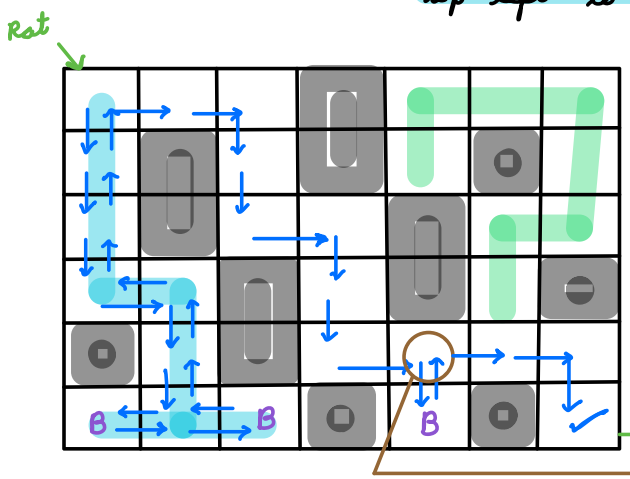
Recursion → Solving a problem using subproblems.

Backtracking → Trying all possibilities using recursion. (Bruteforce)

Q → Rat in a maze

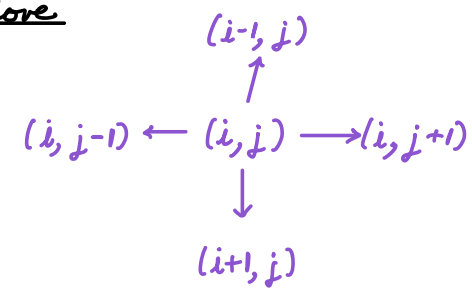
Zoho, Amazon

Check if it is possible to go from top left to bottom right cell in a maze with blocked cells.



I/P → A[N][M] → 0 empty, 1 blocked

Move



Exit

Blocked → Backtrack one step.

Keep track of visit cells.

Ans = true

1) vst[N][M] → SC = O(N*M)

2) A[N][M] → 0 empty, 1 blocked, 2 visited SC = O(1)

to get original matrix → update 2 → 0

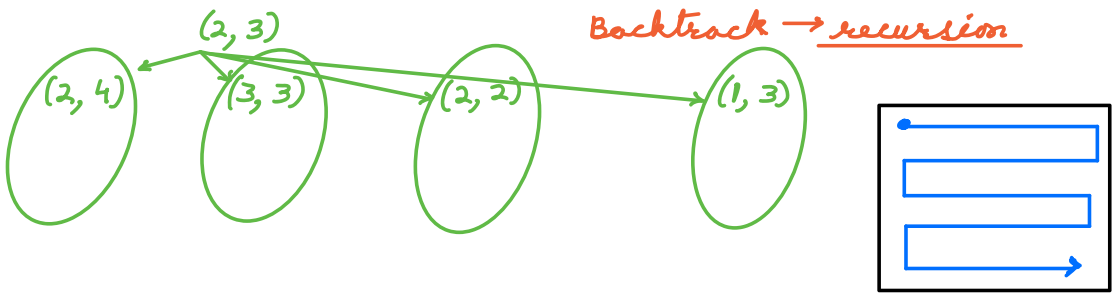
```
// A[N][M]
boolean check (i, j) {
    if (i == N-1 && j == M-1)   return true   // Reached
    if (i < 0 || i >= N || j < 0 || j >= M)   return false  // Outside
    if (A[i][j] == 1 || A[i][j] == 2)   return false  // Blocked or Visited

    A[i][j] = 2
```

// All possibilities

TC = O(N*M)

return check(i, j+1) || check(i+1, j) || SC ≤ O(N*M)
check(i, j-1) || check(i-1, j)

