Recursive Backtracking

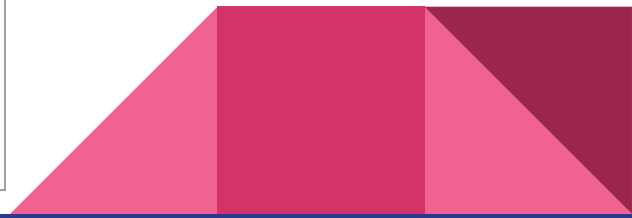
Part 1

Warm-up: Problem

How would you write a recursive method **printBinary** that receives an integer number of digits and prints all binary numbers that have exactly that many digits, in ascending order, one per line. Another parameter is a string that should be printed when the base case is reached.

```
printBinary(2, "");
00
01
10
11
```

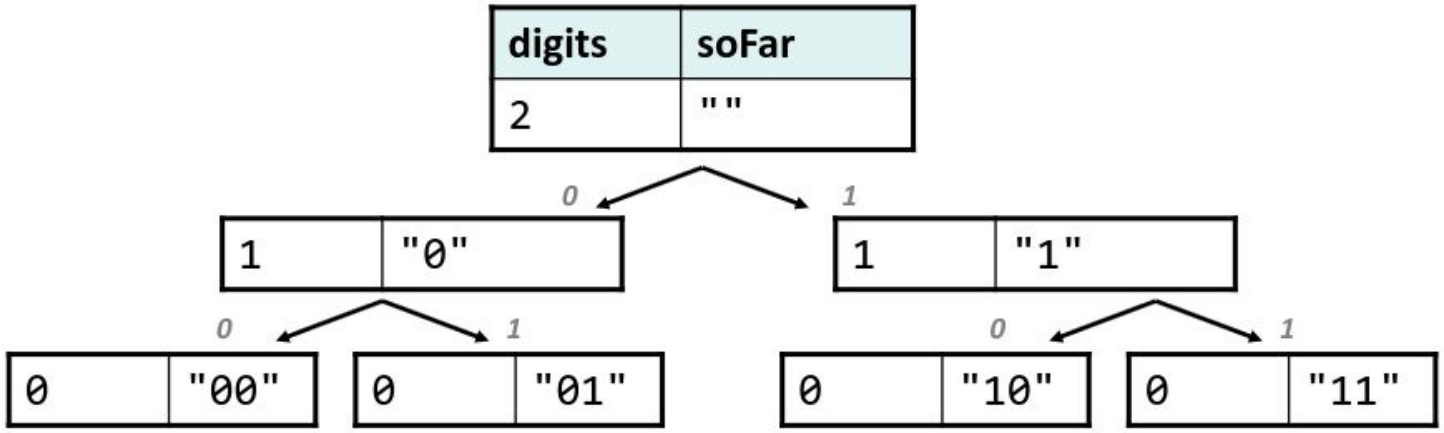
```
printBinary(3, "");
000
001
010
011
100
101
110
111
```



Tree of calls or decision tree

Each call is a choice or decision made by the algorithm:

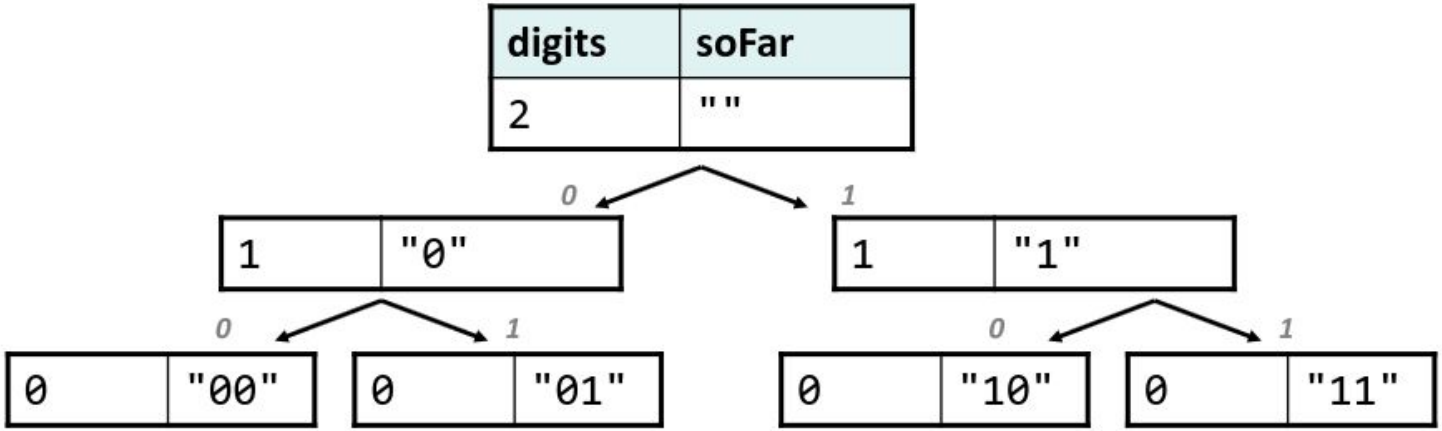
- Should I choose 0 as the next digit?
- Should I choose 1 as the next digit?



