# Data Structures

Homework Assignment 3 - Recursion

Problem 1 - Deepest Element in Nested List - 25 Points
Problem 2 - Sum of Positive Numbers in Nested Dictionary - 25 Points
Problem 3 - Permutation - 25 Points
Problem 4 - Maze Path - 25 Points

**25% of Gradescope Autograder test cases are hidden for this assignment.**

Fall 2024 - Revision 1 - *September*

# Problem 1 - Deepest Element - 25 Points

Write a recursive function that takes a list of lists $l$ and returns the deepest nested element.

### Example 1

```
L = [[[7, 1]]]
res = deepest(L)
# res == [7, 1]
```

### Example 2

```
L = [[[7, [8, [9]]]]]
res = deepest(L)
# res == [9]
```

### Requirements

- You have to use recursion.
- Your function must use a constant amount of space.
- You can only use primitive data types.
- You can not cast **L** to string.

### Hint

- There is no solution in which two or more lists have equally deep elements as their deepest element.

# Problem 2 - Sum of Positive Numbers - 25 Points

Write a recursive Python function that returns the sum of all positive numbers in a dictionary **d** with nested dictionary elements.

### Example 1

```
d = {“a”: 2, “b”: {“c”: 3, “d”: {“e”: 2}}}
res = rec_sum(d)
# res == 7
```

### Example 2

```
d = {“nn”: {“lil”: 2}, “mm”: “car”}
res = rec_sum(d)
# res == 2
```

### Requirements

- You have to use recursion.
- Your function must use a constant amount of space.
- You can only use primitive data types.
- You can not cast **d** to string.

# Problem 3 - Permutation - 25 Points

The set **s** contains the natural numbers **[1, n]** with **n!** unique permutations. Write a recursive function that returns the ***i-th*** permutation for a given **n** in lexicographical order.

Lexicographical order for n = 3:

1. "123"
2. "132"
3. "213"
4. "231"
5. "312"
6. "321"

### Example 1

```
n = 3, i = 2
res = perm(n, i)
# res == “132”
```

### Example 2

```
n = 4, i = 9
res = perm(n, 1)
# res == “2314”
```

### Requirements

- You have to use recursion.
- Your function can be at most $O(n^2)$ time complex.
- Provided bounds: 1 <= n <= 9; 1 <= i <= n!

# Problem 4 - Maze Path - 25 Points

Given a maze represented by a 2D array of integers, write the recursive function solver(maze) to find a path from the starting point (top-left corner) to the ending point (bottom-right corner). The maze contains obstacles represented by 0s, and free spaces represented by 1s. The function should return a list of tuples representing the path coordinates from start to end, or an empty list if no path exists. We define the priority of each direction: down > right > up > left. Your solution should choose the direction with the highest priority if there are multiple directions available.

### Example 1

```
res = solver([[1, 0, 1, 1, 1],
              [1, 0, 1, 0, 1],
              [1, 0, 1, 0, 1],
              [1, 1, 1, 0, 1]])

print(res ) # Should print:
            # [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (2, 2),
            # (1, 2), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4)]
```

### Example 2

```
res = solver([[1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
              [0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1],
              [1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1],
              [1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1],
              [1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
              [1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
              [1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1],
              [1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1],
              [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1],
              [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

print(res ) # Should print: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2),
            # (3, 2), (4, 2), (5, 2), (5, 3), (5, 4), (4, 4), (3, 4),
            # (2, 4), (1, 4), (0, 4), (0, 5), (0, 6), (0, 7), (1, 7),
            # (2, 7), (3, 7), (4, 7), (4, 8), (4, 9), (4, 10), (3, 10),
            # (2, 10), (2, 11), (2, 12), (3, 12), (4, 12), (5, 12),
            # (6,12), (7, 12), (8, 12), (9, 12), (9, 13), (9, 14)]
```

### Requirements

- You have to use recursion.

### Time Complexity

- **O(mn)**, at most (where m and n are the dimensions of the maze)