Details

Students

Roll	Name
2451033	Abhishek Biswas
2451035	Aman Kumar
2451040	Sougata Patra
2451049	Aditiya Raj

Problem Statement

WAP in C to create the following data structure: Dancing Links

Acknowledgement

Firstly, we would like to thank our family and friends, for their constant support throughout out university life.

Moreover, we would like to express our sincere gratitude to the Department of Computer Science, for allowing us to apply our expertise in this assignment, and also for helping us develop the required knowledge to program using the C programming language for this project.

Introduction

Algorithm

It is a step-by-step procedure to solve a problem in finite time.

```
graph LR
i[/Input/] -->
Algorithm -->
o[/Output/]
```

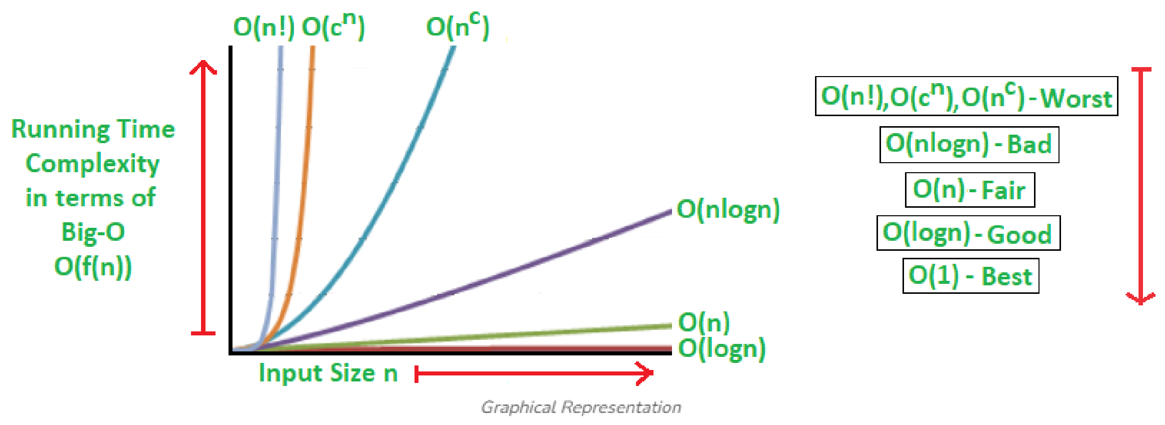
Complexity

Run-time complexity refers to the amount of time it takes to run an algorithm. The goal of a program is to maximize the best-case scenario and minimize the worst-case scenario.

The worst-case scenario is measured using order of complexity. The most common notation for representing this order of complexity is Big-Oh notation O(g(n)).

Some common orders of complexity are:-

$$O(1) < O(log n) < O(n) < O(n*log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

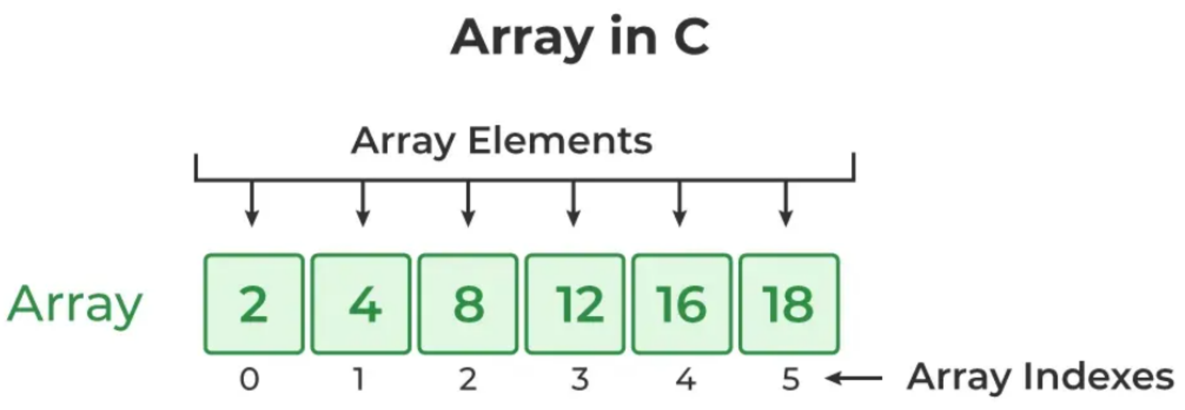


Data Structure

It is a way of organizing and storing data in a computer so that it can be accessed and used efficiently.

