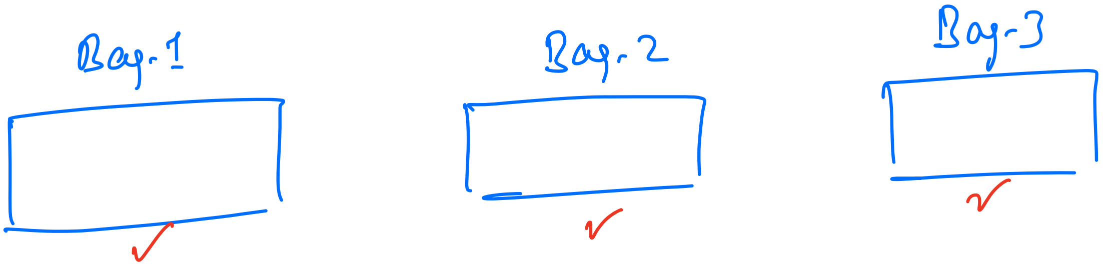
Recursion → solving a problem using sub-problems.

Backtracking → an algorithmic technique by which we can try out all the possibilities using recursion.

↓

Brute-force



Rat In A Maze [N*M]

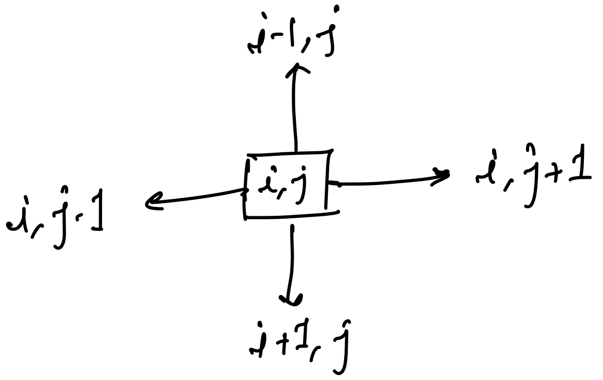
Check if it is possible to go from top-left to bottom right cell in a maze with blocked cell.

Note → You can't visit a cell more than once.

src

	0	1	2	3	4	5	6
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
2	0	1	0	0	1	0	0
3	0	0	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0

destination



arr[i][j] = 0 ["empty"]

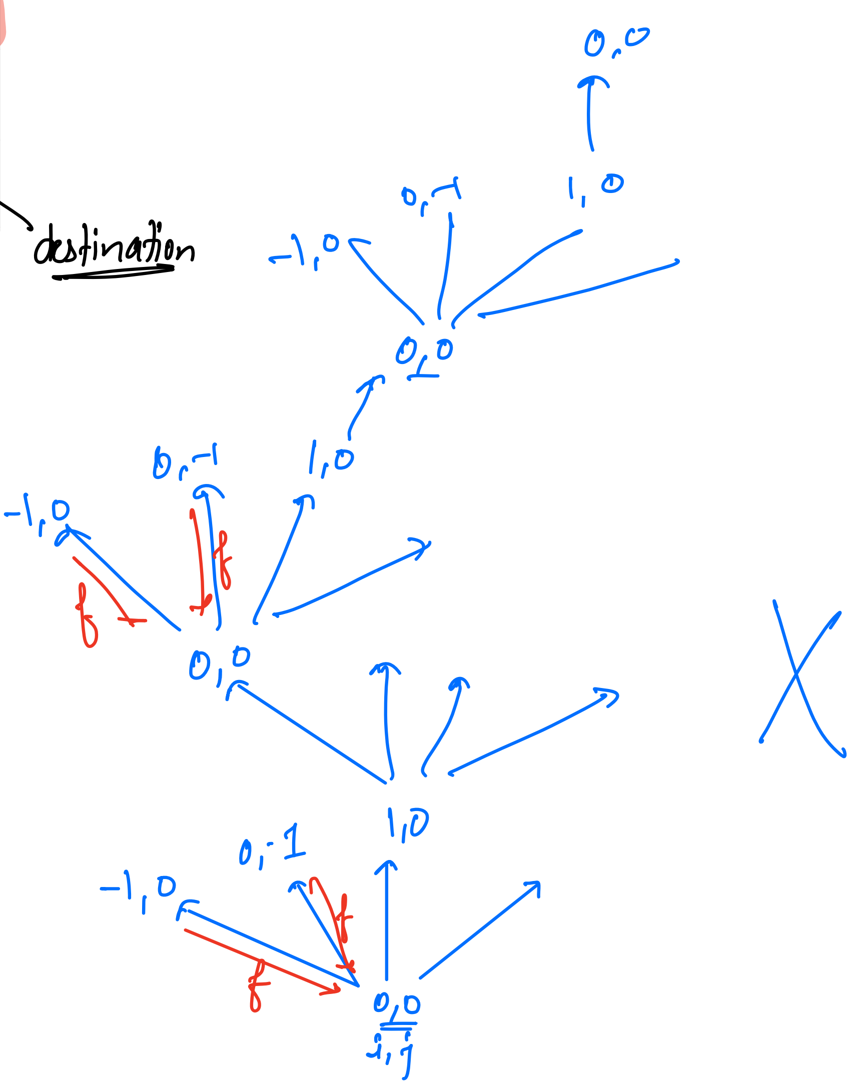
arr[i][j] = 1 ["blocked"]

