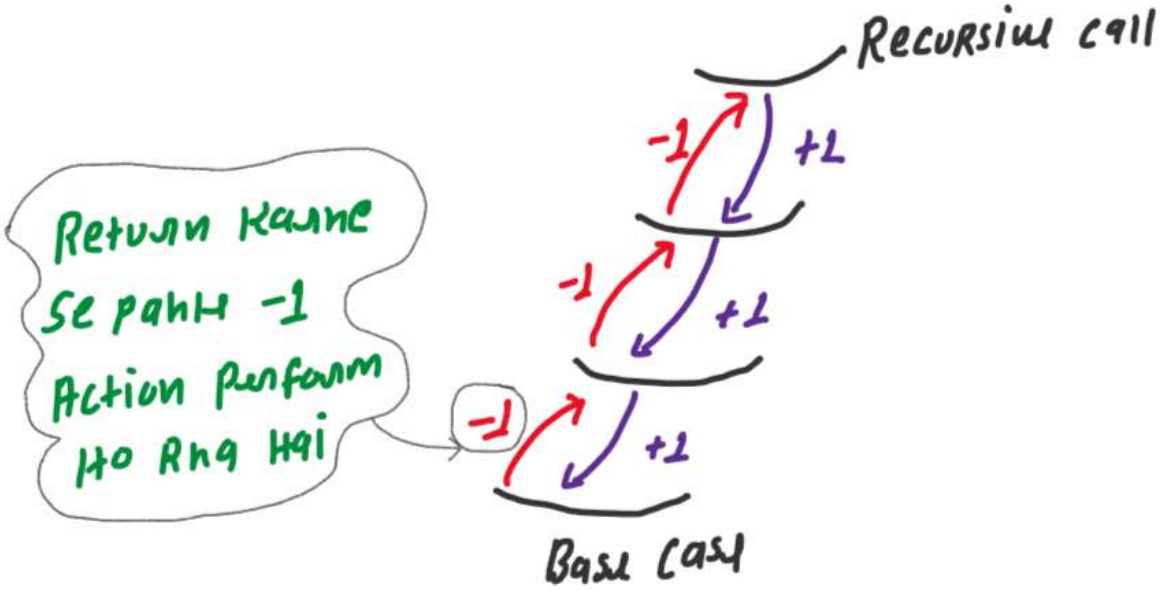
20/10/2023

D&C AND BACKTRACKING - CLASS 2

1. What is backtracking?

Flow of execution of recursive call jab bapas ja rha hoga us time par hum koi operation perform karte hai to ussi operation ko backtracking kahte hai.



2. Permutation of string

Ex1 String → "abc"

abc, acb } a at 1st position

bac, cab } a at 2nd position

bca, cba } a at 3rd position

} Har EK character Har EK position par exist karta hai

size of string = N

No. of Permutation = $N!_0$ ⇒ $3!_0 = 3 \times 2 \times 1 = 6$

Ex2 String → "ab"

ab } a at 1st pos.

ba } a at 2nd pos.