}



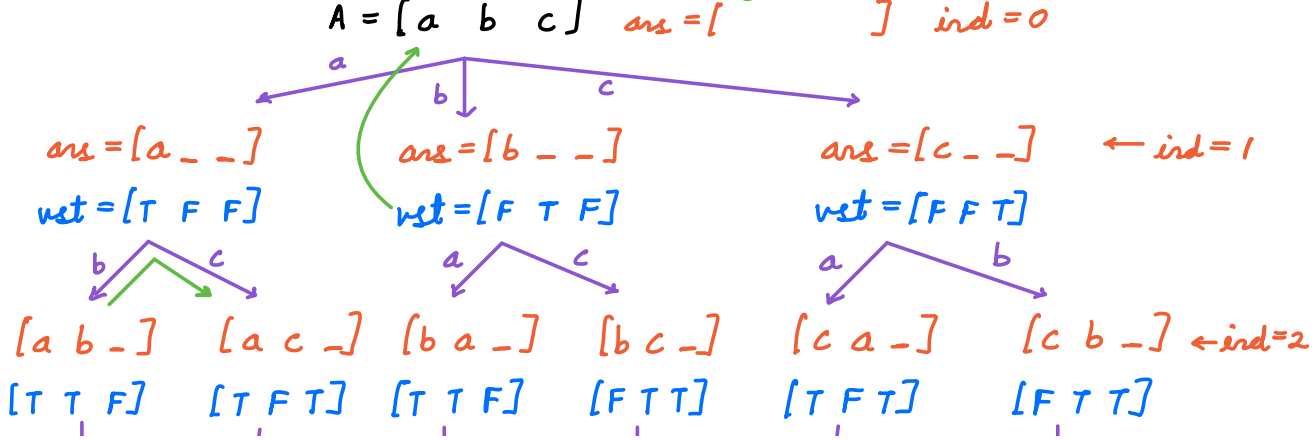
Q → Given a character array with distinct elements from a — z. Print all permutations of the array without modifying the input.

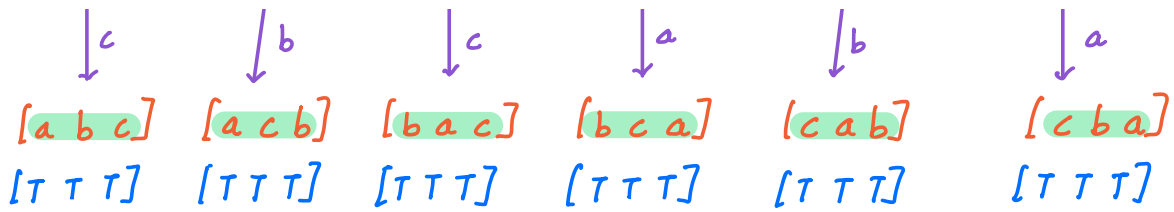
Max size of array → 26 ✓ small constraint ⇒ backtracking.

permutations → N! ✓

A = [a b c] o/p → a b c, a c b, b a c, b c a, c a b, c b a

✓ ✓ ✓
3 * 2 * 1 = 3!






```
// A[N]
void permute ( ans[], ind, vst[] ) {
    if ( ind == N ) {                    // Base Case
        print (ans)  (→ copy ans in a 2D list)
        return                           ✓
    }
→   for i → 0 to (N-1) {                 // All Possibilities
        if ( ! vst[i] ) {                // Valid Possibility
            vst[i] = true                // Do
            ans[ind] = A[i]
            permute ( ans, ind+1, vst )  // Recursion
            vst[i] = false               // Undo
        }
    }
}
```

to restore original state for next possibility. ✓

TC = O(N! * N) = O(N!) ✓ SC = O(N)