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

      if( i < 0 || j < 0 || i ≥ N || j ≥ m || arr[i][j] == 1 ){
              return false;
      }
    return ( check[ arr[][], i-1, j) || check( arr[][], i, j-1) ||
             check ( arr[][], i+1, j)  || check( arr[][], i, j+1) );
}
```

src

	0	1	2	3	4	5	6
0				1			
1		1		1		1	
2		1			1		
3			1		1		1
4	1		1				
5				1		1	

destination



∴ We should keep track of visited cells.

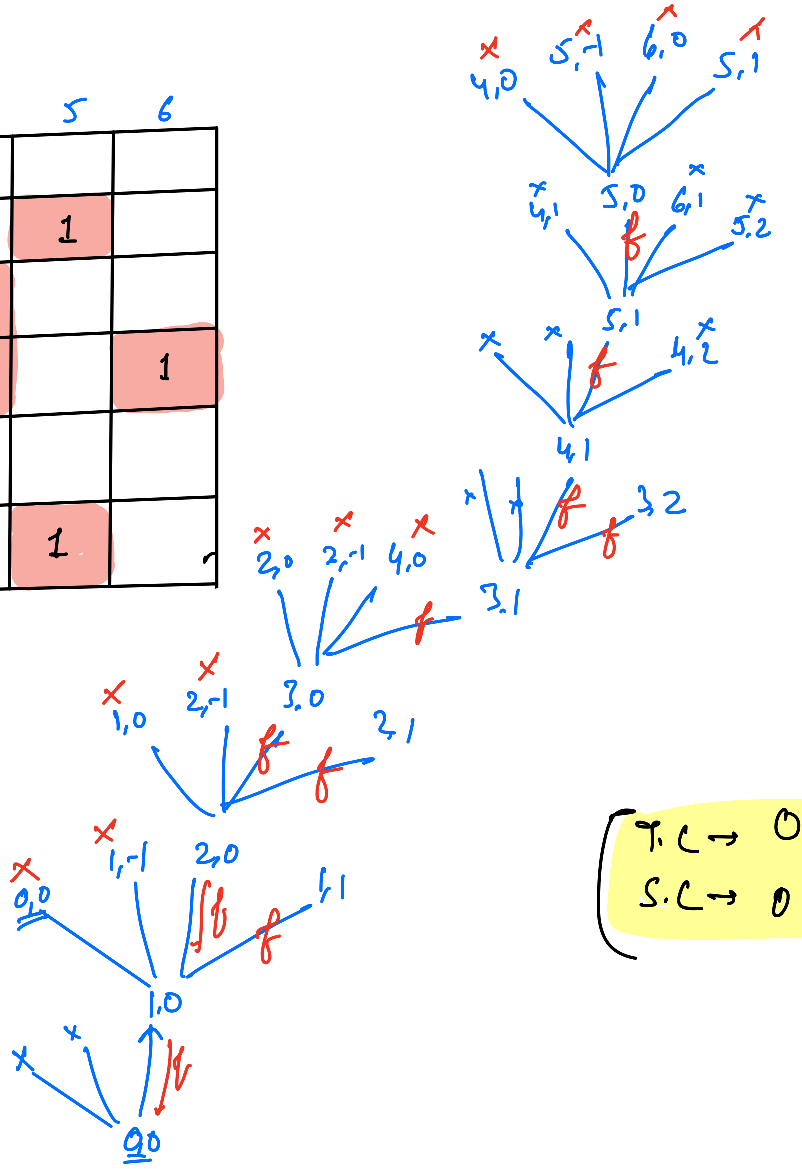
boolean [N][M] → true (cell is visited)
→ false (cell is not visited)

arr [N][M] → 0 → empty
→ 1 → blocked
→ 2 → visited.

final-code.

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

    if( i<0 || j<0 || i≥N || j≥m || arr[i][j] ==1 || arr[i][j]==2){
        return false;
    }
    arr[i][j] = 2 ;     [marking it visited]

    return ( check( arr[][], i-1, j) || check( arr[][], i, j-1) ||
             check (arr[][], i+1, j)  || check(arr[][], i, j+1) );
}
```