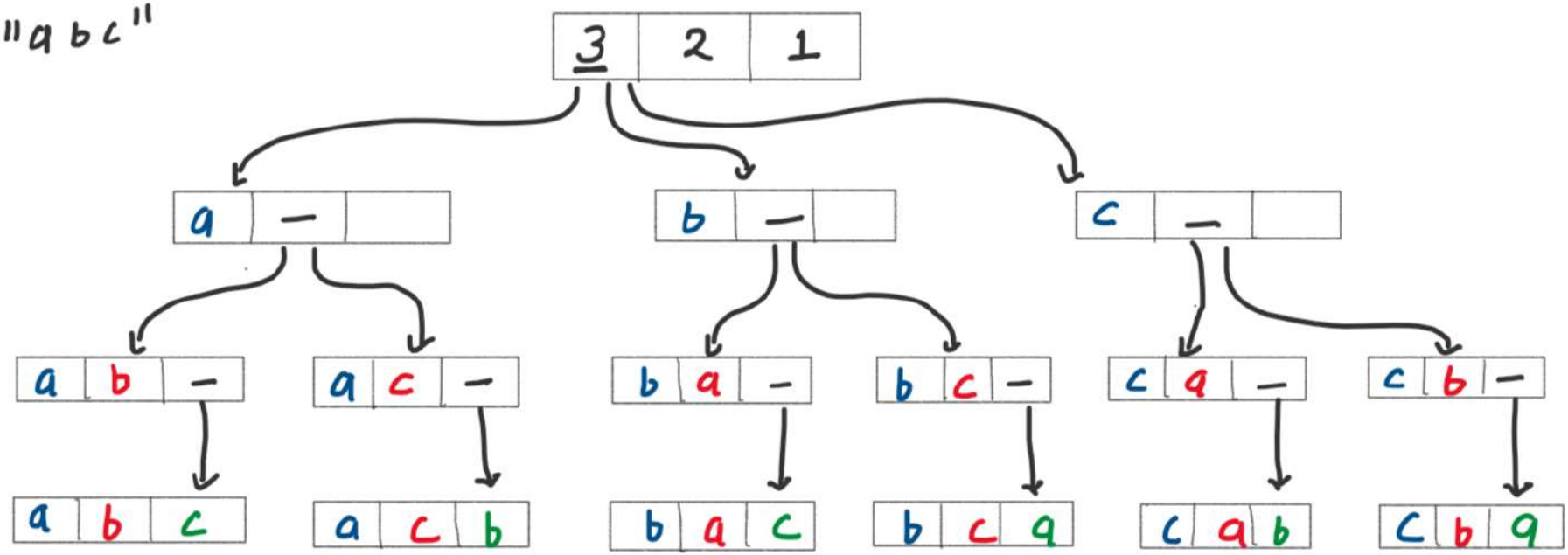
Total Per. = $N!_0$

$= 2!_0$

$= 2 \times 1$

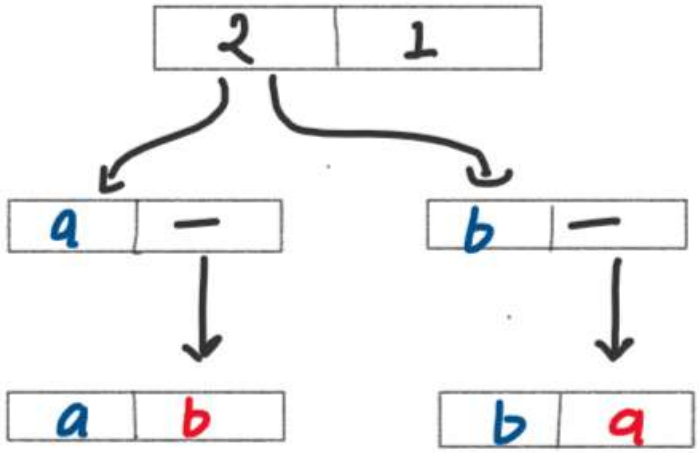
$= 2$

Ex:1 "a b c"



To Permute { = 6

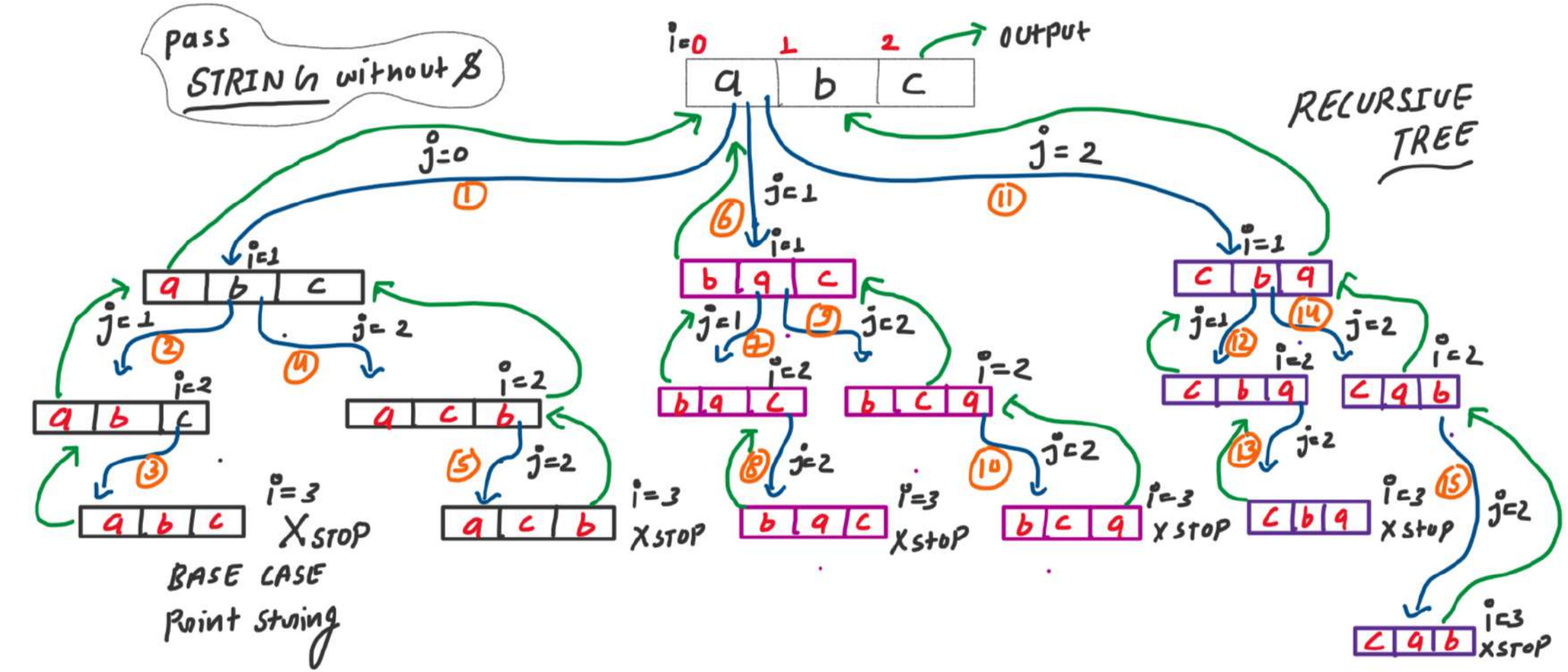
EX:2 "ab"