Tree of calls or decision tree

printBinary(int digits, String soFar)

printBinary(2, "");



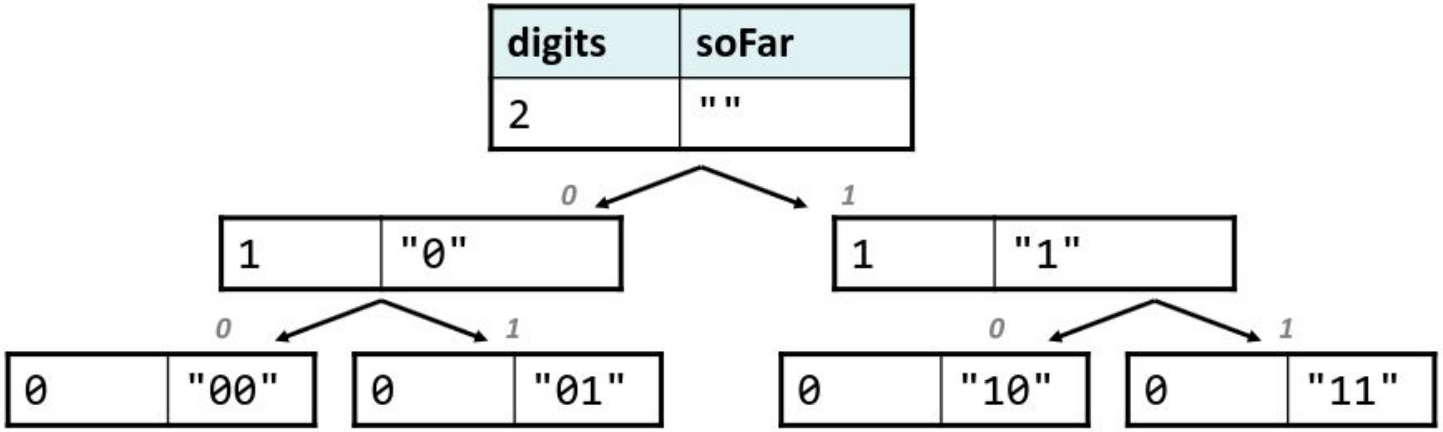
What is the base case?

What is the recursive case?

The base case

Base Case: printBinary(0)

- Each call should keep track of the work it has done
- Base case should print the result of the work done by prior calls
- Work done by each call is kept track in some variable(s) - in this case, string soFar.



Java code

```
public static void printBinary(int digits, String soFar){
        if (digits == 0)
                System.out.println(soFar);
        else {
                printBinary(digits - 1, soFar + "0");
                printBinary(digits - 1, soFar + "1");
        }
}
```

Call Stack

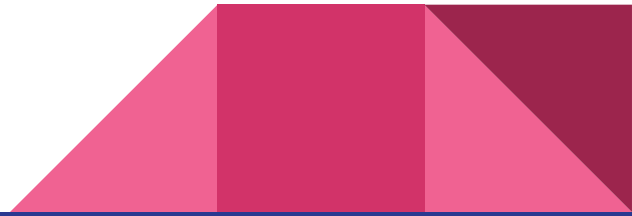
```
printBinary(2, "")
 |
 |--> printBinary(1, "0")
 |    |
 |    |--> printBinary(0, "00")  ->  prints "00"
 |    |
 |    |--> printBinary(0, "01")  ->  prints "01"
 |
 |--> printBinary(1, "1")
      |
      |--> printBinary(0, "10")  ->  prints "10"
      |
      |--> printBinary(0, "11")  ->  prints "11"
```

What is recursive backtracking?

