Brute force + Backtracking 02

PrepTech Google

Roteiro da aula de hoje:

- Revisitar o conceito força bruta;
- Entender o método Pruning com um exemplo;
- Exercícios com backtracking + pruning

Binary Watch

https://leetcode.com/problems/binary-watch/

Easy | Topics | Companies | Hint

A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

- For example, the below binary watch reads `"4:51"`.

Given an integer `turnedOn` which represents the number of LEDs that are currently on (ignoring the PM), return *all possible times the watch could represent*. You may return the answer in **any order**.

The hour must not contain a leading zero.

- For example, `"01:00"` is not valid. It should be `"1:00"`.

The minute must consist of two digits and may contain a leading zero.

- For example, `"10:2"` is not valid. It should be `"10:02"`.

Example 1:

```
Input: turnedOn = 1
Output:
["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00
"]
```

Example 2:

```
Input: turnedOn = 9
Output: []
```

ada

Binary Watch

https://leetcode.com/problems/binary-watch/

```java
public List<String> readBinaryWatch(int turnedOn) {
    List<String> result = new ArrayList<>();
    for(int hour = 0; hour <= 11; hour++) {
        for(int minute = 0; minute <= 59; minute++) {
            int numberOf1s = count1s(hour) + count1s(minute);
            if (numberOf1s == turnedOn) {
                String minuteString = minute < 10 ? "0" + minute : "" + minute;
                result.add(hour + ":" + minuteString);
            }
        }
    }
    return result;
}

private int count1s(int num) {
    int count = 0;
    while(num > 0) {
        count += num % 2;
        num /= 2;
    }
    return count;
}
```

```java
public List<String> readBinaryWatch(int turnedOn) {
    List<String> output = new ArrayList<>();

    for (int hour = 0; hour < 12; hour++) {
        for (int minute = 0; minute < 60; minute++) {
            int leds = count1s(hour) + count1s(minute);
            if (leds == turnedOn) {
                if (minute < 10) {
                    output.add(hour + ":0" + minute);
                } else {
                    output.add(hour + ":" + minute);}
            }
        }
    return output;
}
public int count1s(int n) {
    int counter = 0;
    while (n > 0) {
        counter += n % 2;
        n /= 2;
    }
    return counter;
}
```

Complexidade Computacional de tempo e de espaço: O(1)

Método Pruning

```java
public class Solution {
    public static void main(String[] args) {
        combinations(5, 3, new ArrayList<>());
    }
    public static void combinations(int n, int k, List<Integer> partial) {
        System.out.println(partial);
        if (partial.size() == k) {
            System.out.println(partial);
        } else {
            int start = partial.isEmpty() ? 1 : partial.get(partial.size() - 1) + 1;
            for (int i = start; i <= n; i++) {
                //pruning
                int availableNumbers = n + 1 - i;
                if (partial.size() + availableNumbers < k) break;

                partial.add(i);
                combinations(n, k, partial);
                partial.remove(partial.size() - 1);
            }
        }
    }
}
```

Letter Combinations of a Phone Number

https://leetcode.com/problems/letter-combinations-of-a-phone-number

Medium | Topics | Companies

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

ada

Letter Combinations of a Phone Number

https://leetcode.com/problems/letter-combinations-of-a-phone-number

```java
public List<String> letterCombinations(String digits) {
    if (digits.length() == 0) return new ArrayList<>();
    List<String> result = new ArrayList<>();
    Map<Character, String> keyboardMap = new HashMap<>();
    initKeyboard(keyboardMap);
    letterCombinations(keyboardMap, digits, 0,
                       new StringBuilder(), result);
    return result;
}

private void letterCombinations(Map<Character, String> keyboardMap,
                                String digits, int index,
                                StringBuilder partial, List<String> result) {
    if (index == digits.length()) {
        result.add(partial.toString());
        return;
    }

    char number = digits.charAt(index);
    char[] letters = keyboardMap.get(number).toCharArray();
    for(char letter : letters) {
        partial.append(letter); // try
        letterCombinations(keyboardMap, digits, index + 1, partial, result);
        partial.setLength(partial.length() - 1); // backtrack
    }
}
```

```java
void initKeyboard(Map<Character, String> map) {
    map.put('2', "abc");
    map.put('3', "def");
    map.put('4', "ghi");
    map.put('5', "jkl");
    map.put('6', "mno");
    map.put('7', "pqrs");
    map.put('8', "tuv");
    map.put('9', "wxyz");
}
```

