Rajan Gautam
19BCP101

Div. II, CE 19

SOT, PDEU

Pandit Deendayal Energy University

School of Technology

Design & Analysis of Algorithm (20CP209P)

B. Tech - Computer Science & Engineering (Sem-IV)

Table of Contents

Lab 6 Assignment: Solving Optimization Problems through Backtracking technique and draw state-space diagram.

AIM: **To Solve Optimization Problems through Backtracking technique and draw state-space diagram.**

Problem 1: **Write a C/C++ program to solve given sudoku puzzle using Backtracking Approach. Analyze its time complexity.**

	3			1			6	
7	5			3			4	8
		6	9	8	4	3		
		3				8		
9	1	2				6	7	4
		4				5		
		1	6	7	5	2		
6	8			9			1	5
	9			4			3	

http://printablesudoku.blogspot.com

CODE:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 9

// Function to print result
void PrintSolution(int sudokuBoard[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf(" %d ", sudokuBoard[i][j]);
        }
        printf("\n");
    }
}

// Function to check whether it's legal or not
bool SudokuCheck(int sudokuBoard[N][N], int row, int col, int num)
```

```
{
    // If we find the same number in similar row, then we will return 0
    for (int i = 0; i < N; i++)
    {
        if (sudokuBoard[row][i] == num)
        {
            return false;
        }
    }
    // If we find the same number in similar column, then we will return 0
    for (int i = 0; i < N; i++)
    {
        if (sudokuBoard[i][col] == num)
        {
            return false;
        }
    }
    // If we find the same number in particular 3x3 matrix, then we will return 0
    int startRow = row - row % 3;
    int startCol = col - col % 3;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (sudokuBoard[i + startRow][j + startCol] == num)
            {
                return 0;
            }
        }
    }
    return true;
}

// Funtion to solve Sudoku puzzle
bool SudokuSolver(int sudokuBoard[N][N], int row, int col)
{
    // If we have reached the 8th row and 8th column, we are returing true to avoid backtracking
    if (row == N - 1 && col == N)
    {
        return true;
    }
    // If column becomes 9th, we set it to 0 and move to next row
    if (col == N)
```

```
    {
        row++;
        col = 0;
    }
    // If current position value > 0, we iterate for next column
    if (sudokuBoard[row][col] > 0)
    {
        return SudokuSolver(sudokuBoard, row, col + 1);
    }

    for (int num = 1; num <= N; num++)
    {
        if (SudokuCheck(sudokuBoard, row, col, num))
        {
            // Assigning the number
            sudokuBoard[row][col] = num;

            //Checking for possibility with next column
            if (SudokuSolver(sudokuBoard, row, col + 1))
            {
                return true;
            }
        }
        // Removing assigned number since our assumption was wrong. We will go for next assumption with different value
        sudokuBoard[row][col] = 0;
    }
    return false;
}

int main()
{
    int unsolvedSoduku[N][N] = {{0, 3, 0, 0, 1, 0, 0, 6, 0},
                                 {7, 5, 0, 0, 3, 0, 0, 4, 8},
                                 {0, 0, 6, 9, 8, 4, 3, 0, 0},
                                 {0, 0, 3, 0, 0, 0, 8, 0, 0},
                                 {9, 1, 2, 0, 0, 0, 6, 7, 4},
                                 {0, 0, 4, 0, 0, 0, 5, 0, 0},
                                 {0, 0, 1, 6, 7, 5, 2, 0, 0},
                                 {6, 8, 0, 0, 9, 0, 0, 1, 5},
                                 {0, 9, 0, 0, 4, 0, 0, 3, 0}};

    if (SudokuSolver(unsolvedSoduku, 0, 0) == true)
    {
```

```c
        printf("\nThe Solved Soduku Puzzle will look like: \n\n");
        PrintSolution(unsolvedSoduku);
    }
    else
    {
        printf("No Solution Exists");
    }

    return 0;
}

```

OUTPUT:

```
PROBLEMS   OUTPUT   TERMINAL   ...        1: Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm> cd "f:\Programs\
C Programming\Projects\Design and Analysis of Algorithm\Backtracking\" ; if ($?) { gcc s
udoku.c -o sudoku } ; if ($?) { .\sudoku }

The Solved Soduku Puzzle will look like:

 4  3  8  5  1  7  9  6  2
 7  5  9  2  3  6  1  4  8
 1  2  6  9  8  4  3  5  7
 5  7  3  4  6  9  8  2  1
 9  1  2  8  5  3  6  7  4
 8  6  4  7  2  1  5  9  3
 3  4  1  6  7  5  2  8  9
 6  8  7  3  9  2  4  1  5
 2  9  5  1  4  8  7  3  6
PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm\Backtracking>
```

TIME COMPLEXITY:

Time Complexity of Sudoku Puzzle using Backtracking → O(N^M)

where N = Size of Matrix (Generally 9 in classic Sudoku)

M = Number of blank spaces

Problem-2: **Write a C/C++ program to solve 8-Queen's problem. Analyze the time required by algorithm using Backtracking to solve the problem.**

CODE:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 8

// Function to print result
void PrintSolution(int chessBoard[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf(" %d ", chessBoard[i][j]);
        }
        printf("\n");
    }
}

// Function to check whether it's a safe move or not
bool SafeMove(int chessBoard[N][N], int row, int col)
{
    int i, j;
    // Checking for the left column
    for (i = 0; i < col; i++)
    {
        if (chessBoard[row][i])
        {
            return false;
        }
    }
    // Checking for the left top diagonal
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
    {
        if (chessBoard[i][j])
        {
            return false;
        }
    }
    // Checking for the left bottom diagonal
```

```
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
    {
        if (chessBoard[i][j])
        {
            return false;
        }
    }
    return true;
}

// Function to solve N Queen problem
bool QueenSolver(int chessBoard[N][N], int col)
{
    // If all Queens are already placed at correct position, then simply return true
    if (col >= N)
    {
        return true;
    }

    // Consider the column at 0th position and try placing the Queen in all rows one by one
    for (int i = 0; i < N; i++)
    {
        // If safe, place the Queen
        if (SafeMove(chessBoard, i, col))
        {
            chessBoard[i][col] = 1;

            // Recursive call to other Queens
            if (QueenSolver(chessBoard, col + 1))
            {
                return true;
            }
            chessBoard[i][col] = 0;
        }
    }
    return false;
}

int main()
{
    int board[N][N] = {{0, 0, 0, 0, 0, 0, 0, 0},
                       {0, 0, 0, 0, 0, 0, 0, 0},
                       {0, 0, 0, 0, 0, 0, 0, 0},
                       {0, 0, 0, 0, 0, 0, 0, 0},
```

```c
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0},
                        {0, 0, 0, 0, 0, 0, 0, 0}};

    if (QueenSolver(board, 0) == true)
    {
        printf("\nThe Queens can be placed where there are 1s \n\n");
        PrintSolution(board);
    }
    else
    {
        printf("Solution does not exist");
        return false;
    }
    return 0;
}
```

OUTPUT:

PROBLEMS OUTPUT TERMINAL ··· 1: Code

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm> cd "f:\P
ueen.c -o queen } ; if ($?) { .\queen }

The Queens can be placed where there are 1s

 1  0  0  0  0  0  0  0
 0  0  0  0  0  0  1  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  1
 0  1  0  0  0  0  0  0
 0  0  0  1  0  0  0  0
 0  0  0  0  0  1  0  0
 0  0  1  0  0  0  0  0
PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm\Backtracking>
```

TIME COMPLEXITY:

Time Complexity of N Queen's Problem ==> O(N!)

Problem 3: Write a C/C++ program to solve Sum of subset problem for given array A= {7,3,2,5,8}, Sum=14. Print all the subsets for the given sum.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define N 5

int totalNodes;

void PrintSolution(int A[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf("%*d", N, A[i]);
    }
    printf("\n");
}

/*
    A -> Set Vector
    V -> Tuplet Vector
    ASize -> Set Size
    VSize -> Tuplet Size so Far
    Sum -> Sum so far
    Nodes -> Nodes Count
    m -> Sum to be found
*/

void subsetSum(int A[], int V[], int ASize, int VSize, int sum, int Nodes, int m)
{
    totalNodes++;
    if (m == sum)
    {
        // We found subset
        PrintSolution(V, VSize);

        // Consider next item to find another combination
        subsetSum(A, V, ASize, VSize - 1, sum - A[Nodes], Nodes + 1, m);
        return;
    }
```

```
    else
    {
        // Generate Nodes along the breadth
        for (int i = Nodes; i < ASize; i++)
        {
            V[VSize] = A[i];

            // Consider next level node along depth
            subsetSum(A, V, ASize, VSize + 1, sum + A[i], i + 1, m);
        }
    }
}

// A Function to generate subset
void SubsetGenerator(int A[], int size, int m)
{
    int *TupletVector = (int *)malloc(size * sizeof(int));

    subsetSum(A, TupletVector, size, 0, 0, 0, m);

    free(TupletVector);
}

int main()
{
    // Getting the input weights
    int Weight[] = {7, 3, 2, 5, 8};

    SubsetGenerator(Weight, N, 14);

    printf("Total Nodes Generated: %d", totalNodes);

    return 0;
}

```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL            1: Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm> cd "f:\Programs\C P
rogramming\Projects\Design and Analysis of Algorithm\Backtracking\" ; if ($?) { gcc subsum.
c -o subsum } ; if ($?) { .\subsum }
    7    2    5
Total Nodes Generated: 32
PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm\Backtracking>
```

TIME COMPLEXITY:

Time Complexity of N Queen's Problem ==> $O(2^N)$

Link: https://github.com/rgautam320/Design-and-Analysis-of-Algorithm-Lab/tree/master/Lab_6_Backtracking