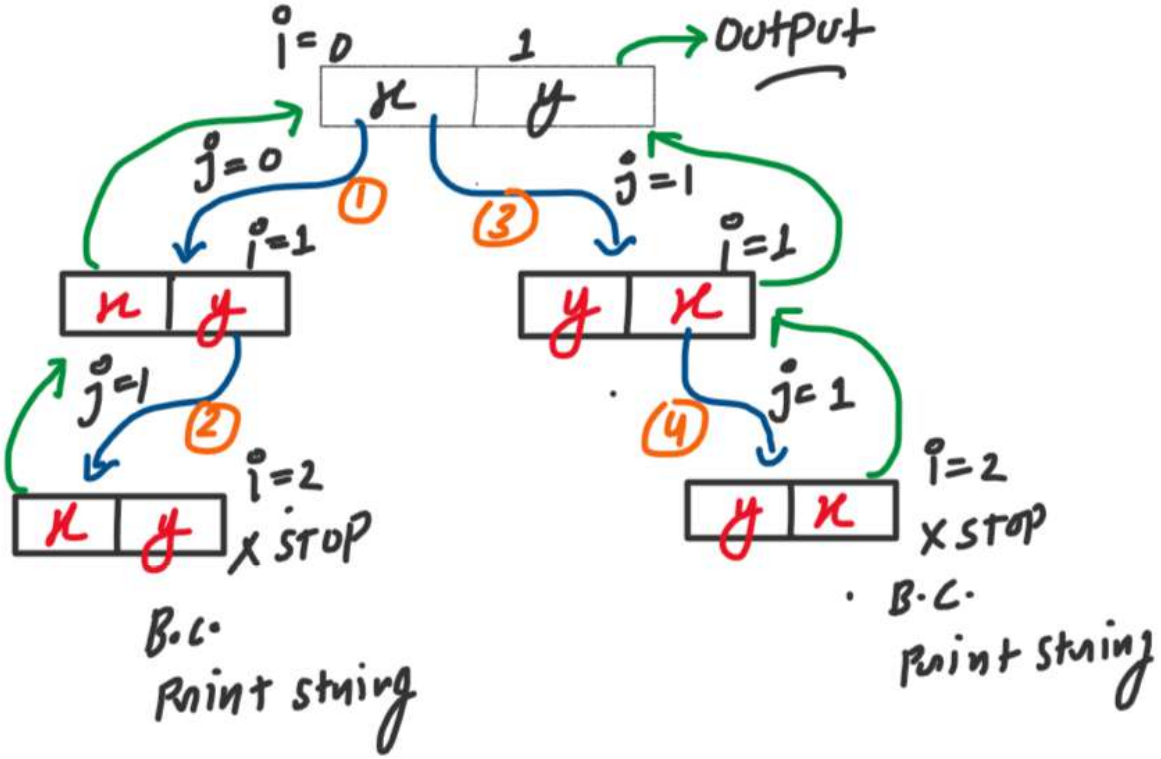
To Permutation{

= 2!

= 2


pass
STRING without S
i=0 1 2
a b c
output
RECURSIVE TREE
j=0
1
j=1
6
j=2
11
i=1
a b c
j=1
2
j=2
4
i=2
a b c
3
i=3
X STOP
BASE CASE
Print string
a c b
5
i=3
X STOP
b a c
7
8
i=3
X STOP
b c a
9
10
i=3
X STOP
c b a
12
14
i=2
c b a
13
i=3
X stop
c a b
15
i=3
X STOP

EX2 pass string without &



```cpp
// ✅Permutation of string (Passing string without reference(&str))
#include<iostream>
#include<string>
using namespace std;

void printPermutation(string str, int index){
    // Base Case
    if(index >= str.length()){
        cout<< str << " ";
        return;
    }

    for(int j=index; j<str.length(); j++){
        // As a first step ek character ko me khud se swap kr doonga
        swap(str[index], str[j]);
        // Ab baki ki character swaping recursion sambhal lega
        printPermutation(str, index+1);
    }
}

int main(){
    string str = "abc";
    int index = 0;
    printPermutation(str, index);
    return 0;
}
```

