Question
https://leetcode.com/problems/number-of-islands/ (https://leetcode.com/problems/number-of-islands/)

Python: DFS

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        # TODO: put a size check

        # visited matrix filled with zeros for each corresponding pos
ition in grid
        visited = [ [0 for _ in range(len(grid[0]))] for _ in range(l
en(grid)) ]

        count = 0


def numIslands(self, grid: List[List[str]]) -> int:
        count = 0
        for r,row in enumerate(grid):
            for c,col in enumerate(row):
                if grid[r][c] == '1':
                    self.removeNeighbors(r,c,grid)
                    count += 1
        return count

    def removeNeighbors(self, r ,c, grid):
        grid[r][c] = 0
        if r+1 < len(grid) and grid[r+1][c] == '1':
            self.removeNeighbors(r+1,c,grid)
        if c+1 < len(grid[0]) and grid[r][c+1] == '1':
            self.removeNeighbors(r,c+1,grid)
        if r-1 >= 0 and grid[r-1][c] == '1':
            self.removeNeighbors(r-1,c,grid)
        if c-1 >= 0 and grid[r][c-1] == '1':
            self.removeNeighbors(r,c-1,grid)
```

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        # TODO: put a size check

        # visited matrix filled with zeros for each corresponding pos
ition in grid
        visited = [ [0 for _ in range(len(grid[0]))] for _ in range(l
en(grid)) ]

        count = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                count += self.numIslandsUtil(i, j, grid, visited)

        return count


    def numIslandsUtil(self, r, c, grid, visited) -> int:
        if not self.isWithinBoundary(r, c, grid):
            return 0

        if grid[r][c] == "0":
            return 0

        if visited[r][c] == 1:
            return 0

        visited[r][c] = 1
        self.numIslandsUtil(r,c+1,grid, visited)
        self.numIslandsUtil(r,c-1,grid, visited)
        self.numIslandsUtil(r+1,c,grid, visited)
        self.numIslandsUtil(r-1,c,grid, visited)
        return 1

    def isWithinBoundary(self, r, c, grid):
        if r >=0 and r < len(grid) and c >= 0 and c < len(grid[0]):
            return True
        return False
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Question
https://leetcode.com/problems/max-area-of-island/ (https://leetcode.com/problems/max-area-of-island/)

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:

        # visited matrix filled with zeros for each corresponding pos
ition in grid
        visited = [ [0 for _ in range(len(grid[0]))] for _ in range(l
en(grid)) ]

        maxSize = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                maxSize = max(self.numIslandsUtil(i, j, grid, visite
d), maxSize)

        return maxSize


    def numIslandsUtil(self, r, c, grid, visited):
        if not self.isWithinBoundary(r, c, grid):
            return 0

        if grid[r][c] == "0":
            return 0

        if visited[r][c] == 1:
            return 0

        visited[r][c] = 1

        return 1 + self.numIslandsUtil(r,c+1,grid, visited) +
                   self.numIslandsUtil(r,c-1,grid, visited) +
                   self.numIslandsUtil(r+1,c,grid, visited) +
                   self.numIslandsUtil(r-1,c,grid, visited)

    def isWithinBoundary(self, r, c, grid):
        if r >=0 and r < len(grid) and c >= 0 and c < len(grid[0]):
            return True
        return False
     0   1   2   3   4
0 ["1","1","0","0","0"],   [ 1  1  0  0  0],
1 ["1","1","0","0","0"],   [ 1  1  0  0  0]
2 ["0","0","1","0","0"],   [ 0  0  0  0  0]
3 ["0","0","0","1","1"]    [ 0  0  0  0  0]
```

Java

```java
class Solution {
    public int dfs(int [][] grid, int i , int j ){
        if(i < 0 || j < 0 || i >= grid.length  || j >= grid[0].length
|| grid[i][j] != 1){
            return 0;
        }
        grid[i][j] = 2;
        int ans = 1;
        ans += dfs(grid,i+1,j);
        ans += dfs(grid,i-1,j);
        ans += dfs(grid,i,j-1);
        ans += dfs(grid,i,j+1);

        return ans;

    }
    public int maxAreaOfIsland(int[][] grid) {
        int ans =0 ;

        for(int i =0 ;i < grid.length ; i++){
            for(int j = 0; j < grid[0].length ; j++){
                if(grid[i][j] == 1){
                    ans = Math.max(ans , dfs(grid,i ,j));
                }
            }
        }
        return ans;
    }
}
```

