# horizontal lineData Structures

Homework Assignment 3 - Recursion

**Update**

* Feb 23, 10 am - Problem 4 - **eased space complexity** constraint to **O(n^2)**

Problem 1 - Subsets - 25 Points

Problem 2 - Knapsack - 25 Points

Problem 3 - Expression Combination - 25 Points

Problem 4 - Maze Path - 25 Points

**Notes and Requirements**

* Your submission must be your effort. You can not copy other students' code.
* This worksheet is graded on performance; Implementations must be correct.
* You are encouraged to visit our office hours to ask coding questions.
* Only the latest (most recent) submission is graded.
* Late submissions are not considered for grading.
* You can not use any third-party libraries.

**Some assignments on this worksheet are manually graded.**

## Problem 1 - Subsets - 25 Points

Implement the function *subsets(s)* to generate all possible subsets of a given set.

**Example 1**

res = subsets(['A', 'B', 'C'])

print(res) # Should print [[], ['C'], ['B'], ['B', 'C'],

# ['A'], ['A', 'C'], ['A', 'B'], ['A', 'B', 'C']]

**Example 2**

res = subsets(['K', 'L', 'M', 'Z'])

print(res) # Should print [[], ['Z'], ['M'], ['M', 'Z'], ['L'],

# ['L', 'Z'], ['L', 'M'], ['L', 'M', 'Z'], ['K'],

# ['K', 'Z'], ['K', 'M'], ['K', 'M', 'Z'],

# ['K', 'L'], ['K', 'L', 'Z'], ['K', 'L', 'M'],

# ['K', 'L', 'M', 'Z']]

**Requirements**

* You have to use recursion.
* The space complexity of your implementation must be **O(2^n)**.
* You can only use Python lists. Sets are not allowed.
* You can return the result list in any order.

## Problem 2 - Knapsack - 25 Points

The knapsack is an optimization challenge where, given a set of items with associated weights and values, the goal is to select a subset of items to maximize the total value without exceeding a predefined weight limit (capacity of the knapsack). Write a recursive Python function to solve the knapsack problem given a list of items with their weights and values and a maximum weight capacity for the knapsack.

**Example**

weights = [1, 3, 4, 5]

values = [1, 4, 5, 7]

capacity = 7

n = len(weights)

result = knapsack\_recursive(weights, values, capacity, n)

print(result) # Output: 9. Weights 3 and 4 have been selected

# with values 4 and 5.

**Requirements**

* You have to use recursion.
* The space complexity of your implementation must be **O(1)**.
* You can only use primitive variables (no stack, queue, string, etc.).

## Problem 3 - Expression Combination - 25 Points

Implement the recursive function *expression(numbers)* to generate all possible valid expressions that can be formed using a list of integers and operators. The operators are addition (+), subtraction (-), and multiplication (\*), and they can be used between any two adjacent integers. The function should return a list of all possible valid expressions that can be formed using the given integers and operators.

**Example 1**

res = expression([1,2,3])

print(res ) # Should print: ["1+2+3", "1+2-3", "1+2\*3", "1-2+3",

# "1-2-3", "1-2\*3", "1\*2+3", "1\*2-3", "1\*2\*3"]

**Example 2**

res = expression([8,9])

print(res ) # Should print: ["8+9", "8\*9", "8-9"]

**Requirements**

* You have to use recursion.
* The space complexity of your implementation can be at most **O(3^(n-1))**.
* You can return the list elements in any order.

## 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 multiple directions are 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.
* The space complexity of your implementation can be at most **O(n^2)** (auxiliary).
* The time complexity of your implementation can be at most **O(n^2)**.