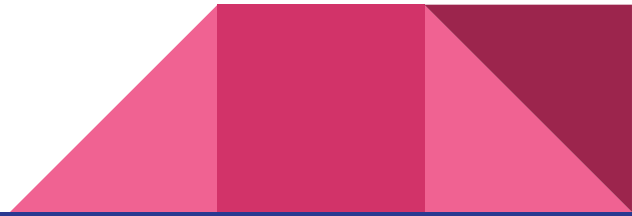
Recursive Backtracking

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.

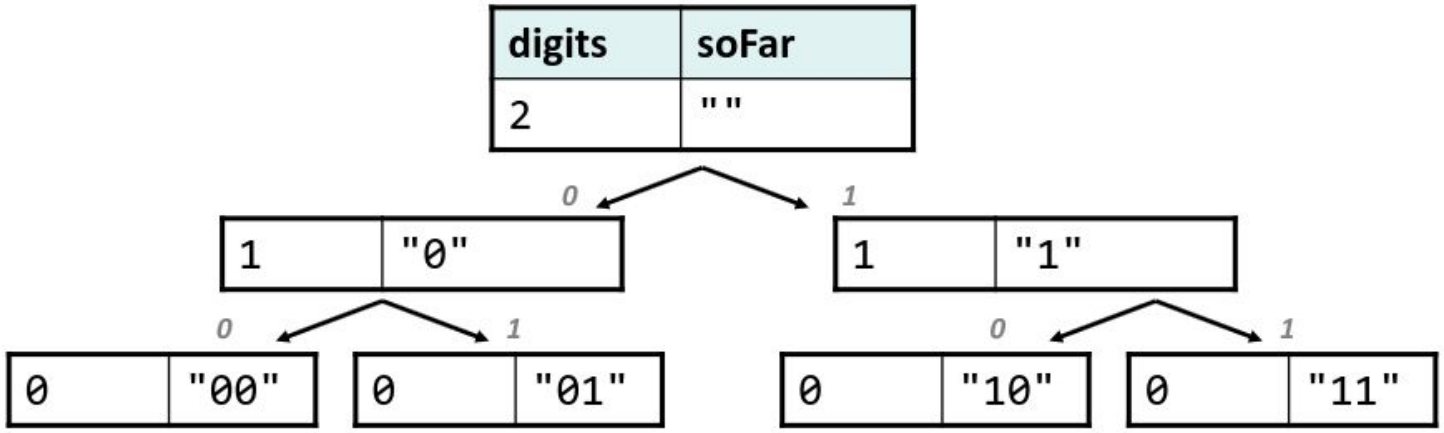
printBinary(2, ""); 00 01 10 11	printBinary(3, ""); 000 001 010 011 100 101 110 111



Tree of calls or decision tree

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

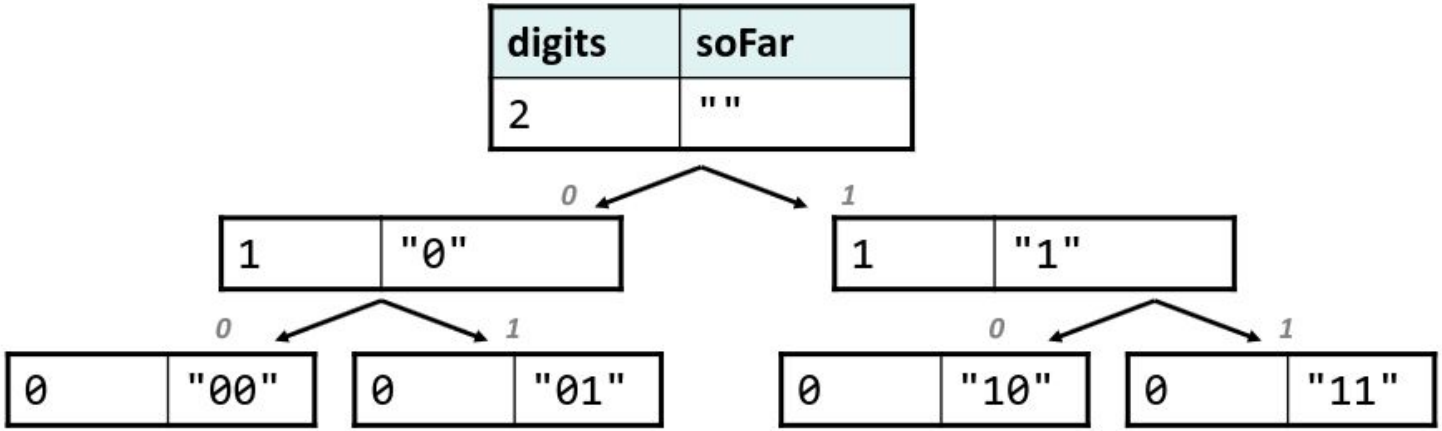
- Should I choose 0 as the next digit?
- Should I choose 1 as the next digit?