Take more space

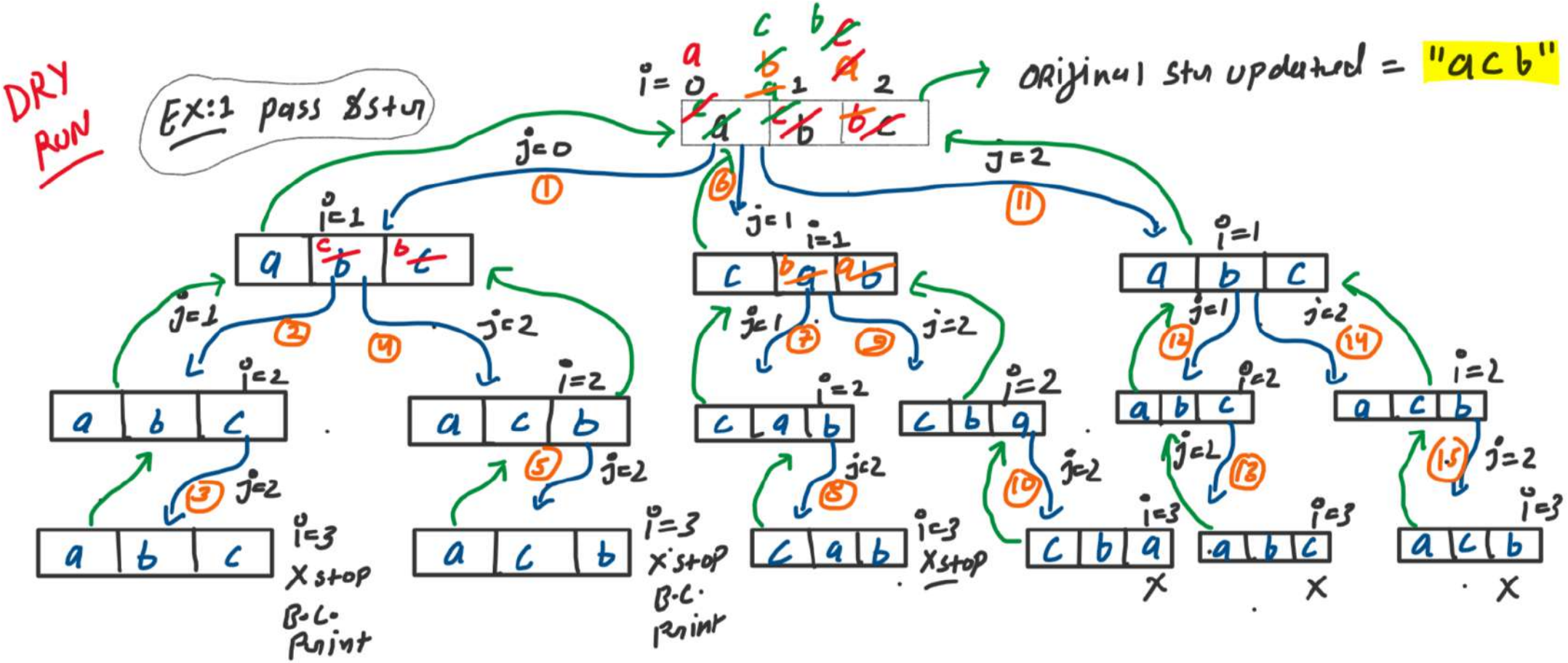
Right o/p

Output: abc acb bac bca cab cba

```cpp
// ✅Permutation of string (Passing string with reference(&str))
#include<iostream>
#include<string>
using namespace std;

void printPermutation(string &str, int index){
    // Base Case
    if(index >= str.length()){
        cout<< str << " ";
        return;
    }

    for(int j=index; j<str.length(); j++){
        // As a first step ek character ko me khud se swap kr doonga
        swap(str[index], str[j]);
        // Ab baki ki character swaping recursion sambhal lega
        printPermutation(str, index+1);
    }
}

int main(){
    string str = "abc";
    int index = 0;
    printPermutation(str, index);
    return 0;
}

/*
Example 1:
Input: str = "abc"
Expected Output: abc acb bac bca cab cba
Your Output: abc acb cab cba abc acb
*/
```

Take less space

Wrong o/p


DRY RUN
EX:1 pass "abc"
i=0
j=0
j=1
j=2
Original str updated = "acb"
i=1
i=2
i=3
X stop
B.C.
Print
X

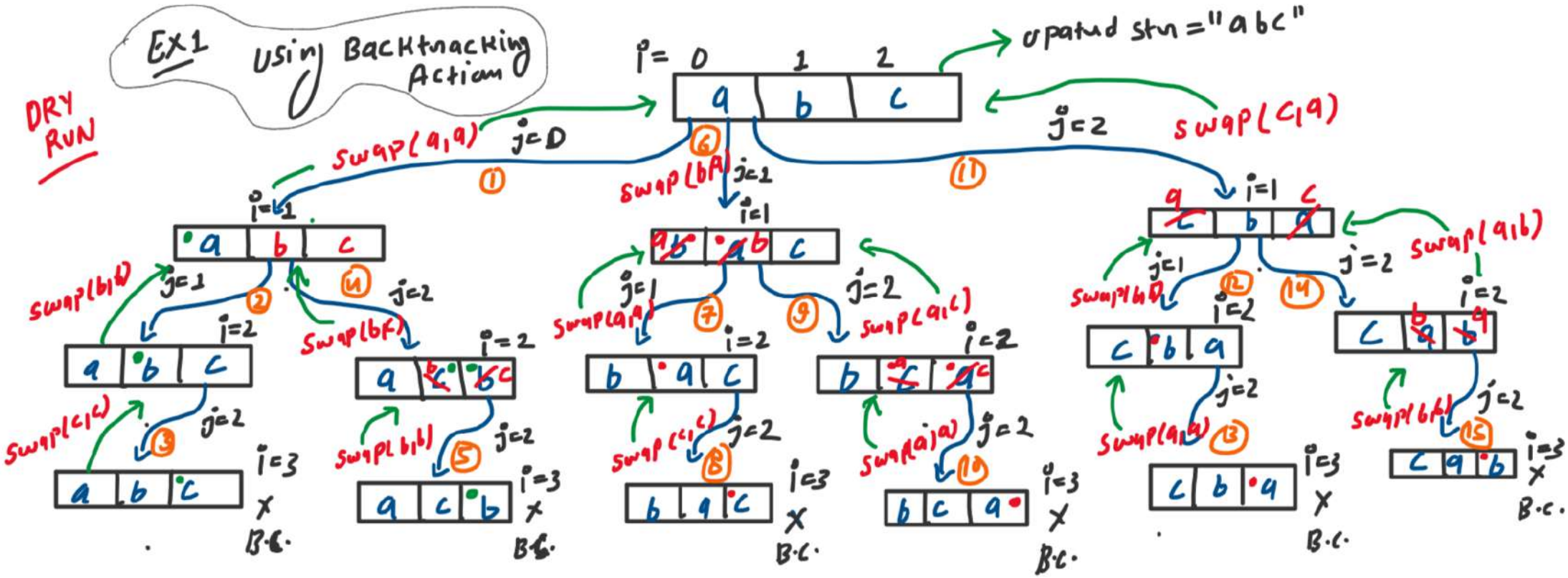
```cpp
// ✅Permutation of string (Passing string with reference(&str) and using Backtracking Action)
#include<iostream>
#include<string>
using namespace std;

void printPermutation(string &str, int index){
    // Base Case
    if(index >= str.length()){
        cout<< str << " ";
        return;
    }

    for(int j=index; j<str.length(); j++){
        // As a first step ek character ko me khud se swap kr doonga
        swap(str[index], str[j]);
        // Ab baki ki character swaping recursion sambhal lega
        printPermutation(str, index+1);
        // Backtracking action
        swap(str[index], str[j]);
    }
}

int main(){
    string str = "abc";
    int index = 0;
    printPermutation(str, index);
    return 0;
}

/*
Example 1:
Input: str = "abc"
Expected Output: abc acb bac bca cba cab
Your Output: abc acb bac bca cba cab
*/
```

It's a Backtracking

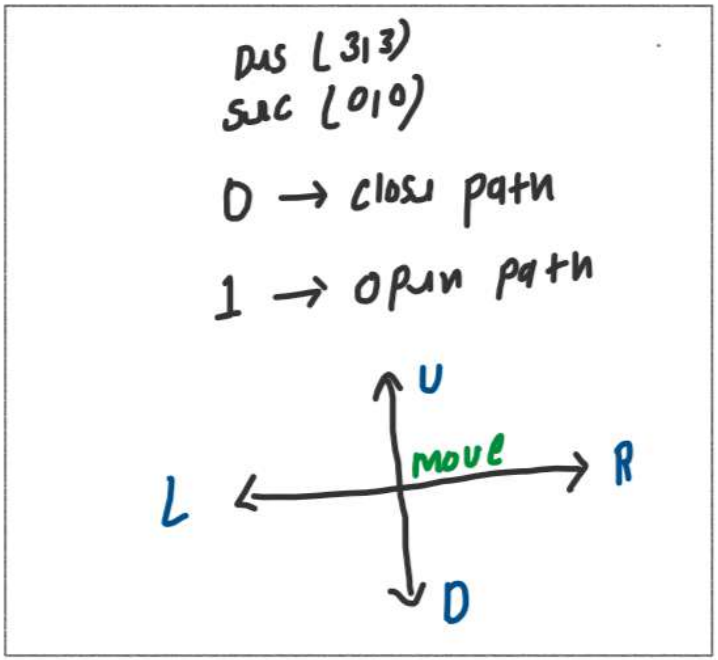
takes less space & Right output

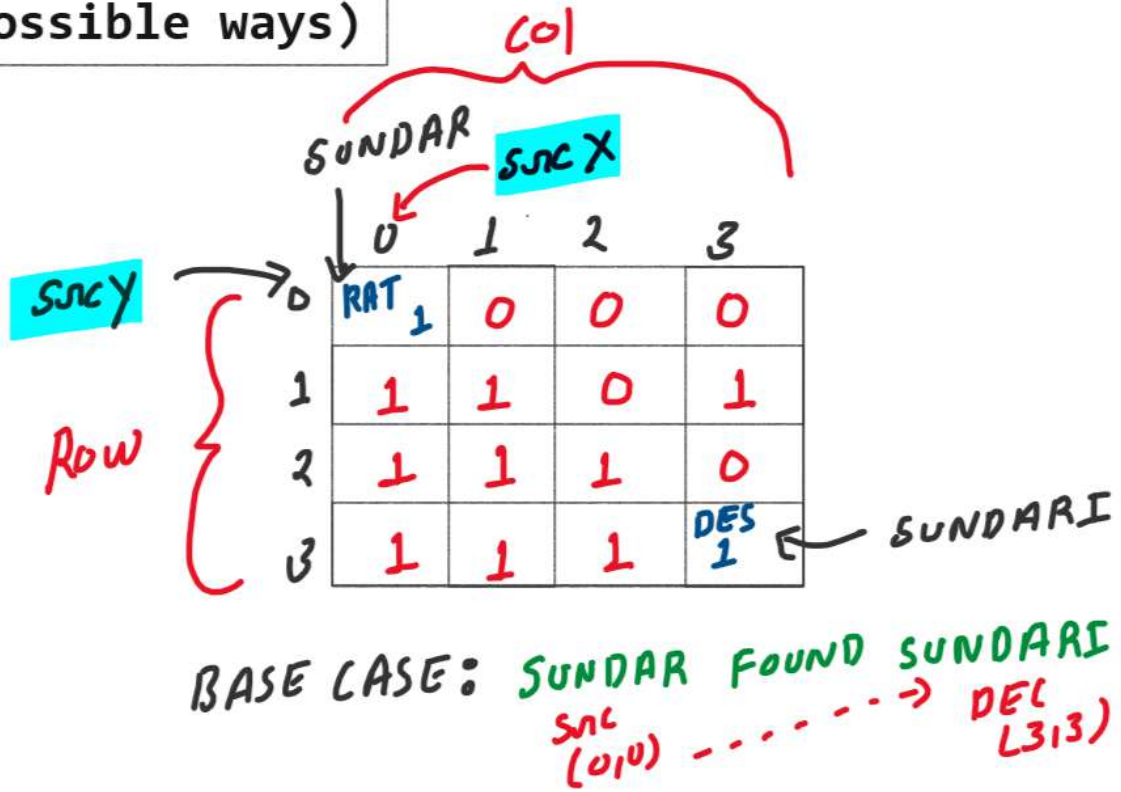
Time = ?

