Question 1:

Implement a program to solve the N-Queens problem on an N×N chessboard. The goal is to place N queens on the board in such a way that no two queens threaten each other. Design a program that takes the size of the chessboard as input and outputs one possible arrangement of queens or a message indicating that no solution exists.

Input Format:

- The first line contains an integer 'N' (1 <= N <= 15), representing the size of the chessboard.

Output Format:

- If a valid arrangement exists, the program should output the chessboard configuration with 1s representing queens and 0s representing empty squares. Each row should be printed on a new line.
- If no valid arrangement exists, the program should output "No solution exists."

Title for Question 1: N-Queens Problem

Solution:

```
#include<iostream>
#define size 10
using namespace std;

void display(int mat[size][size],int N) {
    for(int i=0;i<N;i++,cout<<endl)
        for(int j=0;j<N;j++)
            cout<<mat[i][j]<<" ";
    cout<<endl;
}

bool check(int mat[size][size],int N,int row,int col) {
    int i,j;
    // vertical
    for(i=row;i>=0;i--)
        if(mat[i][col]==1)
            return false;
    // left Diag
    for(i=row,j=col;i>=0&&j>=0;i--,j--)
        if(mat[i][j]==1)
            return false;
    // right Diag
    for(i=row,j=col;i>=0&&j<N;i--,j++)
        if(mat[i][j]==1)
            return false;
    return true;
}
```

```
void NQueen(int mat[size][size],int N,int row) {
    int col;
    if(N == row) {
        display(mat,N);
        return;
    }

    for(col=0;col<N;col++) {
        if(check(mat,N,row,col)) {
            mat[row][col] = 1;
            NQueen(mat,N,row+1);
            mat[row][col] = 0;
        }
    }
}

int main() {
    int N;
    cin>>N;
    int mat[size][size];
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            mat[i][j] = 0;
    if(N==2||N==3) {
        cout<<"No solution exists.";
        return 0;
    }
    NQueen(mat,N,0);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	4	0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0
2	3	No solution exists.
3	2	No solution exists.
4	5	1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0
5	6	0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0
6	1	1

White List:

Black List:

Question 2:

You are working on a text processing application, and your task is to create a function that generates all possible combinations of strings from a given set of characters. Each combination should be unique, and the length of the combinations should vary from 1 to the length of the input string. Write a function that takes a string of characters as input and returns a list of all possible unique combinations of strings.

Input Format:

- The string should consist of alphanumeric characters, and it can be of any length.

Output Format:

- The program generates and prints all the combinations of the input string.
- Each combination is printed on a new line.

Title for Question 2: String Combination

Solution:

```
#include <iostream>
#include <algorithm>

int Check(std::string str,int s,int i) {
    while(s<i) {
        if(str[i] == str[s])
            return 0;
        s++;
    }
    return 1;
}

void Combination(std::string str, int start, int end) {
    if (start == end) {
        std::cout << str << std::endl;
        return;
    }

    for (int i = start; i <= end; ++i) {
        if(Check(str,start,i)) {
            std::swap(str[start], str[i]);
            Combination(str, start + 1, end);
            std::swap(str[start], str[i]);
        }
    }
}

int main() {
    std::string input;
    std::cin >> input;
    Combination(input, 0, input.length() - 1);
    return 0;
```

```
}
```

TestCases:

S.No	Inputs	Outputs
1	abb	abb bab bba
2	bbb	bbb
3	abc	abc acb bac bca cba cab
4	ab	ab ba
5	abcd	abcd abdc acbd acdb adcb adbc bacd badc bcad bcda bdca bdac cbad cbda cabd cadb cdab cdba dbca dbac dcba dcab dacb dabc
6	a	a

White List:

Black List:

Question 3:

Imagine you are organizing a backpack for a hiking trip. Each item in your backpack has a specific weight, and you are trying to determine if it's possible to select a subset of items whose total weight equals a target value. The weights of the items are given in an array, and you need to write a program to check if there is a subset whose total weight matches the target.

Input Format:

- The first line contains an integer n, representing the number of elements in the set.
- The second line contains n space-separated integers, representing the elements of the set.
- The third line contains an integer sum, representing the target sum.

Output Format:

- If there exists a subset of the given set with the sum equal to the target sum, the program
- Subset with the given sum exists.
- If there is no subset with the sum equal to the target sum, the program outputs:
- Subset with the given sum does not exist.

Title for Question 3: Subset Sum

Solution:

```
#include <iostream>
#include <vector>
```

```
bool isSubsetSum(const std::vector<int>& set, int n, int sum) {
    std::vector<std::vector<bool>> dp(n + 1, std::vector<bool>(sum + 1, f

    // If the sum is 0, then the empty set is a subset with the given sum
    for (int i = 0; i <= n; ++i)
        dp[i][0] = true;

    // Fill the dp table
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= sum; ++j) {
            if (set[i - 1] > j)
                dp[i][j] = dp[i - 1][j];
            else
                dp[i][j] = dp[i - 1][j] || dp[i - 1][j - set[i - 1]];
        }
    }

    // Return the result
    return dp[n][sum];
}

int main() {
    int n;
    std::cin >> n;
    std::vector<int> set(n);
    for (int i = 0; i < n; ++i)
        std::cin >> set[i];
    int sum;
    std::cin >> sum;
    if (isSubsetSum(set, n, sum))
        std::cout << "Subset with the given sum exists.\n";
    else
        std::cout << "Subset with the given sum does not exist.\n";
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	4 5 2 12 2 8	Subset with the given sum does not exist.
2	5 3 7 1 2 8 11	Subset with the given sum exists.
3	4 5 2 3 1 8	Subset with the given sum exists.
4	5 1 2 9 4 5 6	Subset with the given sum exists.
5	6 1 2 3 4 5 6 12	Subset with the given sum exists.
6	3 10 5 3 8	Subset with the given sum exists.

White List:

Black List:

Question 4:

You are a software developer working on a social network platform that connects users based on shared interests. The platform utilizes a graph data structure to represent user connections, where

each user is a node, and edges between nodes indicate shared interests. To enhance user experience, you are tasked with modifying the coloring algorithm used to visualize these connections. The goal is to ensure that users with similar interests are visually distinguishable by assigning them the same color. Your task is to modify the coloring algorithm to return a valid coloring of the graph if it exists

Input Format:

- The first line contains an integer numVertices, representing the number of vertices in the graph.
- The second line contains an integer numEdges, representing the number of edges in the graph.
- The next numEdges lines contain two integers u and v, separated by a space, indicating an edge between vertices u and v.
- The last line contains an integer colors, representing the number of colors to be used for graph coloring.

Output Format:

- If a valid coloring exists : The program outputs "Valid Coloring:" on a new line.
- For each vertex i, the program outputs a line in the format: "Vertex i: Color color[i]".
- If no valid coloring exists: The program outputs "No valid coloring exists." on a new line.

Title for Question 4: Valid Coloring

Solution:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Graph {
public:
    int vertices;
    vector<vector<int>> adjacencyMatrix;

    Graph(int v) : vertices(v), adjacencyMatrix(v, vector<int>(v, 0)) {}

    void addEdge(int u, int v) {
        adjacencyMatrix[u][v] = 1;
        adjacencyMatrix[v][u] = 1;
    }
};

bool isSafe(int v, Graph& graph, vector<int>& color, int c) {
    for (int i = 0; i < graph.vertices; ++i) {
        if (graph.adjacencyMatrix[v][i] && c == color[i]) {
            return false;
        }
    }
    return true;
```

```
}

bool graphColoringUtil(Graph& graph, int m, vector<int>& color, int v) {
    if (v == graph.vertices) {
        return true;
    }

    for (int c = 1; c <= m; ++c) {
        if (isSafe(v, graph, color, c)) {
            color[v] = c;

            if (graphColoringUtil(graph, m, color, v + 1)) {
                return true;
            }

            color[v] = 0; // Backtrack
        }
    }
    return false;
}

bool graphColoring(Graph& graph, int m, vector<int>& color) {
    if (!graphColoringUtil(graph, m, color, 0)) {
        cout << "No valid coloring exists." << endl;
        return false;
    }
    return true;
}

int main() {
    int numVertices, numEdges, colors;
    cin >> numVertices;
    Graph graph(numVertices);
    cin >> numEdges;
    for (int i = 0; i < numEdges; ++i) {
        int u, v;
        cin >> u >> v;
        graph.addEdge(u, v);
    }
    cin >> colors;
    vector<int> color(numVertices, 0);
    if (graphColoring(graph, colors, color)) {
        cout << "Valid Coloring:" << endl;
        for (int i = 0; i < numVertices; ++i) {
            cout << "Vertex " << i << ": Color " << color[i] << endl;
        }
    }
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 6 0 1 0 2 1 2 1 3 2 4 3 4 3	Valid Coloring: Vertex 0: Color 1 Vertex 1: Color 2 Vertex 2: Color 3 Vertex 3: Color 1 Vertex 4: Color 2
2	5 8 0 1 0 2 1 2 1 3 2 4 3 4 3 0 4 0 2	No valid coloring exists.