Tree of calls or decision tree

printBinary(int digits, String soFar)

printBinary(2, "");



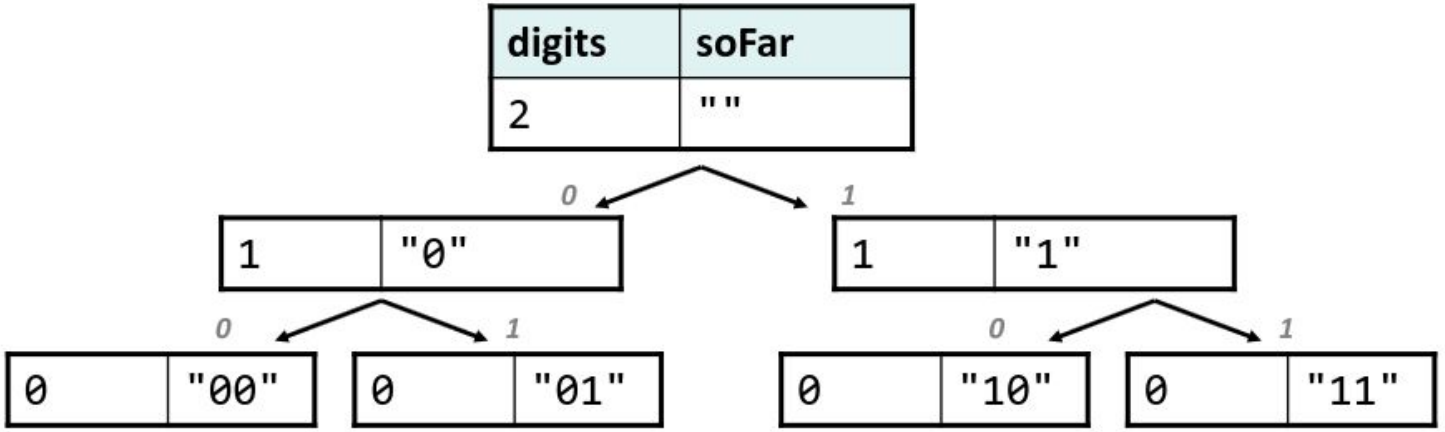
What is the base case?

What is the recursive case?

The base case

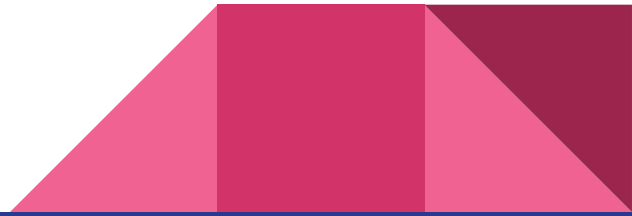
Base Case: printBinary(2) => printBinary(1) => printBinary(0)