Space = ?

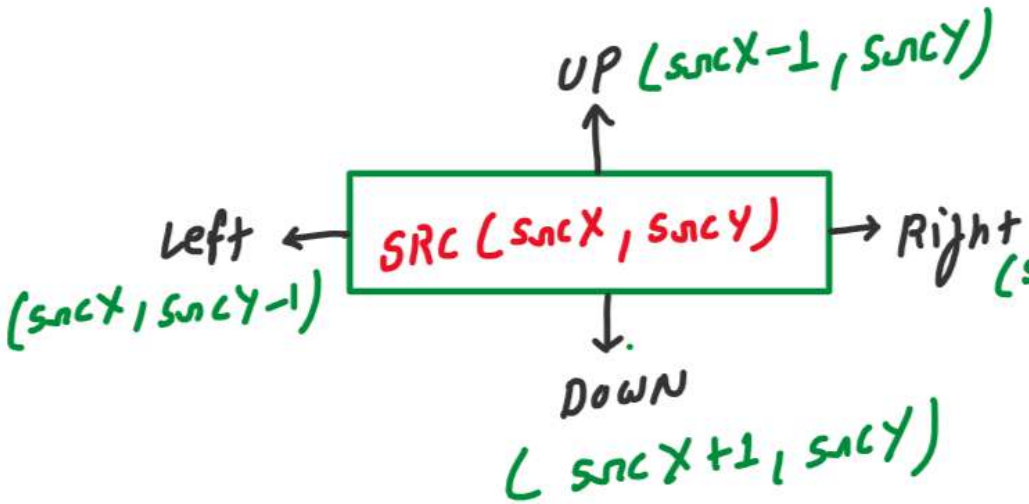

EX1 Using Backtracking Action
DRY RUN
i= 0 1 2
a b c
updated str = "a b c"
swap(a,a) j=0
swap(c,a)
j=2
swap(b,b) j=1
swap(b,c) j=2
swap(a,a) j=1
swap(a,c) j=2
swap(a,b) j=1
swap(a,c) j=2
swap(b,b) j=2
swap(b,c) j=2
swap(c,c) j=2
swap(a,a) j=2
swap(c,b) j=2
swap(b,b) j=2
swap(b,c) j=2
swap(a,b) j=2
B.C.

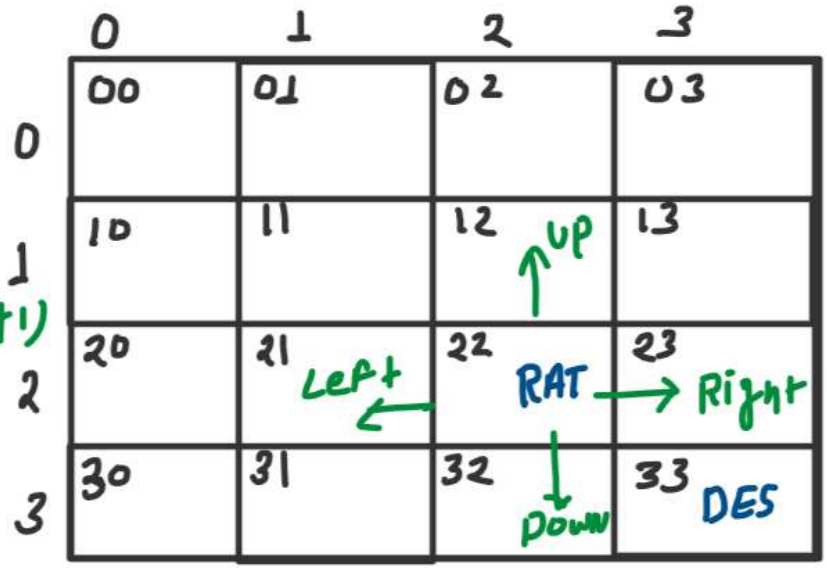
3. Rat in a maze (Print all possible ways)