In [ ]:
```
1
```

Permutations

In [4]:

```
def make_permutations(symbols, stack, res):

    if len(symbols) == 0:
        res.append(stack.copy())
        return

    for i in range(len(symbols)):

        curr = symbols[i]
        left = symbols[0:i] # take symbols from index 0 till i-1
        right = symbols[i+1:] # take symbols from index i+1 till end

        stack.append(curr)
        make_permutations(left + right, stack, res)
        stack.pop()

r = []
make_permutations('ABCD', [], r)

print(r)
print(len(r))
print(4*3*2*1)
```

```
[['A', 'B', 'C', 'D'], ['A', 'B', 'D', 'C'], ['A', 'C', 'B', 'D'], ['A', 'C',
'D', 'B'], ['A', 'D', 'B', 'C'], ['A', 'D', 'C', 'B'], ['B', 'A', 'C', 'D'],
['B', 'A', 'D', 'C'], ['B', 'C', 'A', 'D'], ['B', 'C', 'D', 'A'], ['B', 'D',
'A', 'C'], ['B', 'D', 'C', 'A'], ['C', 'A', 'B', 'D'], ['C', 'A', 'D', 'B'],
['C', 'B', 'A', 'D'], ['C', 'B', 'D', 'A'], ['C', 'D', 'A', 'B'], ['C', 'D',
'B', 'A'], ['D', 'A', 'B', 'C'], ['D', 'A', 'C', 'B'], ['D', 'B', 'A', 'C'],
['D', 'B', 'C', 'A'], ['D', 'C', 'A', 'B'], ['D', 'C', 'B', 'A']]
24
24
```

```
void makePermutations(string symbols, vector<char>  &stack, vector<ve
ctor<char>> &res){

    if (symbols.size() == 0)
        res.push_back(stack);

    for (int i=0; i < symbols.size(); i++){

        auto curr = symbols.at(i);
        auto left = symbols.substr(0, i);
        auto right = symbols.substr(i+1,symbols.size());

        stack.push_back(curr);
        makePermutations(left + right, stack, res);
        stack.pop_back();
    }

}


int main() {
    vector<vector<char>> res;
    vector<char> stack;

    makePermutations("ABCD", stack, res);
    for (auto row: res) {
        for (auto c: row) {
            cout << c;
        }
        cout << endl;
    }

}
```

```
vector<string> makePermutations(string symbols){

    if (symbols.size() == 0) {
        vector<string> res = {""};
        return res;
    }

    vector<string> res;
    for (int i=0; i < symbols.size(); i++){

        auto curr = symbols.at(i);
        auto left = symbols.substr(0, i);
        auto right = symbols.substr(i+1,symbols.size());

        vector<string> currPerms = makePermutations(left + right);
        for (int i = 0; i < currPerms.size(); i++) {
            res.push_back( curr + currPerms[i]);
        }
    }
    return res;
}


int main() {

    vector<string> results = makePermutations("ABCD");
    for (auto perm : results) {
        cout << perm << endl;
    }

}
```

Question
https://leetcode.com/problems/sudoku-solver/ (https://leetcode.com/problems/sudoku-solver/)

C++ rough code: use at your own risk

```
class Solution {
public:
    void solveSudoku(vector<vector<char>>& board) {

    }

    bool solveSudokuUtil(vector<vector<char>>& board, int r, int c) {
        if (r == 10)
            return True

        if (board[r][c] != '.') {
            if (c == 9)
                r = r+1
                c = -1
            if (solveSudokuUtil(board, r, c + 1))
                return true
        }

        for (ini i = 1; i < 10; i++){

            if (isValidForCurrPos) {
                board[r][c] = itoa(i);

                if (c == 9)
                    r = r+1
                    c = -1
                if (solveSudokuUtil(board, r, c + 1))
                    return true

            }

            board[r][c] = '.';
        }
    }

    bool isValidForCurrPos(int num, int r, int c, vector<vector<char>
>& board) {
        // TODO: implement

        for (int i = 0; i < 9; i++) {
            // board[r][i] ??
        }

        for (int i = 0; i < 9; i++) {
            // board[c][i] ??
        }

        int rr = (r/3)*3;
        int cc = (c/3)*3;
```

```
for (int i = r; i <r+3;i++)
```

In [ ]:
```
1
```

In [ ]:
```
1
```