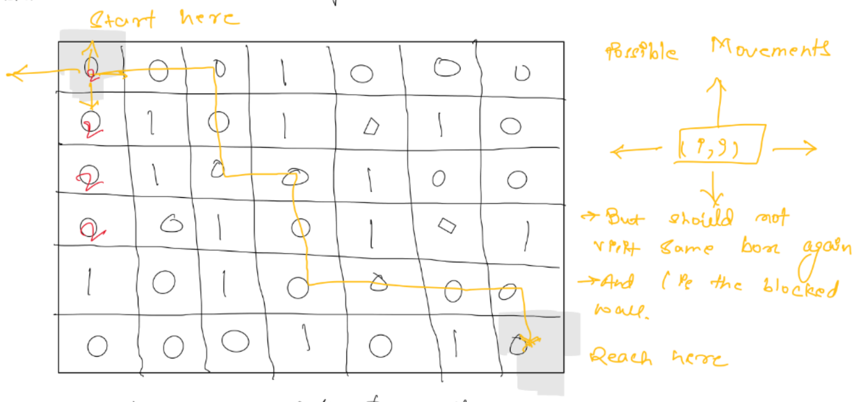
Backtracking 1

Rat In a Maze

Backtracking :- Subset of recursion + Generate all possibilities.

Recursion :- Solving problems with subproblems.

Problem 1 :- Rat in a maze.



→ But should not visit same box again

→ And i.e. the blocked wall.

Check if there is a path to reach

→ keep track of visited cells.

boolean [N] [N] → true (cells visited)
→ false (not visited)

Or

arr [N][N] → 0 (Empty cell)
→ 1 (Blocked cell)
→ 2 (visited)

Pseudo code :-

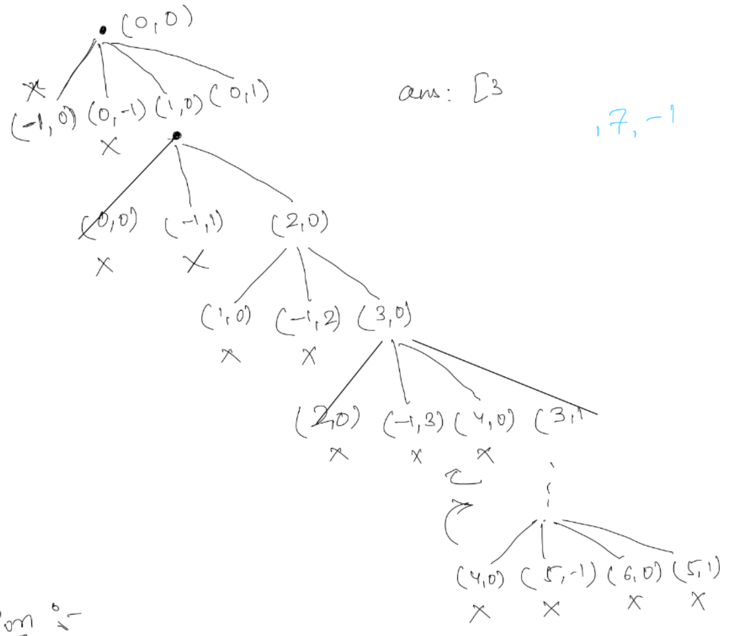
```
boolean checkPath (arr [N][M], i, j) {
        if (i == N-1 && j == M-1)
            return true;

        if (arr [i][j] == 1 || arr[i][j] == 2 || i<0 || j <0 || i>N || j>M)
            return false;

        arr [i][j] = 2;

        return  checkPath (arr, i-1, j) ||
                checkPath (arr, i, j-1) ||
                checkPath (arr, i+1, j) ||
                checkPath (arr, i, j+1);
}
```

Dry Run



ans: [3 ,7, -1

Optimization :-

$dx = [-1, 0, 1, 0]$
$dy = [0, -1, 0, 1]$

```
boolean checkPath (arr [N] [M], i, j)
{
    if ( i == N-1 && j = M -1)
        return true;

    arr[i][j] = 2

    dx = [-1, 0, 1, 0]
    dy = [0, -1, 0, 1]
    for (k = 0; k < 4; k++)
        ni = i + dx [k]
        nj = j + dy [k]
        if (nI >= 0 && nj >= 0 && nI < N && nJ < M
            && arr[nI][nJ] == 0)
            return checkPath (arr, nI, nJ);
    return false
}
```

$T.C : O(4 * M * N) = O(N * M)$
$S.C : O(N * M)$

Generate All Permutations (Unique Chars)

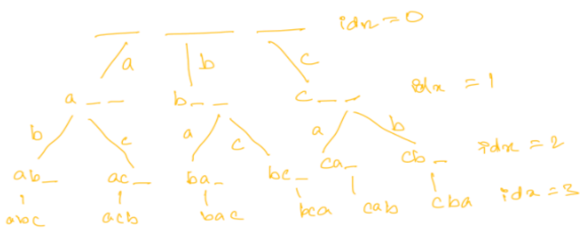
Problem 2 :-

Generate all permutations of a String, without modifying the string.