Recursive Backtracking: It is a recursive algorithm for finding all the possible solutions by exploring all possible ways.

It could be used in the following situations:

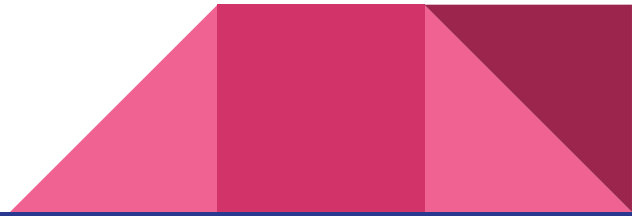
- Determine whether a solution exists
- Find a solution
- Find the best solution
- Count the number of solutions
- Print or find all the solutions



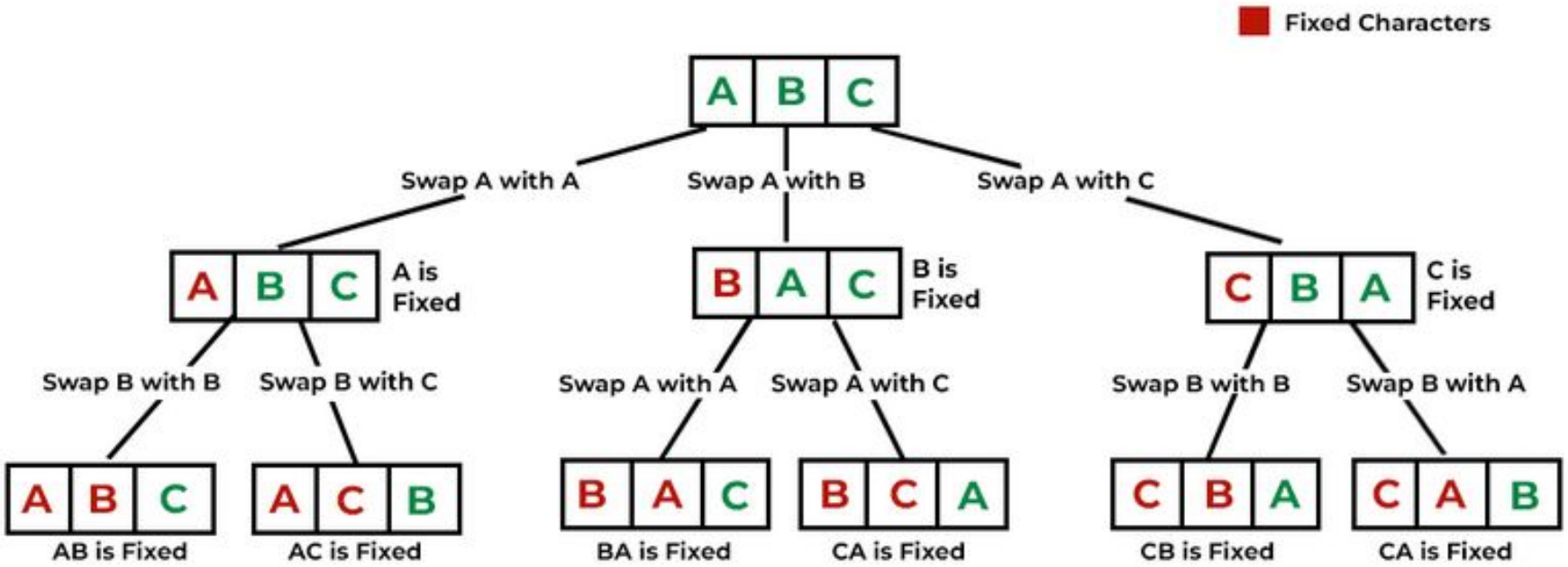
Recursive Backtracking Applications

- Puzzle solving (Sudoku, Crosswords, etc.)

- Game playing (Chess, Solitaire, etc.)