- 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");
        }
}
```

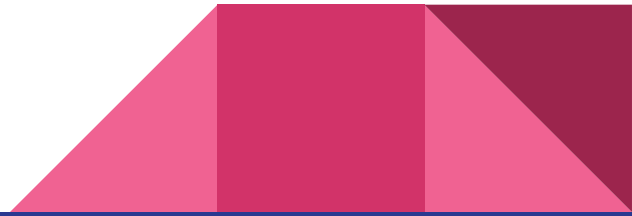


What is recursive backtracking?

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:

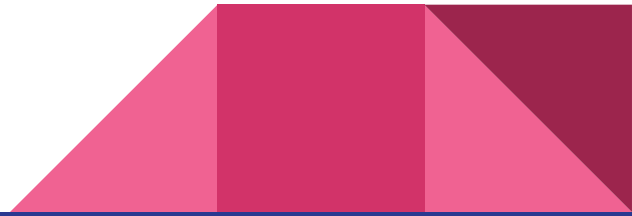
- Determine whether a solution exists
- Find a solution
- Find the best solution
- Count the number of solutions
- Print or find all the solutions



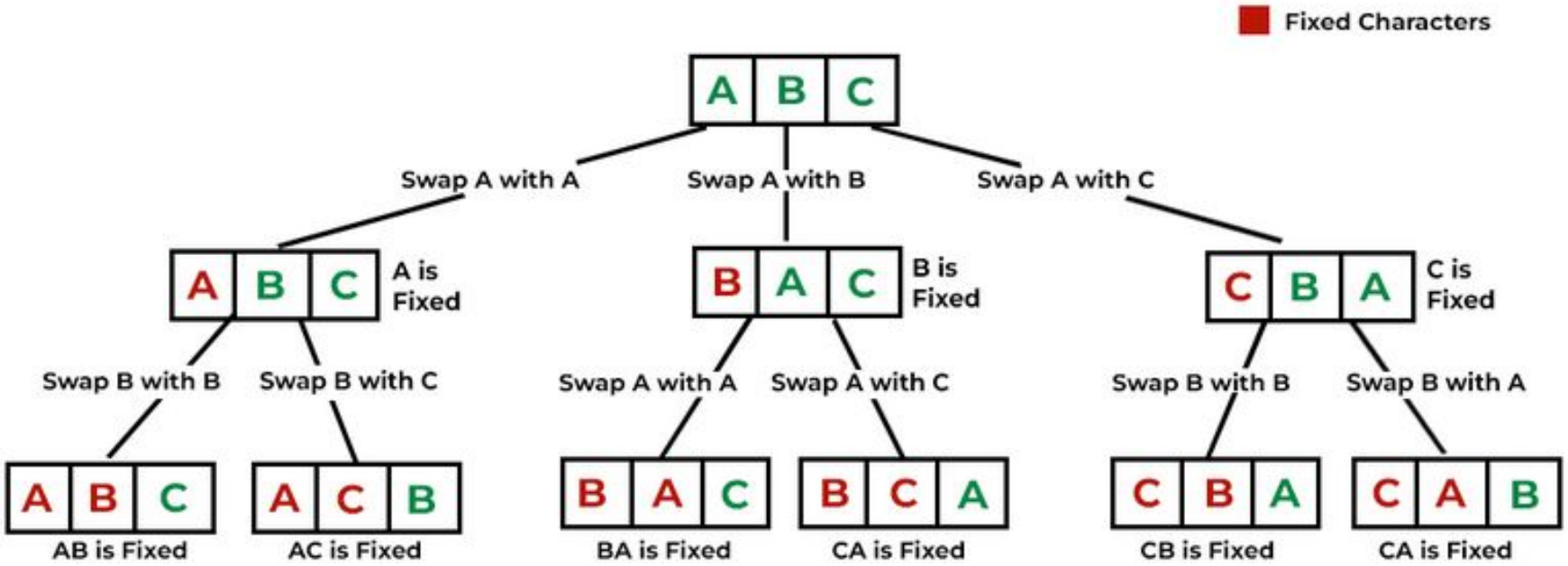
Recursive Backtracking Applications

- Puzzle solving (Sudoku, Crosswords, etc.)

- Game playing (Chess, Solitaire, etc.)



String permutation using backtracking



String Permutation - Backtracking Algorithm

- The backtracking function considers the first index of the given string.
- If the index is **N**, i.e. length of the string, it means that the current permutation is completed.
- Run a loop from current index **idx** until **N – 1** and do the following:
  - Create a new string which stores the original one
  - Use the new string and swap element at position i with the one at position idx (Swap S[i] and S[idx]).
  - Construct all other possible permutations (idx + 1).

Write your solution here: **classwork/xx_backtracking/PermuteString.java**

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

Call the method:

```
String str = "ABC";
permute(str, 0, str.length() - 1);
```