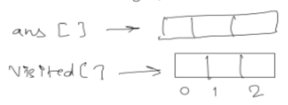
Char [] arr ⟶ Contains unique characters.

A: [a b c]   o/p → { abc acb bac / bca cab cba }

$\underline{3}\ \underline{2}\ \underline{1} = 6 \quad (N!)$

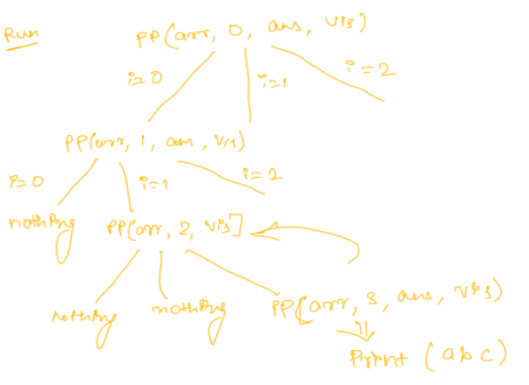


Char [] arr ← ip
idx → 0



```
Void printPermutation(char[] arr, int idx, char[] ans, bool[] vis)
{
    if (idx == n)
        print(ans);
        return;
    for(i=0; i<n; i++)
    {   if (visited [i] == false) {
            visited [i] = true;
            ans [idx] = arr [i]
            printPermutations(arr, idx+1, ans, vis)
            visited [i] = false;
    }   }
```

Dry Run



ans → abc
vis   t t t

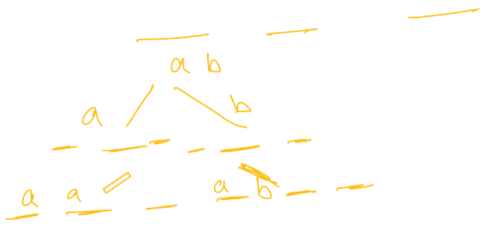
Generate All permutations (Duplicate Chars)

Permutations - 2

Print all permutations of given char array (duplicates allowed)

[a b a]

→ a a b
→ a b a
→ b a a



Pseudo code

```
                    0            Hashmap       temp char array
void permutation2(index, freq, temp[ ])
{
    // Base Condition
    if (index == temp.length) { Print (temp) return; }

    // Iterate over the key of Hashmap
    for ( key in freq ) {
        if ( freq [key] > 0)
            freq [key] -= 1;
            temp [index] = key
            permutation2( index+1 , freq , temp);
            freq [key] += 1;
        }
    }
```

T.C : O(N N!)
S.C : O(N)

Generate All Subsets

Subset Sum

Print the sum of all the subsets of a given A[]

A[] = {1 2 3}

for array of size 3 → 8

$\underline{2}\ \underline{2}\ \underline{2} = 2*2*2 = 8$

{ } → 0

{1} → 1

{2} → 2

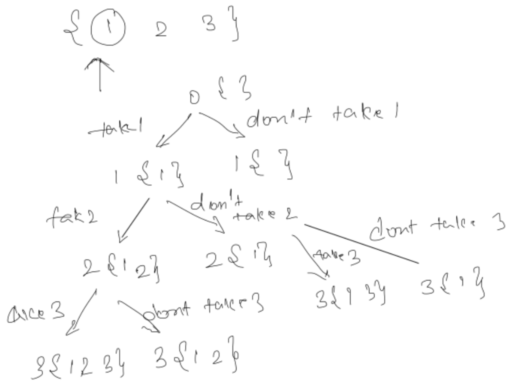
{3} → 3

{1, 2} → 3

{2 3} → 5

{1 3} — 4

{1, 2, 3} → 6



$$\boxed{\begin{array}{l} T.C : O(2^N) \\ S.C : O(N) \end{array}}$$

Pseudocode :-

```
Void SubsetSum (index, sum, A[]) {
    // Base Case
    if (index == A.length) {
        Print (sum);
        return;
    // take A[index]
    subsetSum (index+1, sum + A[index], A)
    // don't take A[index]
    subsetSum (index+1, sum, A);
}
```