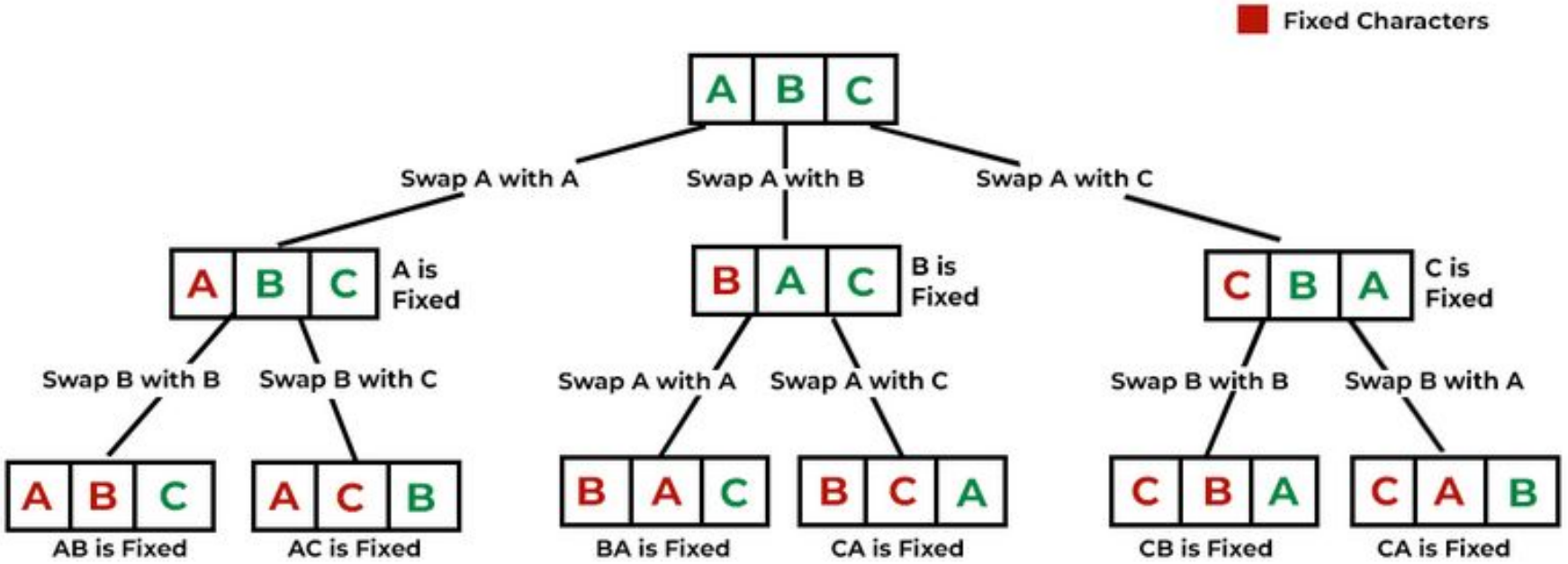


String permutation using backtracking



String permutation using backtracking

```
public static void permute(String str, int l){}
```



String Permutation - Backtracking Algorithm

The backtracking function considers the first index of the given string (fixed character). The function explores all possible swaps at each position.

Let's say the length of the string is N. Current index refers to the fixed character.

Base Case:

If current index reaches N - 1, print str. It means that the current permutation is completed.

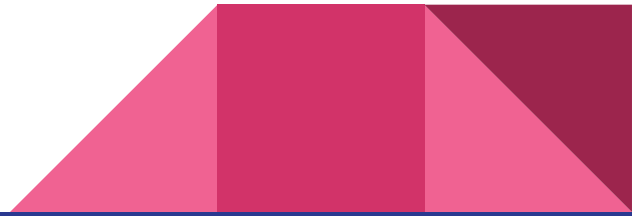
Recursive Case:

Loop through each character from index **l** to the end of the string.
Swap the character at index **l** with the character at index **i (**loop), creating a new string.
Recursively call permute() with the new swapped string and increment **l** to generate further permutations.

-

The Backtracking Checklist

- **Find what choice(s) we have at each step.** What different options are there for the next step?
- For each valid choice:
  - **Make it and explore recursively.** Pass the information for a choice to the next recursive call(s).
  - **Go back after exploring.** Restore everything to the way it was before making this choice.
- **Find the base case(s).** What should we do when we are out of decisions?



Coding Time!!!

Save your work here:

.../APCSA2/apcsa-assignments-spring-YourUsername/classwork/02_03_permutations/PermuteString.java

```
public static void permute(String str, int l) {

}
```

Call the method:

```
String str = "ABC";
permute(str, 0);
```

Expected output:

```
ABC
ACB
BAC
BCA
CBA
CAB
```