**Week 6 Submit Task**

**Part A**

For these questions, take the time to practise your exam writing skills.

1. Define backtracking and explain its basic operation. (3 marks)

Backtracking is a technique used to solve problems by incrementally building options to the solution and dropping an option as soon as it’s determined that it can’t possibly lead to a valid solution. This approach can be thought of as a depth-first search of the solution space, where each step involves making a choice from a set of possible options. By systematically trying out all possible options and abandoning paths that do not lead to a valid solution, backtracking ensures that all potential solutions are considered, while also potentially reducing the number of paths explored through pruning and constraint checks.

1. Explain the concept of recursion and its role in backtracking algorithms. (2 marks)

Recursion is where a function calls itself directly or indirectly to solve a problem. In backtracking, it plays crucial role as it allows the algorithm to explore each branch of the solution space systematically. Each call represents a step in the process of building the solution, and when a dead end is encountered, the function can backtrack by returning to the previous call

1. What is the purpose of pruning in backtracking algorithms? Provide an example of how pruning can improve the efficiency of a backtracking solution. (2 marks)

Pruning in backtracking algorithms aims to eliminate branches in the search space that do not lead to a valid solution early in the process. This improves efficiency by reducing the number of possible solutions that need to be explored.

In the N-Queens problem for example, if two queens are placed in the same row, column, or diagonal, any further placements in that branch are guaranteed to be invalid. By pruning these branches early, we avoid unnecessary exploration and significantly reduce the number of potential solutions to check.

1. What are some common applications of backtracking algorithms in computer science or real-world scenarios? (3 marks)

Pathfinding, TSP (optimal sol while exploring all possible routes), Puzzles where solution must satisfy a lot of constraints like sudoku

1. Describe the graph colouring problem and explain how backtracking can be used to solve it. (3 marks)

The graph colouring problem involves assigning colours to the vertices of a graph such that no two adjacent vertices share the same colour, using a minimum number of colours. Backtracking can be used to solve this problem by:

* Assigning a colour to a vertex.
* Recursively trying to colour the next vertex while ensuring no two adjacent vertices have the same colour.
* Backtracking when a conflict is detected, i.e., when a vertex cannot be coloured without violating the constraints, and trying the next available colour.

1. Discuss the time complexity of the graph colouring problem when solved using backtracking. (2 marks)

Exponential as it explores all possible ways to color a graphy. For graph with V verts and k colours, worst case is O(k^v) but can be reduced with pruning and heuristics

**Part B – Optional Extension**

The Subset Sum problem involves, for a given set of integers and target sum, determining whether there is a subset of the given set whose elements sum up to the target value. This can be solved with backtracking.

Write Python code based on the following pseudocode structure. It should return the current subset if the target is found, or null if the target cannot be found. Otherwise add the next element from the set to the current subset and call recursively. If the subset is not found, the element should be excluded and the call repeated with the next element.

function subsetSum(set, targetSum):

return backtrack(set, targetSum, [], 0)

function backtrack(set, targetSum, currentSubset, currentIndex):

'''The Subset Sum problem involves, for a given set of integers and target sum, determining whether there is a subset of the given set whose elements sum up to the target value. This can be solved with backtracking.

Write Python code based on the following pseudocode structure. It should return the current subset if the target is found, or null if the target cannot be found. Otherwise add the next element from the set to the current subset and call recursively. If the subset is not found, the element should be excluded and the call repeated with the next element.

function subsetSum(set, targetSum):

  return backtrack(set, targetSum, [], 0)

function backtrack(set, targetSum, currentSubset, currentIndex):

'''

*def* subsetSum(*set*, *targetSum*):

    return backtrack(*set*, *targetSum*, [], 0)

*def* backtrack(*set*, *targetSum*, *currentSubset*, *currentIndex*):

    if *targetSum* == 0:

        return *currentSubset*

    if *targetSum* < 0 or *currentIndex* >= len(*set*):

        return None

    print("Set: "+str(*set*) + " New sum: "+ str(*targetSum* -*set*[*currentIndex*]) + " Unknown: " + str(*currentSubset* + [*set*[*currentIndex*]]), " New idx: " + str(*currentIndex*+1) )

    withElement = backtrack(*set*, *targetSum* - *set*[*currentIndex*], *currentSubset* + [*set*[*currentIndex*]], *currentIndex* + 1)

    if withElement:

        return withElement

    return backtrack(*set*, *targetSum*, *currentSubset*, *currentIndex* + 1)

# Test cases

print(subsetSum([3, 34, 4, 12, 5, 2], 9)) # [4, 5]

print(subsetSum([3, 34, 4, 12, 15, 2], 30)) # [3, 12, 15]

print(subsetSum([3, 34, 4, 12, 5, 2], 1)) # None

print(subsetSum([3, 34, 4, 12, 5, 2], 0)) # []