S.No	Inputs	Outputs
3	6 10 0 1 0 2 1 2 1 3 2 4 3 4 3 5 4 5 5 0 5 1 3	Valid Coloring: Vertex 0: Color 1 Vertex 1: Color 2 Vertex 2: Color 3 Vertex 3: Color 1 Vertex 4: Color 2 Vertex 5: Color 3
4	4 5 0 1 1 2 2 3 3 0 1 3 2	No valid coloring exists.
5	4 4 0 1 1 2 2 3 3 0 3	Valid Coloring: Vertex 0: Color 1 Vertex 1: Color 2 Vertex 2: Color 1 Vertex 3: Color 2
6	5 8 0 1 0 2 1 2 1 3 2 4 3 4 3 0 4 0 3	Valid Coloring: Vertex 0: Color 1 Vertex 1: Color 2 Vertex 2: Color 3 Vertex 3: Color 3 Vertex 4: Color 2

White List:

Black List:

Question 5:

Imagine you are tasked with designing a navigation system for a delivery drone fleet in a city. The city is represented as a graph, where each intersection is a vertex, and roads between intersections are edges. To optimize the drone routes, you need to find a Hamiltonian cycle in the graph, ensuring that each intersection is visited exactly once, and the cycle starts and ends at the same intersection. How would you adapt a backtracking algorithm to find a Hamiltonian cycle.

Input Format:

- The first line of input specifies the number of vertices in the graph.
- The second line of input specifies the number of edges in the graph.
- The following lines represent the edges of the graph, where each line contains two integers representing the vertices connected by an edge.

Output Format:

- If a Hamiltonian cycle is found, the program outputs the vertices of the Hamiltonian cycle in order.
- If a Hamiltonian cycle is not found, the program outputs a message indicating that no Hamiltonian cycle exists.
- Hamiltonian Cycle not found.

Title for Question 5: Hamiltonian Cycle Problem

Solution:

```
#include <iostream>
#include <vector>
using namespace std;

class HamiltonianCycle {
public:
    HamiltonianCycle(int vertices);
```

```
    void addEdge(int v, int w);

    void findHamiltonianCycle();

private:
    bool isSafe(int v, int pos);

    bool solve(int pos);

    int vertices;
    vector<vector<int>> graph;
    vector<int> path;
};

HamiltonianCycle::HamiltonianCycle(int vertices) {
    this->vertices = vertices;
    graph.resize(vertices, vector<int>(vertices, 0));
    path.resize(vertices, -1);
}

void HamiltonianCycle::addEdge(int v, int w) {
    graph[v][w] = 1;
    graph[w][v] = 1;
}

bool HamiltonianCycle::isSafe(int v, int pos) {
    if (!graph[path[pos - 1]][v]) {
        return false;
    }

    for (int i = 0; i < pos; ++i) {
        if (path[i] == v) {
            return false;
        }
    }
    return true;
}

bool HamiltonianCycle::solve(int pos) {
    if (pos == vertices) {
        // Check if the last vertex is connected to the first vertex
        if (graph[path[pos - 1]][path[0]] == 1) {
            return true; // Hamiltonian cycle found
        }
        return false;
    }

    for (int v = 1; v < vertices; ++v) {
        if (isSafe(v, pos)) {
            path[pos] = v;

            if (solve(pos + 1)) {
                return true;
            }

            path[pos] = -1; // Backtrack
        }
    }
    return false;
}
```

```
void HamiltonianCycle::findHamiltonianCycle() {
    path[0] = 0; // Start from the first vertex

    if (solve(1)) {
        cout << "Hamiltonian Cycle found: ";
        for (int i = 0; i < vertices; ++i) {
            cout << path[i] << " ";
        }
        cout << path[0] << endl;
    } else {
        cout << "Hamiltonian Cycle not found." << endl;
    }
}

int main() {
    int vertices, edges;
    cin >> vertices;
    HamiltonianCycle graph(vertices);
    cin >> edges;
    for (int i = 0; i < edges; ++i) {
        int v, w;
        cin >> v >> w;
        graph.addEdge(v, w);
    }
    graph.findHamiltonianCycle();
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	4 5 0 1 1 2 2 3 3 0 0 2	Hamiltonian Cycle found: 0 1 2 3 0
2	7 10 0 1 1 2 2 3 3 4 4 5 5 0 0 3 1 4 2 5 3 6	Hamiltonian Cycle not found.
3	6 9 0 1 1 2 2 3 3 4 4 5 5 0 0 3 1 4 2 5	Hamiltonian Cycle found: 0 1 2 3 4 5 0
4	3 2 0 1 1 2	Hamiltonian Cycle not found.
5	6 8 0 1 1 2 2 3 3 4 4 5 5 0 0 3 1 4	Hamiltonian Cycle found: 0 1 2 3 4 5 0
6	5 7 0 1 1 2 2 3 3 4 4 0 0 2 1 4	Hamiltonian Cycle found: 0 1 2 3 4 0

White List:

Black List:

Question 6:

A rat starts from the top left corner of a square maze and wants to reach the bottom right corner where its food is placed. The maze is represented as a 2D square matrix of dimension n, where each cell can either be a 0 (representing a wall) or a 1 (representing a path where the rat can move). The rat can move only in two directions: forward and down.

Create a program that reads a square maze and outputs one possible path for the rat to reach its food if such a path exists. The path should be marked in the maze, with all the positions the rat will visit marked as 1 in the solution matrix. If no path exists, your program should output "No solution exists."

Input Format

- The first line contains an integer n, representing the dimensions of the square maze (n x n).
- The next n lines contain n integers each, representing the maze's rows. Each integer is either 0 (wall) or 1 (path).

Output Format

- If a solution exists, print the given maze with an additional message "Given Maze:" followed by the solution matrix with an additional message "Solution:". Each row of the maze and solution matrix should be printed on a new line, with each integer in the row separated by a space.
- If no solution exists, print "Given Maze:" followed by the maze and then print "No solution exists."

Title for Question 6: Rat in a Maze

Solution:

```
#include <iostream>
#include <vector>
using namespace std;

// Utility function to print the solution matrix
void printSolution(const vector<vector<int>>& solution) {
    for (int i = 0; i < solution.size(); i++) {
        for (int j = 0; j < solution[i].size(); j++) {
            cout << solution[i][j] << " ";
        }
        cout << endl;
    }
}

// Utility function to check if a move is valid
bool isSafe(const vector<vector<int>>& maze, int x, int y) {
    return (x >= 0 && x < maze.size() && y >= 0 && y < maze[0].size() &&
}

bool solveMazeUtil(const vector<vector<int>>& maze, int x, int y, vector<
    // If (x,y) is the goal, return true
    if (x == maze.size() - 1 && y == maze[0].size() - 1 && maze[x][y] ==
        solution[x][y] = 1;
        return true;
    }

    if (isSafe(maze, x, y)) {
```

```
        // Mark x, y as part of the solution path
        solution[x][y] = 1;

        if (solveMazeUtil(maze, x + 1, y, solution))
            return true;

        if (solveMazeUtil(maze, x, y + 1, solution))
            return true;

        // If none of the above movements work, then BACKTRACK:
        // unmark x, y as part of the solution path
        solution[x][y] = 0;
        return false;
    }

    return false;
}

bool solveMaze(const vector<vector<int>>& maze) {
    vector<vector<int>> solution(maze.size(), vector<int>(maze[0].size(),

    if (!solveMazeUtil(maze, 0, 0, solution)) {
        cout << "Solution doesn't exist";
        return false;
    }

    printSolution(solution);
    return true;
}

int main() {
    int n;
//    cout << "Enter the number of rows: ";
    cin >> n;


    vector<vector<int>> maze(n, vector<int>(n));

 //   cout << "Enter the maze (0 = wall, 1 = path):\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> maze[i][j];
        }
    }

    cout << "Given Maze: " << endl;
    for (const auto &row : maze) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }

    cout << "\nSolution: \n";
    if (!solveMaze(maze)) {
        cout << "No solution exists." << endl;
    }

    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	4 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1	Given Maze: 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 Solution: 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 1
2	2 1 0 1 0	Given Maze: 1 0 1 0 Solution: Solution doesn't existNo solution exists.
3	3 1 1 0 1 1 1 0 1 1	Given Maze: 1 1 0 1 1 1 0 1 1 Solution: 1 0 0 1 1 0 0 1 1
4	4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Given Maze: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Solution: 1 0 0 0 1 0 0 0 1 0 0 0 1 1 1 1
5	3 1 1 0 1 0 1 1 1 1	Given Maze: 1 1 0 1 0 1 1 1 1 Solution: 1 0 0 1 0 0 1 1 1
6	2 0 0 0 0	Given Maze: 0 0 0 0 Solution: Solution doesn't existNo solution exists.

White List:

Black List:
