Page No.
Date 14-01-25

Backtracking

- Recursion is dumb. It traverse in all paths knowing that it won't find answer in this path.
- What Backtracking does is it marks all those path false jaha se answer tak nahi pahuch skte.
- It checks that iss path me jaane ka koi faida bhi hai, if no then it marks it false and backtrack kar jaata hai
- Helps in optimisation of the code.

Find Subsets

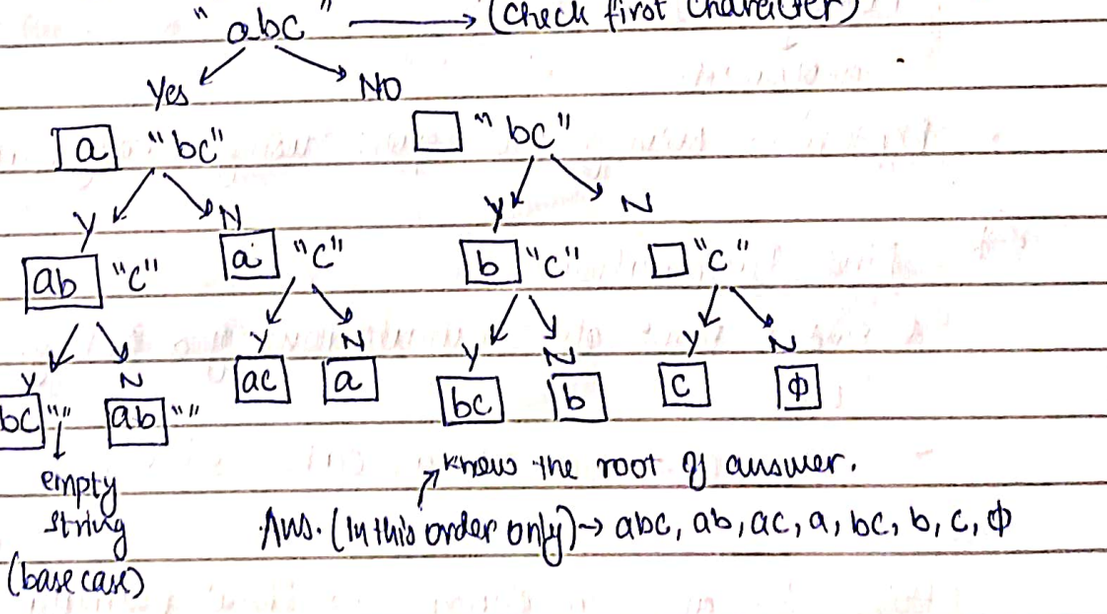
Q. Find and print all subsets of a given string.

"abc"

→ a, b, c, ab, bc, ac, abc, ϕ = 8 subsets

Note :- for any string length 'n' → Total subsets = 2^n (elements/ (class 11 maths))

Approach → (Recursive)



Note:- In recursion post order traversal is done. (L-R-N) → (similar to) Here in this ques.

- The process of going back after reaching to one of the solution is called Backtracking (Brute force backtracking)

Ajanta
Page No.
Date

code =>
```
static void findSubsets(String str, String ans, int i){
    // base condition
    if(i == str.length()){
        print(ans);
        return;
    }
    // Yes (on left side :. print first)
    findSubsets(str, ans+str.charAt(i), i+1);
    // NO
    findSubsets(str, ans, i+1);
}
```

(Backtracking) ek ans. milne pr baaki ke liye return; → ★★★ v.imp. to mention ★

Yes pehle likhte hai wo matter karta hai

take and proceed

initialise this in main fn. → findSubsets("abc", "", 0); ✓

DRY run this properly to get the understanding (Jitna chota dikhta hai utna hai nhi) in stack format (or check video, if you don't know why I wrote this)

Strings are immutable

- T.C. → $O(2^n \times n)$ → ek subset ko nikalne ka time.
  - no. of subsets
- Explore → solving this ques. using StringBuilder.

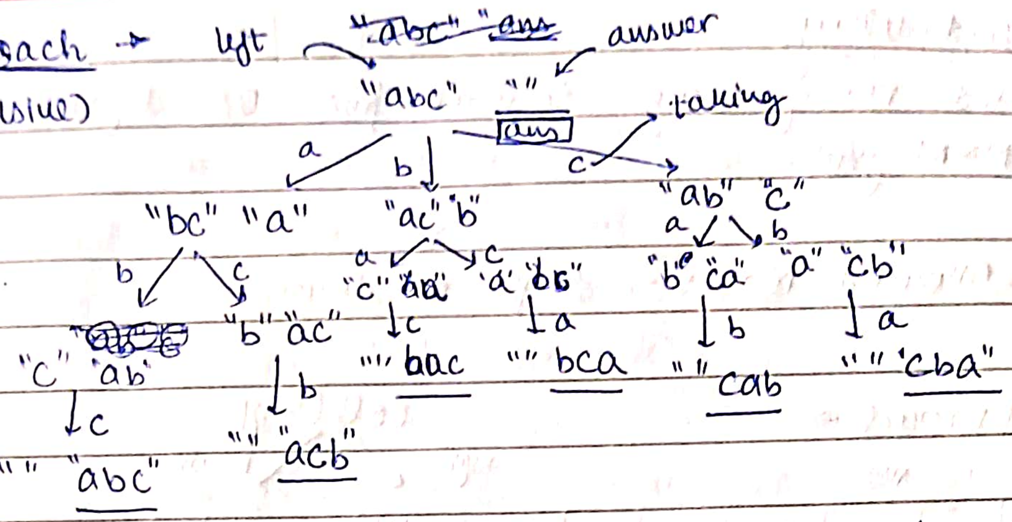
✓* **Find Permutations**

Q. Find & print all permutations of a string.
"abc"
→ abc, acb, bac, bca, cab, cba.

Note:- for any 'n' elements → Total permutations = n! (class 12 maths)

★★★★★ Note:- Find subsets and Permutations are enumerations type of backtracking where you need all solutions possible.

- Approach → (recursive)



- Code →

```
static void permutations(String str, String ans){
    //base case
    if(str.length() == 0){
        print(ans);
        return;
    }
    // Take all character choices individually
    for(i = 0 to str.length()){
        * char curr = str.charAt(i);
```

*→ Now to remove one character from string, take two substring and append them. e.g. "abcde" —remove "c"→ "ab" + "de" = abde ✓.

```
        String NewStr = str.substring(0, i) + str.substring(i+1);
```

*→ We made a NewStr, not used just str because usaka length change ho jayega aur loop mein dikkat hogi.

*→ In this str.substring(0, i) — 0 is inclusive, i) → this bracket is exclusive

also for till the end → str.substring(i+1, str.length()) == str.substring(i+1) → both are equal.

```
        permutations(NewStr, ans+curr);
    }
}
```

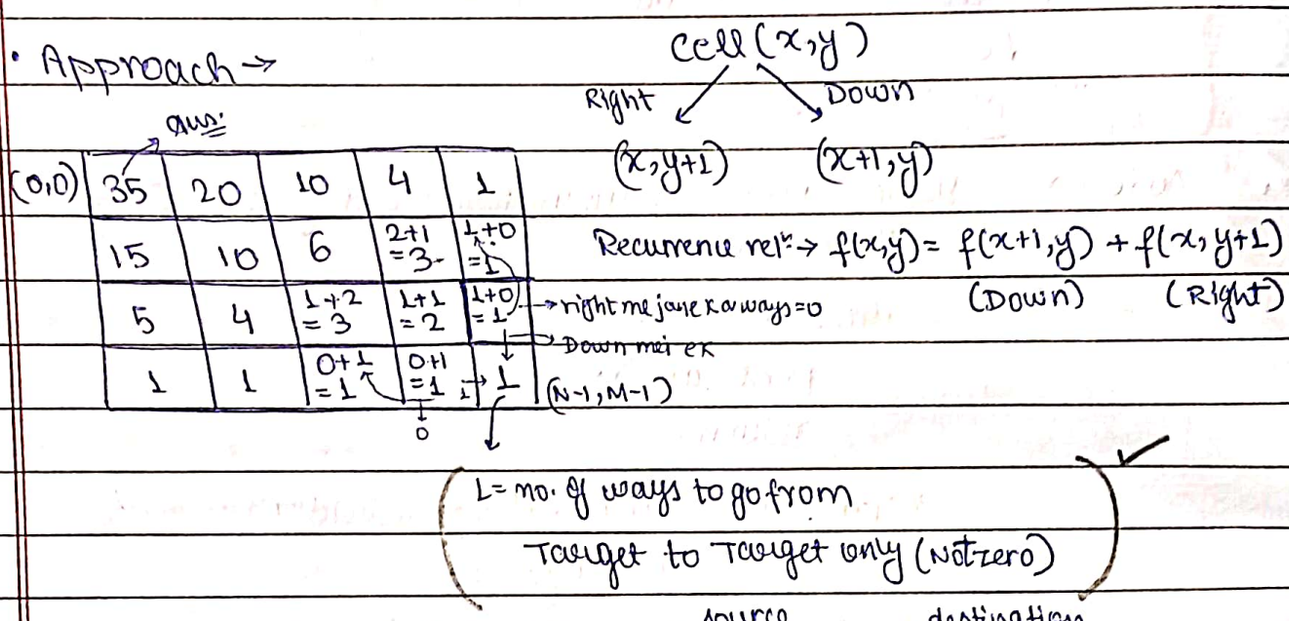
T.C. → O(n × n!).