Letter Combinations of a Phone Number

https://leetcode.com/problems/letter-combinations-of-a-phone-number

```java
String[] keyboard = new String[] { "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" };

public List<String> letterCombinations(String digits) {
    if (digits.length() == 0) return new ArrayList<>();
    List<String> result = new ArrayList<>();
    letterCombinations(digits, 0, new StringBuilder(), result);
    return result;
}

private void letterCombinations(String digits, int index,
                                StringBuilder partial, List<String> result) {
    if (index == digits.length()) {
        result.add(partial.toString());
        return;
    }

    int number = digits.charAt(index) - '0';
    char[] letters = keyboard[number].toCharArray();
    for(char letter : letters) {
        partial.append(letter); // try
        letterCombinations(digits, index + 1, partial, result);
        partial.setLength(partial.length() - 1); // backtrack
    }
}
```

Letter Combinations of a Phone Number

https://leetcode.com/problems/letter-combinations-of-a-phone-number

```
String[] keyboard = new String[] { "", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" };

public List<String> letterCombinationsRecursive(String digits) {
    if (digits.length() == 0) return new ArrayList<>();

    List<String> output = new ArrayList<>();
    if (digits.length() == 1) {
        int number = digits.charAt(0) - '0';
        char[] letters = keyboard[number].toCharArray();
        for(char letter : letters) {
            output.add("" + letter);
        }
    } else {
        List<String> tailOutput = letterCombinationsRecursive(digits.substring(1));
        int number = digits.charAt(0) - '0';
        char[] letters = keyboard[number].toCharArray();
        for(char letter : letters) {
            for(String tail : tailOutput) {
                output.add(letter + tail);
            }
        }
    }

    return output;
}
```

```java
class Solution {
    private static final String[] KEYPAD =
{       "",     // 0
        "",     // 1
        "abc",  // 2
        "def",  // 3
        "ghi",  // 4
        "jkl",  // 5
        "mno",  // 6
        "pqrs", // 7
        "tuv",  // 8
        "wxyz"  // 9 };
```

```java
    public List<String> letterCombinations(String digits) {
        List<String> result = new ArrayList<>();
        if (digits == null || digits.length() == 0) {
            return result;
        }
        backtrack(result, new StringBuilder(), digits, 0);
        return result;
    }
```

```java
private void backtrack(List<String> result, StringBuilder current,
String digits, int index) {
        if (index == digits.length()) {
            result.add(current.toString());
            return;
        }
        String letters = KEYPAD[digits.charAt(index) - '0'];
        for (char letter : letters.toCharArray()) {
            current.append(letter);
            backtrack(result, current, digits, index + 1);
            current.deleteCharAt(current.length() - 1);
        } } }
```

Complexidade Computacional de tempo: $O(4^n)$

Complexidade Computacional de espaço: $O(n\ 4^n)$ *onde n é o número de dígitos (profundidade máx da recursão)*

Sudoku Solver

https://leetcode.com/problems/sudoku-solver/

Hard | Topics | Companies

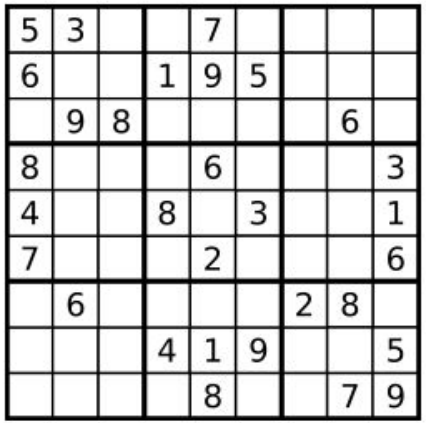
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.