	0	1	2	3	4	5	6
0	2			1			
1	2	1		1		1	
2	2	1			1		
3	2	2	1		1		1
4	1	2	1				
5	2	2	1	1		1	



T.C → O(N*m)
S.C → O(N*m)

Permutations -1

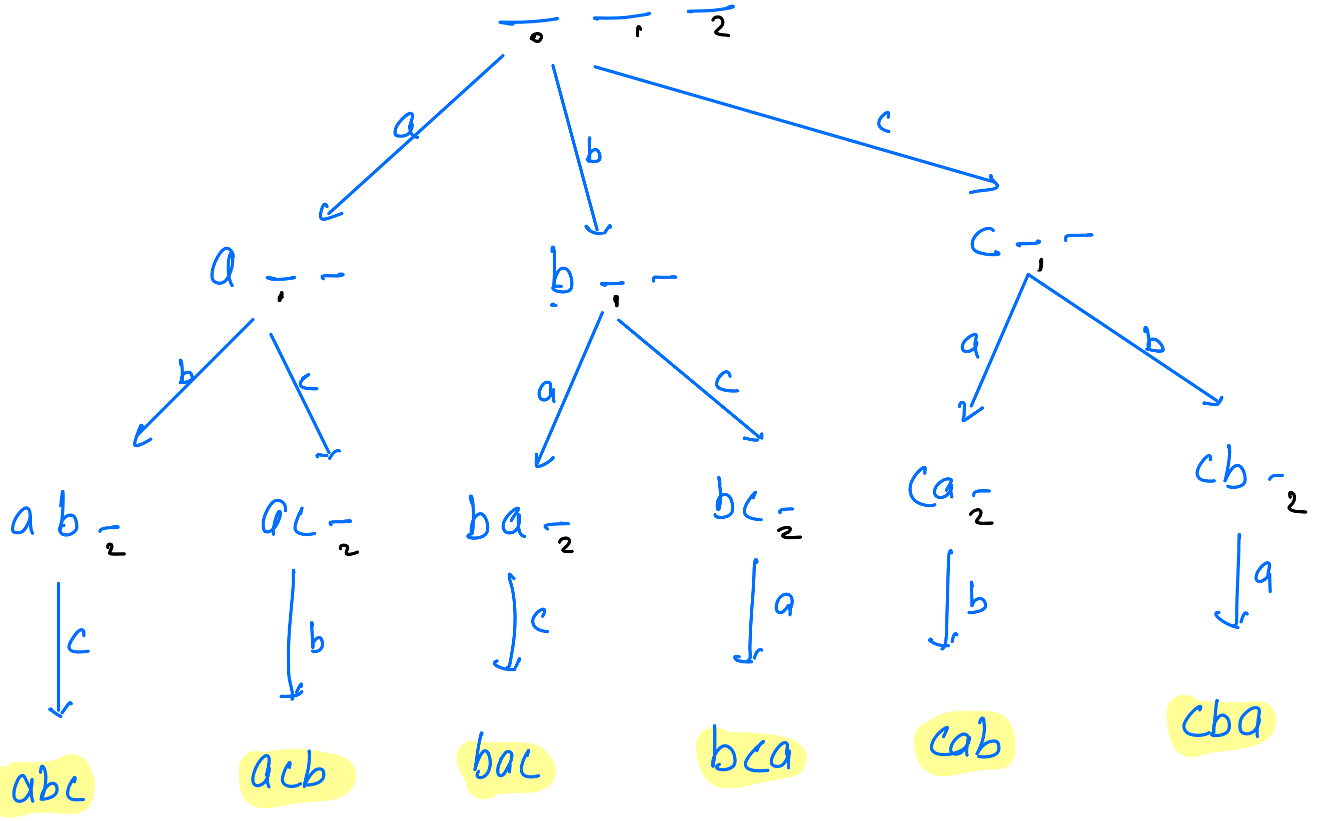
Given a character array with distinct elements, print all permutations of it without modifying it.