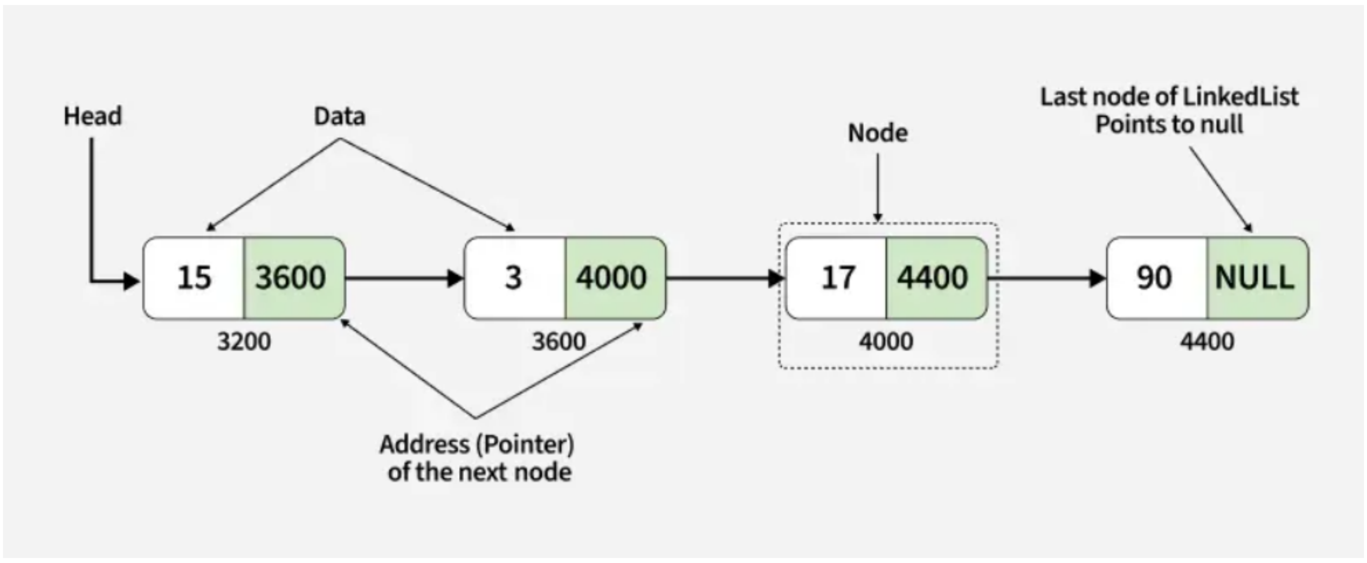
Array

Array is a linear data structure where all elements are arranged sequentially.



Linked List

A linked list is a linear data structure used for storing a sequence of elements, where each element is stored in a node that contains both the element and a pointer to the next node in the sequence.


Head
Data
Node
Last node of LinkedList Points to null
15
3600
3200
3
4000
3600
17
4400
4000
90
NULL
4400
Address (Pointer) of the next node

Tree

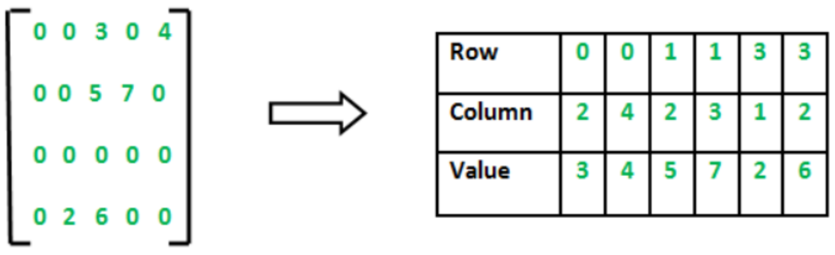
It is a non-linear hierarchical data structure, where there is a single root node and every other node contains a parent and/or child. Linear data structures like arrays, stacks, queues, and linked list can be traversed in only one way. But a hierarchical data structure like a tree can be traversed in different ways.

```
flowchart TB
a --> b & c
b --> d & e
c --> f & g
```

Sparse Matrix

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.