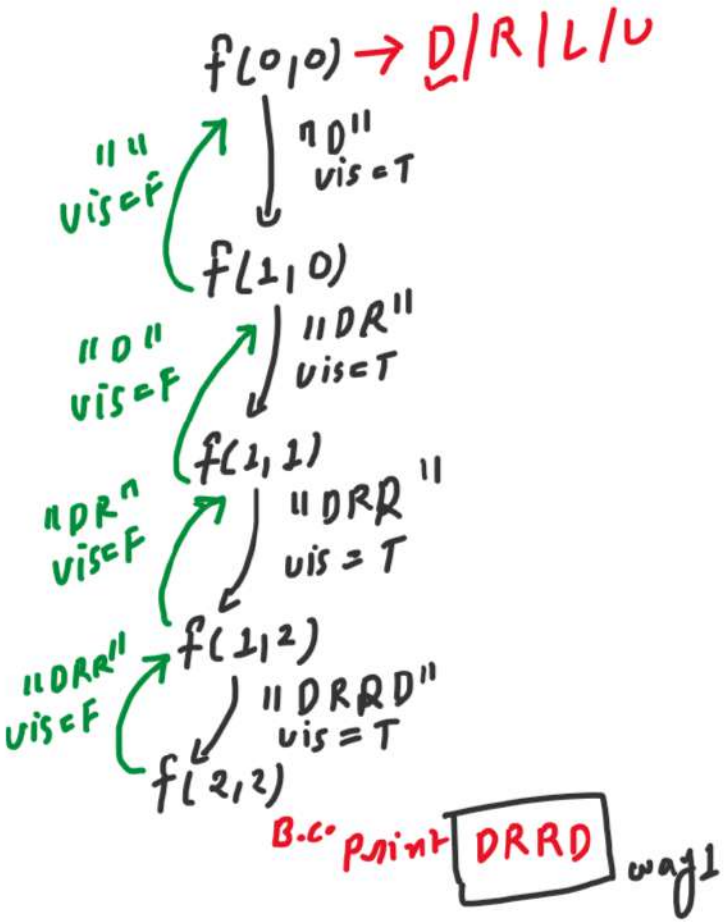


src (2,2) ⇒ RAT


UP (srcX-1, srcY)
Left
SRC (srcX, srcY)
Right
(srcX, srcY-1)
(srcX, srcY+1)
DOWN
(srcX+1, srcY)


0
1
2
3
00
01
02
03
10
11
12
13
20
21
22
23
30
31
32
33
UP
Left
RAT
Right
DOWN
DES