A → [a b c]

o/p. → [abc acb bac
bca cab cba]

$\underline{3} * \underline{2} * \underline{1} = 6.$

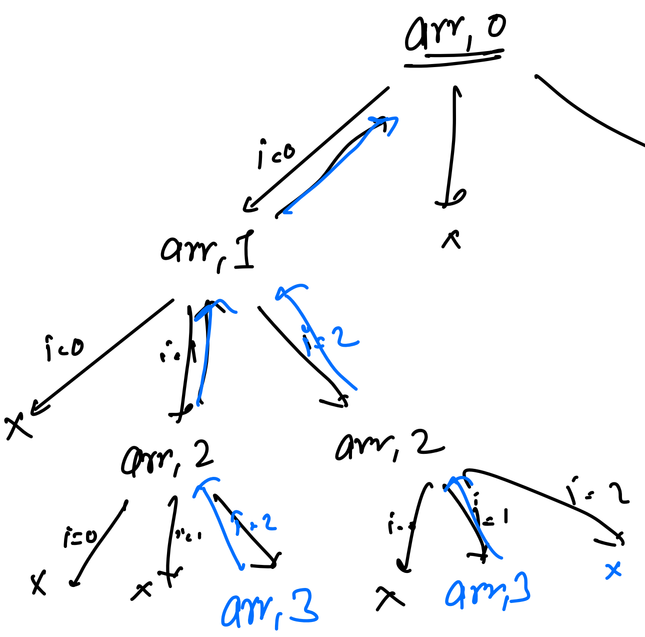
$\underline{N} * \underline{N-1} * \underline{N-2} * \_\_ \_\_ * \underline{1} = \underline{N!}$



⇒ Keep track of visited / acquired characters.

code→

```
                                              0
                                              ↑
void permutations( char[] arr, int idx , char[] ans, boolean[] visited){

    if( idx == N) { print(ans[]) , return }

    for( I=0; i<N; i++){          /exploring all possibilities

        if( visited[i] == false){     // valid possibilities

            visited[i] = true;                ]  do-change
            ans [idx] = arr[i];               ]

            permutations( arr, idx+1, ans, visited); ] Recursive call

            visited[i] = false;               ]  Undo-change.

        }
    }
}
```



visited→

f	f	f
0	1	2

arr[]→

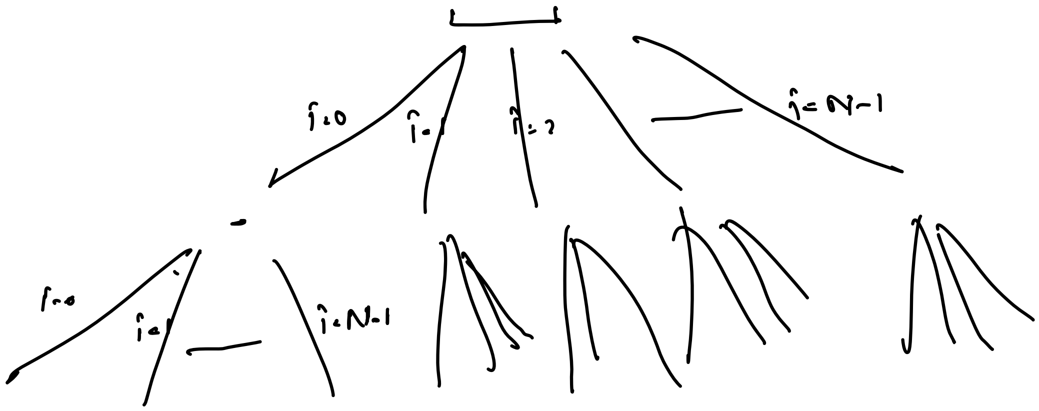
a	b	c
0	1	2

ans[]→

a	~~b~~	~~c~~
0	1	2
	c	b

T.C→ O(N*N!)

S.C→ O(N)



→ $\underline{1}$.