Example 1:



ada

Sudoku Solver

https://leetcode.com/problems/sudoku-solver/

```java
private boolean backtracking(char[][] board, int lin, int col) {
    if (lin >= 9) return true;

    int nextCol = col < 8 ? col + 1 : 0;
    int nextLin = nextCol == 0 ? lin + 1 : lin;

    // '.' indicates an empty cell
    if (board[lin][col] != '.') {
        return backtracking(board, nextLin, nextCol);
    }

    Set<Character> invalid = new HashSet<>();
    // Mark numbers that are already in the same row, column as invalid
    for (int i = 0; i < 9; i++) {
        invalid.add(board[lin][i]);
        invalid.add(board[i][col]);
    }

    // Mark numbers in the 3x3 block as invalid
    int startRow = 3 * (lin / 3);
    int startCol = 3 * (col / 3);
    for (int i = startRow; i < startRow + 3; i++) {
        for (int j = startCol; j < startCol + 3; j++) {
            invalid.add(board[i][j]);
        }
    }

    // Try to fill the empty cell with a valid number
    for (char c = '1'; c <= '9'; c++) {
        if (!invalid.contains(c)) {
            board[lin][col] = c; // try
            if (backtracking(board, nextLin, nextCol)) {
                return true;
            }
            board[lin][col] = '.'; // backtrack
        }
    }

    return false;
}
```

```java
public void solveSudoku(char[][] board) {
    backtracking(board, 0, 0);
}
```

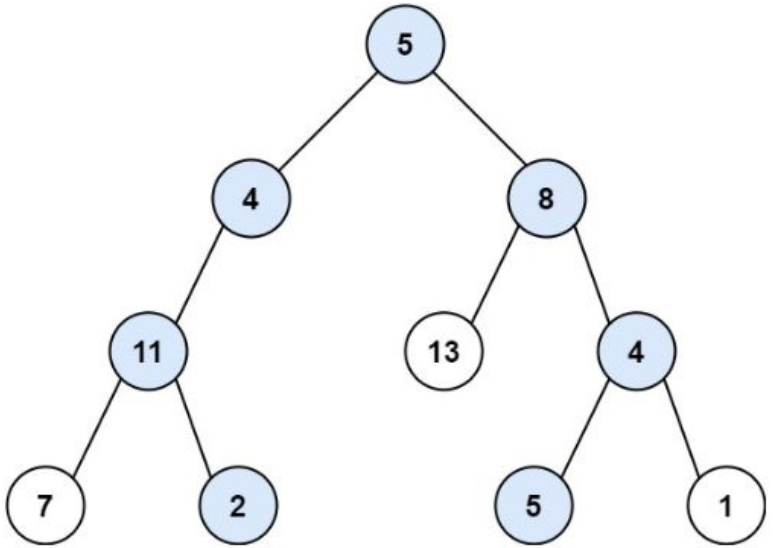
Path Sum II

https://leetcode.com/problems/path-sum-ii

Medium | Topics | Companies

Given the `root` of a binary tree and an integer `targetSum`, return *all* ***root-to-leaf*** *paths where the sum of the node values in the path equals* `targetSum`*. Each path should be returned as a list of the node* ***values****, not node references.*

A **root-to-leaf** path is a path starting from the root and ending at any leaf node. A **leaf** is a node with no children.



Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: [[5,4,11,2],[5,8,4,5]]
Explanation: There are two paths whose sum equals targetSum:
5 + 4 + 11 + 2 = 22
5 + 8 + 4 + 5 = 22

ada

Path Sum II

https://leetcode.com/problems/path-sum-ii

```java
private void preorder(TreeNode root, int targetSum, List<Integer> partial, List<List<Integer>> ans) {
    if (root == null) return;

    partial.add(root.val); // try
    targetSum -= root.val;

    if (root.left == null && root.right == null) {
        if (targetSum == 0) {
            ans.add(new ArrayList<>(partial)); // hard copy
        }
    } else {
        preorder(root.left, targetSum, partial, ans);
        preorder(root.right, targetSum, partial, ans);
    }

    partial.remove(partial.size() - 1); // backtrack
}

public List<List<Integer>> pathSum(TreeNode root, int targetSum) {
    List<List<Integer>> ans = new ArrayList<>();
    preorder(root, targetSum, new ArrayList<>(), ans);
    return ans;
}
```

Obrigado