* Grid Ways

Q. Find no. of ways to reach from (0,0) to ((N-1),(M-1)) in a NXM grid.

I. Allowed Moves - right or down

• Approach →



Recurrence rel" → $f(x,y) = f(x+1,y) + f(x,y+1)$ (Down) (Right)

• code →

```
static int gridWays(int i, int j, int n, int m){
    //base case
    if(i== n-1 && j== n-1){ // at last cell
        return 1;
    }
    else if(i==n || j== m){ // at boundary (for any)
        return 0;
    }
    int w1 = gridWays(i+1, j, n, m); //Right
    int w2 = gridWays(i, j+1, n, m); //Down
    return w1 + w2;
}
```

initialise → print in main → gridWays(0,0, 3,3); //6.
" (0,0, 5,4); //35.

• T.C = $O(2^{(n+m)})$; Not O(n x m)
↳ very bad

How 2^{n+m}, total right = no. of col = n
" down = " " rows = m

Ajanta
Page No.
Date

- Trick to optimise this in linear time:-

for ways $(m-1)$, $(m-1)$ we have { D, D, D, D ... } , { R, R, R, R ... }
(D) (R)

eg. for 2×2 grid → DD, RR

∴ Total permutations possible to reach dest. → DDRR, DRDR, DRRD, RRDD ...

∴ Total ways = $\frac{(n-1+m-1)!}{(n-1)!\,(m-1)!}$ ← $\frac{2}{D} \times \frac{2}{D,R} \times \frac{2}{D,R} \times \frac{1}{D} = 8$

(in case of duplicates)

$(n-1)$ no. of D as duplicates ; $(m-1)$ for R.

This will give $O(n+m)$.

By using this formula directly → $W = \frac{(n-1+m-1)!}{(n-1)!\,(m-1)!}$ Ans.

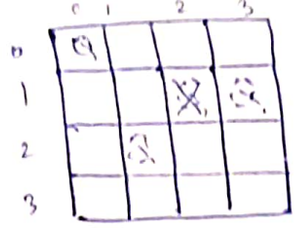
↓
for this src should be on (0,0) otherwise check grids.

Practice this for (if) → Allowed Moves = R, D, L
(check old notes if doubt) " " = R, D, L, Diagonal
** or any from all combinations (concept build ↑↑).
→ If obstacles are there

* N-Queen → Another Type of Backtracking (Main)
↳ maximum queen to be placed in a grid.
→ Recursion ways → n^2C_n → for a grid of n×n.
(Solve N-knight yourself after this)

→ Backtracking main picture comes here, it does not check all ways (n^2C_n) to give max. possible ans.

* Types of Back Tracking :-
1. Basic Backtracking (Brute Force) → e.g. permutation, subset
2. Pruning " → e.g. N-Queen, N-Knight, Sudoku Solver
3. Bounding " (Branch and Bound) → e.g. Travelling salesman problem, Knapsacks.
4. Heuristics " → e.g constraint puzzles, intelligent CSP
(Constrain satisfaction Prob: - less)



Ajanta
Page No.
Date

* In N-Queen Problem, we'll place a Queen at (0,0) then on another row place at (1,2), now come to third row we can't find any ways to place queen → backtrack → remove second queen and place at (1,3), now on third row place at (2,1).
Again on 4th row we can find anywhere to place ∴ backtrack 3 times and place 1st Queen on (0,1) & proceed similarly.

Q. Can we place N queens in N×N grid?

starting from which row

code → boolean nQueen(int[][]board, int row){
if(row == N) return true; → // base case
for(int col=0 to N){
if(issafe(board, row, col)){ ← checks for placing Queen at that position is valid or not by checking its row, col & diag.
board[row][col] = 1; ← place if its safe
if(nQueen(board, row+1)){ ← Now check for further that placing at their in starting was good or not.
return true;
}else{
board[row][col]=0; ← If true ∴ return true
else ∴ make it zero (backtrack for that position, it's currently at (not all))
}
return false;
}

static boolean issafe(int[][] board, int row, int col){
// check vertically up ← since, rows above current row can have Queen as we are placing in that fashion only
for(i=0 to row){
if(board[i][col] == 1) return false;
}
// check upper left diagonal
* for(i=row-1, j=col-1; i>=0 && j>=0; i--, j--){
if(board[i][j] == 1) return false; }
// check upper right diag :-
for(i=row-1, j=col+1; i>=0 && j<board.length; i--, j++){
if(board[i][j] == 1) return false; }
return true; } in last

- this type of loop ~~is~~ has only difference from nested loop that it's incrementing & decrementing simultaneously.

- Sudoku solver, Travelling salesman
  ↓
  using branch & bound + DP.