→ N

→ $N*(N-1)$

→ $N*(N-1)*(N-2)$

→ $N*(N-1)*(N-2)*(N-3)$

|

|

→ $\underline{N!}$

Permutations - 2

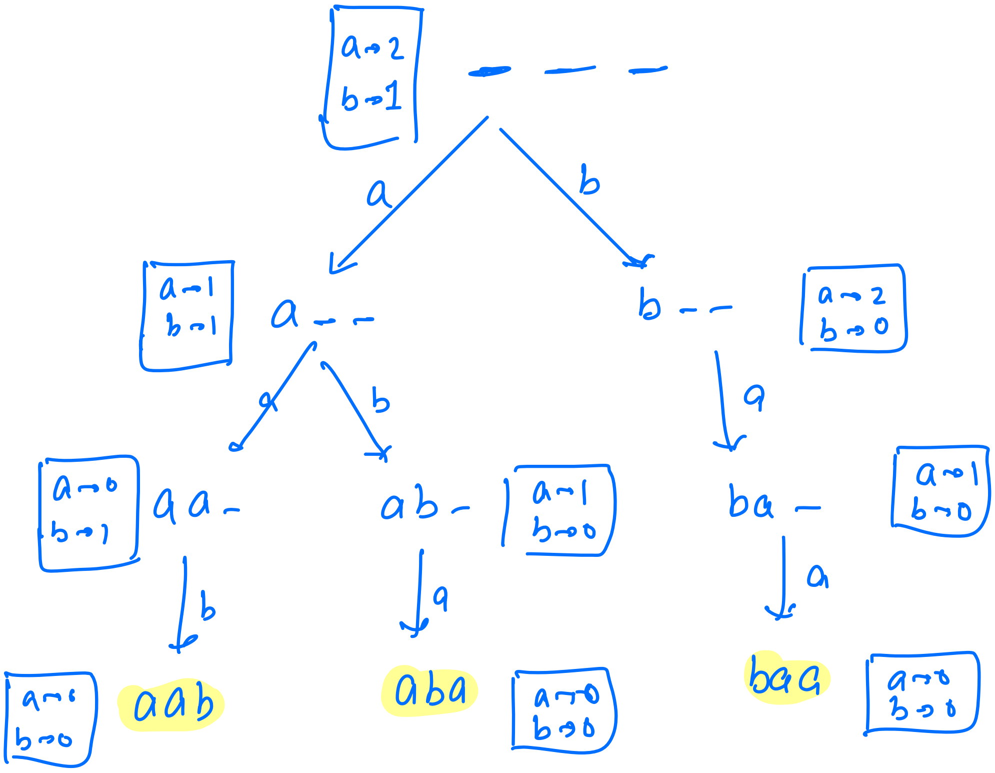
Print all unique permutations of the given char array.

str → [a b a]   o/p → [aab , aba , baa]

MISSISSIPPI → $\frac{11!}{4!\ 4!\ 2!}$ — $\frac{N!}{f(a)!\ f(b)!\ f(c)!\ ——\ f(z)!}$

idea-1 → insert all permutation in hashset.

But SC → O(N!)



code.→

```
                                                     0
                                                     ↑
void permutations2 ( farr[26] , N , ans [N] , idx ) {
        if ( idx == N) { print(ans) , return } ;

        for( i=0; i < 26 ; i++){   // exploring all possibilities
                if ( farr [i] > 0) {   // valid possibilities
                        ans [idx ] = (char) (i + 'a') ;  ]  do change
                        farr[i] -= 1 ;                   ]
                        permutations2 ( farr[], N, ans[], idx+1); //rec. call
                        farr [i] += 1 ;        ]   undo - change
                }
        }
}
```

T.C → O(N!)

S.C → O(N)

Subset.

arr[] → [10, 20, 30]

[-]

[10]

[10, 20]

[10, 20, 30]

[10, 30]

[20]

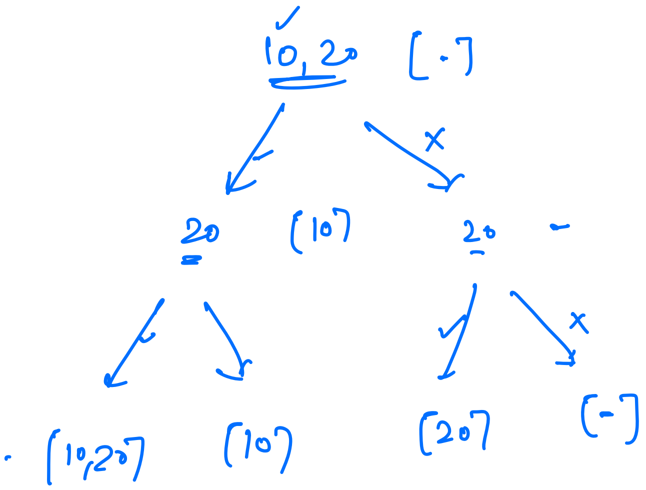
[20, 30]

[30]

```
void  subsets ( arr[], idx , list<int> l ){
        if( idx == N) { print(l) , return}

         subsets( arr[], idx+1, l );
         l. insert( arr[idx]);
         subsets ( arr[], idx+1, l);
         l. remove ( l. size() - 1 );
}
```

$T.C \rightarrow O(2^N)$

$S.C \rightarrow O(N)$



P.S.S → 11:00 A.M – 1:00 PM

- → N-Queens
- → Sudoku.

2. days [medium level]