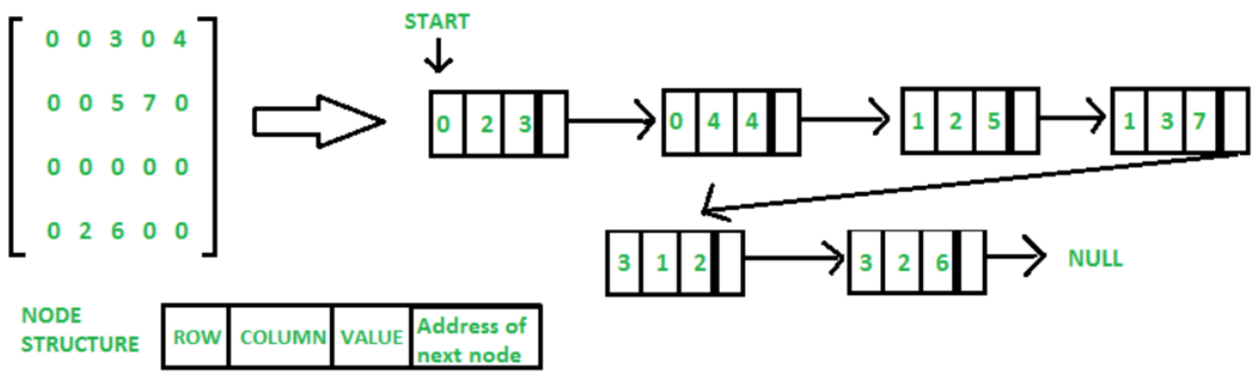
Using Arrays



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

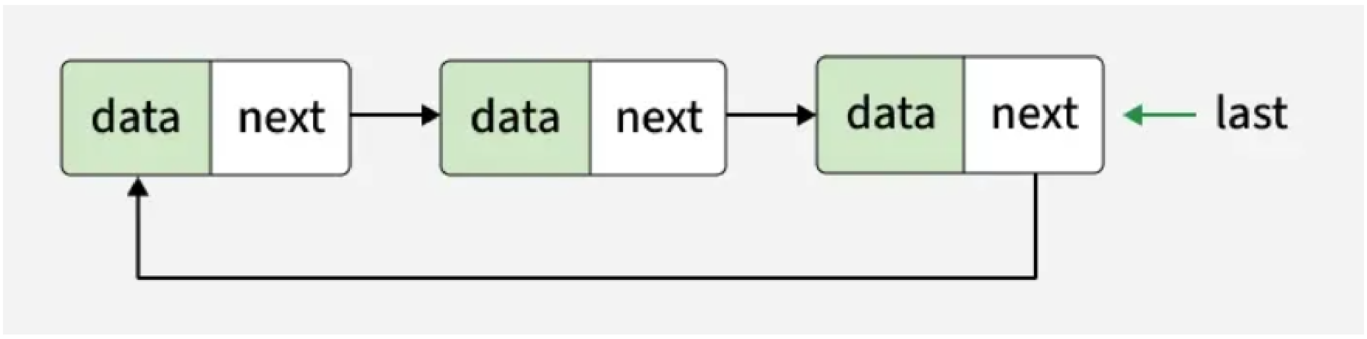
Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Using Linked List



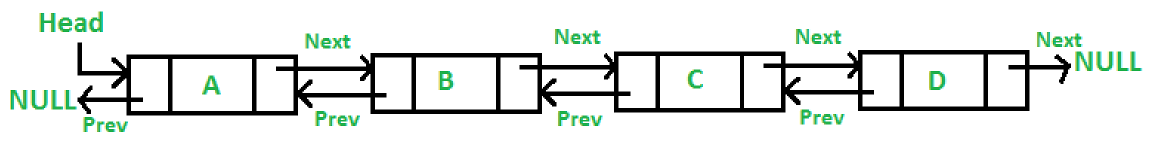
Circular Linked List

In Circular Singly Linked List, each node has just one pointer called the "next" pointer. The next pointer of the last node points back to the first node and this results in forming a circle. In this type of Linked list, we can only move through the list in one direction.