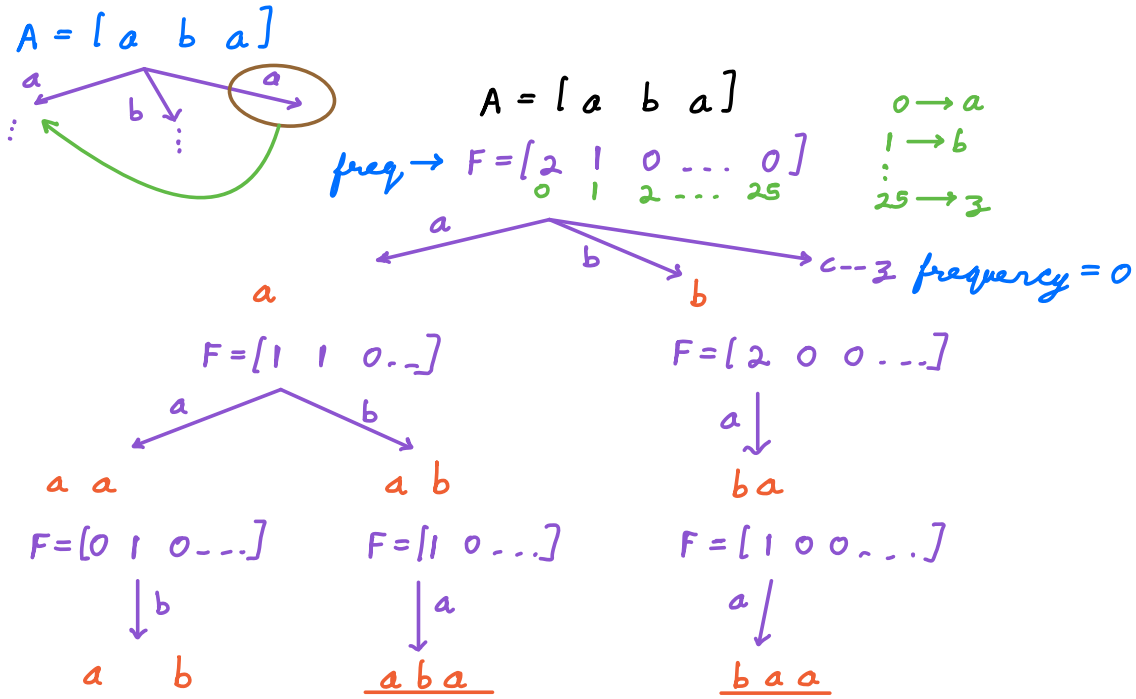
10:55 PM

Q→ Print all unique permutations of the given char array.

A = [a b a] o/p → a a b, a b a, b a a

permutations = N!

F[a]! * F[b]! * .. * F[z]!




```
void permute ( F[], N, ans[], ind) {                    // ind → 0
    if ( ind == N ) {                                   // Base Case
        print (ans)  (→ copy ans in a 2D list)
        return                                          ✓
    }
    for i → 0 to 25 {                                   // All Possibilities
        if (F[i] > 0) {                                 // Valid
            F[i] -= 1                                   // Do
            ans[ind] = (char) (i + 'a')                 ✓
            permute ( F, N, ans, ind + 1)               // Recursion
            F[i] += 1                                   // Undo
        }
    }
}
```

TC = O(# permutations) → O(N!)

SC = O(N)

Q → Generate all subsets for given array with unique elements.

A = [3 4] o/p → [ ] [3] [4] [3 4]

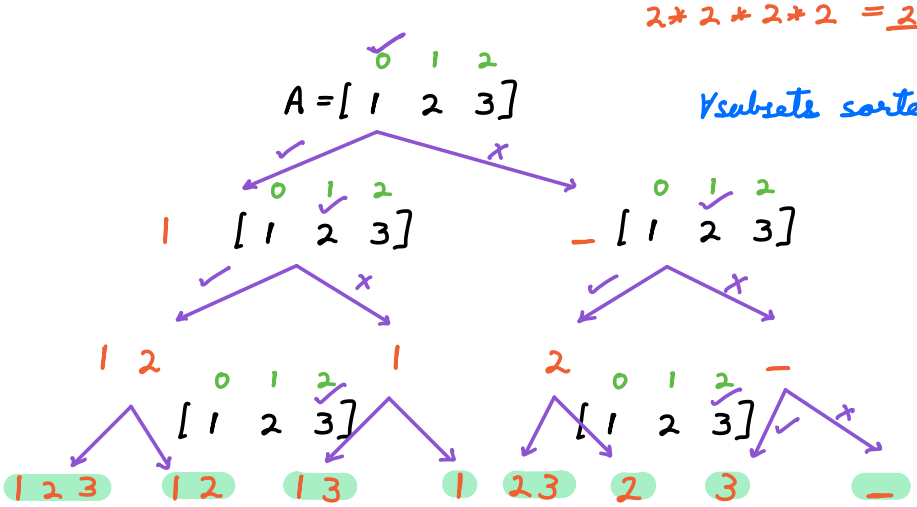
subsets = 2^N

A = [1 2 3 4]

✓ ✓ ✓ ✓

x x x x

2 * 2 * 2 * 2 = 2^4



subsets sorted → sort i/p

← arrange separately after generating subsets. ✓