EXAMPLE:01 **DRY RUN**



Condition

1) visited
2) Index bound
3) Closed path

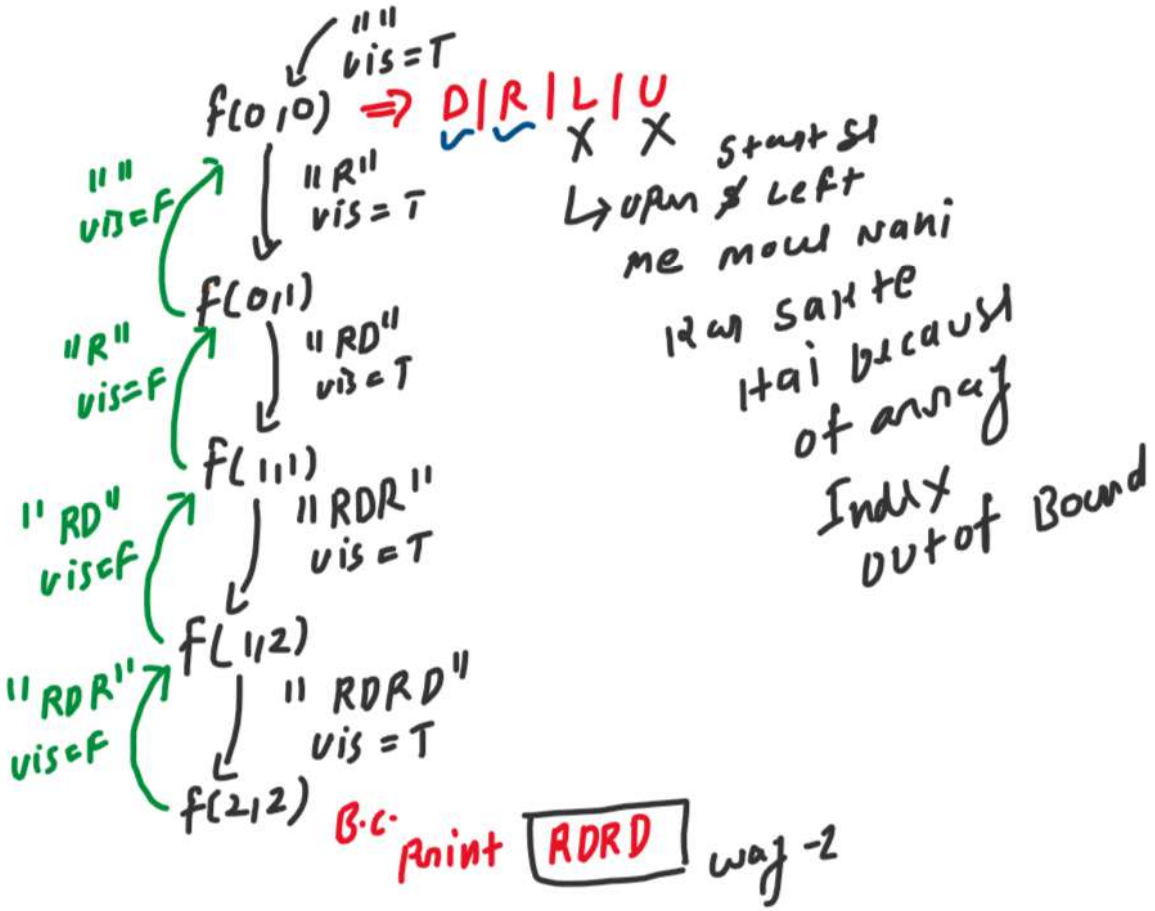
MAZE

	0	1	2
0	1	1	0
1	1	1	1
2	0	0	1

visited

	0	1	2
0	F T	F	F
1	F F T	F F T	F F T
2	F	F	F F T

src (0,0)
dis (2,2)
MOVE ⇒ D/R/L/U



Conditions

1) visited
2) Index bound
3) Closed path

MAZE

	0	1	2
0	1 →	1	0
1	1	1 →	1 ↓
2	0	0	1

visited

	0	1	2
0	F ~~T~~	F ~~F~~	F
1	F	F ~~F~~	F ~~F~~
2	F	F	F ~~F~~ ~~T~~

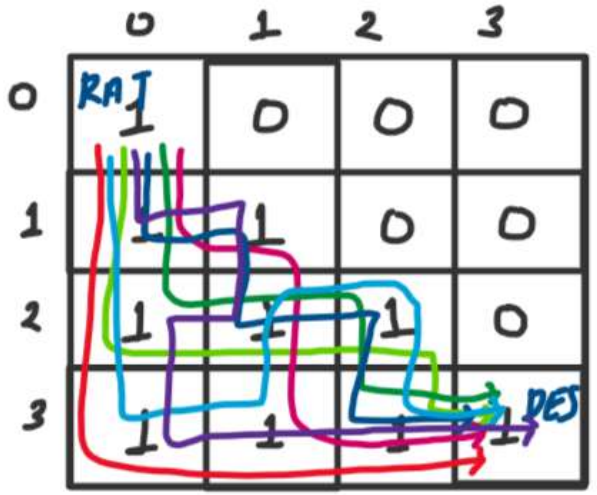
src (0,0)
dest (2,2)
Move ⇒ D|R|L|U

Example : 02

1) D D D R R R
2) D D R R D R
3) D R D D R R
4) D D R R D R
5) D R D R D R
6) D D D U R D R
7) D R D L D R R R

→ vis[0][0] = T
→ SRC (0,0)
→ dis (3,3)
→ move
D | R | L | U

Condition ↴
1) Index Bound
2) Visited
3) Close path



	0	1	2	3
0	F	F	F	F
1	F	F	F	F
2	F	F	F	F
3	F	F	F	F

```cpp
// ✅Rat in a maze (Print all possible ways)
#include <iostream>
#include<vector>
#include<string>
using namespace std;

// isSafe() Function that will handle all the below mentioned possibilities:
// Condition 1: Out of bound
// Condition 2: Path Closed
// Condition 3: Check if position is already visited
bool isSafe(int newx, int newy, int maze[][4], int row, int col,  vector<vector<bool > > &visited) {
    if(
        (newx >=0 && newx <row) && (newy >=0 && newy < col) &&
        (maze[newx][newy] == 1) &&
        (visited[newx][newy] == false)
        ) {
        return true;
        }
    else {
        return false;
    }
}

void printAllPath(int maze[][4], int row, int col, int srcx, int srcy, string &output, vector<vector<bool > > &visited)
{

    // Base Case --> Destination Coordinates are [row-1] and [col-1]
    if(srcx == row-1 && srcy == col-1) {
        // Reached to destination
        cout << output << endl;
        return;
    }

    //1 case hum solve karenge and baaki ka recursion sambhal lega for URDL
    // ✅UP (Rat/src move kab kar skta hu to yeh sab isSafe() method decide karega)
    int newx = srcx-1;
    int newy = srcy;
    if(isSafe(newx, newy,maze,row,col,visited )) {
        // Mark visited
        visited[newx][newy] = true;
        // Call recursion
        output.push_back('U');
        printAllPath(maze, row, col, newx, newy, output , visited );
        // Backtracking
        output.pop_back();
        visited[newx][newy] = false;
    }


    // ✅RIGHT
    newx = srcx;
    newy = srcy+1;
    if(isSafe(newx, newy,maze,row,col,visited )) {
        // Mark visited
        visited[newx][newy] = true;
        // Call recursion
        output.push_back('R');
        printAllPath(maze, row, col, newx, newy, output , visited );
        // Backtracking
        output.pop_back();
        visited[newx][newy] = false;
    }

    // ✅DOWN
    newx = srcx+1;
    newy = srcy;
    if(isSafe(newx, newy,maze,row,col,visited )) {
        // Mark visited
        visited[newx][newy] = true;
        // Call recursion
        output.push_back('D');
        printAllPath(maze, row, col, newx, newy, output , visited );
        // Backtracking
        output.pop_back();
        visited[newx][newy] = false;
    }


    // ✅LEFT
    newx = srcx;
    newy = srcy-1;
    if(isSafe(newx, newy,maze,row,col,visited )) {
        // Mark visited
        visited[newx][newy] = true;
        // Call recursion
        output.push_back('L');
        printAllPath(maze, row, col, newx, newy, output , visited );
        // Backtracking
        output.pop_back();
        visited[newx][newy] = false;
    }

}

int main() {

    int maze[4][4] = {
    {1,0,0,0},
    {1,1,0,0},
    {1,1,1,0},
    {1,1,1,1}
    };
    int row = 4;
    int col = 4;

    int srcx = 0;
    int srcy = 0;
    string output = "";

    // Create visited 2D array
    vector<vector<bool > > visited(row, vector<bool>(col, false));

    if(maze[0][0] == 0) {
        // src position is closed, that means RAT connot move
        cout << "No Path Exists" << endl;
    }
    else {
        visited[srcx][srcy] = true;
        printAllPath(maze, row, col, srcx, srcy, output, visited);
    }
    return 0;
}

/*
Example 01:
input: maze =
{
    {1,1,0},
    {1,1,1},
    {0,0,1}
}
output: total possible way to achieve destination
WAY 1 --> RDRD
WAY 2 --> DRRD

Example 02:
input: maze =
{
    {1,0,0,0},
    {1,1,0,0},
    {1,1,1,0},
    {1,1,1,1}
}
output: total possible way to achieve destination
WAY 1 --> DRDRDR
WAY 2 --> DRDDRR
WAY 3 --> DRDLDRRR
WAY 4 --> DDRRDR
WAY 5 --> DDRDRR
WAY 6 --> DDDRURDR
WAY 7 --> DDDRRR
*/
```