.add(new ArrayList<>(ds));
            return;
        }
        for(int i=idx; i<s.length(); i++){
            if(isPalindrom(s,idx,i)){
                //1. check if the choosen substring is a palindrom or not , if the current substring is a palindrom then add this into our DS and check for further substring who are palindrom.
                //2. if the input is "aab", check if [0,0] "a" is palindrome. then check [0,1] "aa", then [0,2] "aab".
  // While checking [0,0], the rest of string is "ab", use ab as input to make a recursive call.
                ds.add(s.substring(idx,i+1)); //choose
                helper(s,i+1,ans,ds);
                ds.remove(ds.size()-1); //unchoose
            }
        }
    }
    boolean isPalindrom(String str,int l,int r){
        while(l<=r){
            if(str.charAt(l++)!=str.charAt(r--)) return false;
        }
        return true;
    }
}
```

- *time complexity: O(n*(2^n))*
- *Space complexity :*

Rat in a Maze : (L19. Rat in A Maze | Backtracking)

```java
class Solution {
    public static ArrayList<String> findPath(int[][] m, int n) {
        ArrayList<String> ans=new ArrayList<>();
        helper(m,0,0,n,ans,new String());
        return ans;
    }
    static void helper(int m[][],int row,int col,int n,ArrayList<String> ans,String str){
        if(row<0 || col<0 || row>=n || col>=n || m[row][col]==0) return;
        if(row==n-1 && col==n-1){
            ans.add(str);
            // System.out.println(str);
            return;
        }
        m[row][col]=0; //smart way of mark the direction as visited we make this cell 0 so in this qus zero trated as the object in the path sow we can not move again in this direction if we mark this node as 0
        helper(m,row+1,col,n,ans,str+"D");
        helper(m,row,col+1,n,ans,str+"R");
        helper(m,row-1,col,n,ans,str+"U");
        helper(m,row,col-1,n,ans,str+"L");
        m[row][col]=1; //while backtracking we revert back the maze to its orignal postion to explore diffrent paths
    }
}
```

- *time complexity: 4^(n*m)(because we have to go into all four direction)*
- *Space complexity : m*n(if we explore all the path for reacting n-1 and m-1)*

Kth-Permutation Sequence : (L18. K-th Permutation Sequence | Leetcode)

```java
class Solution {
    public String getPermutation(int n, int k) {
        //n=4, k=10
        int fact=1;
        ArrayList<Integer> list=new ArrayList<>();
        String ans="";
        for(int i=1; i<=n; i++){
            fact=fact*i;
            list.add(i);
        }
        //fact=24
fact=fact/list.size(); //fact=24/4=6 so-->
//we are doing this because there are fact/list.size() permutions sequnece which will start from particular number
// say n = 4, you have {1, 2, 3, 4}

// If you were to list out all the permutations you have

// 1 + (permutations of 2, 3, 4) 6 permutation will start with 1

// 2 + (permutations of 1, 3, 4) 6 permutation will start with 2

// 3 + (permutations of 1, 2, 4) 6 permutation will start with 3

// 4 + (permutations of 1, 2, 3) 6 permutation will start with 4


        k=k-1; //since we are dealing with 0 based indexing k=10-1 => k=9
        while(true){
            ans+=String.valueOf(list.get(k/fact));  //(9/6=1) in zero based index we come to know that out no
            //will start from 2 ((line 20) 2 + (permutations of 1, 3, 4) )
            // by k/fact we will know what will be our first no, it will tell us the sequnec in which our no lies so we will get our no
            list.remove(k/fact); //simply remove that no so list become 1 3 4
            if(list.size()==0) break;
            k=k%fact; // the 9%6=3 so the qustion brek down find the 3rd pemutation sequence of 1 3 4
            fact=fact/list.size(); //6=6/3=2
            //so k=3 and fact=2 now repeat the above steps until list.size()!=0
        }
        return ans;
    }
}
```

- *time complexity: O(n)(for looking for n numbers)*O(n)(for evertime removing element from array it will take time so)=O(n^2)*
- *Space complexity :O(n)(using array list and storing ans)*