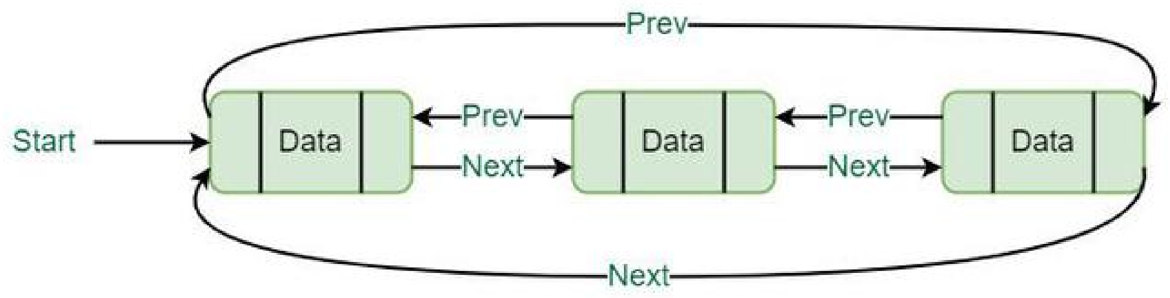
Doubly Linked List

A doubly linked list is a type of linked list in which each node contains 3 parts, a data part and two addresses, one points to the previous node and one for the next node. It differs from the singly linked list as it has an extra pointer called previous that points to the previous node, allowing the traversal in both forward and backward directions.



Circular Doubly Linked List

A circular doubly linked list is defined as a circular linked list in which each node has two links connecting it to the previous node and the next node.



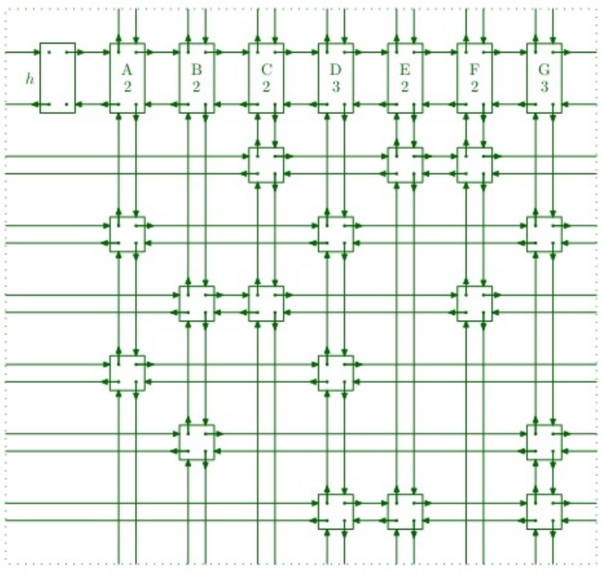
Circular doubly linked list

Dancing Links

Dancing Links is a technique for implementing backtracking algorithms, like Algorithm X, that efficiently solve problems such as Sudoku or Exact Cover by using sparse matrix represented with circular doubly linked lists.

$$\begin{matrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{matrix}$$

The 4-way linked matrix will look like this -



Four way linked matrix

Program

It is the implementation of algorithm in a programming language.

This project uses C programming language for its implementation. C is a programming language that is used globally to develop various application software.

Project

Repository

The code base and this documentation is available on a Github Repository

Algorithms

Algorithm	Complexity
`initialize_dlx(DLX *dlx, int rows, int cols)`	$O(n)$
`insert_node(DLX *dlx, int row, int col)`	$O(n)$
`cover_column(Node *col)`	$O(n)$
`uncover_column(Node *col)`	$O(n)$
`search(DLX *dlx, int k)`	$O(1)$
`print_matrix(DLX *dlx)`	$O(1)$
`print_solutions(DLX *dlx)`	$O(1)$
`input_matrix(DLX *dlx)`	$O(1)$

Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Maximum size for the matrix (adjust as needed)
#define MAX_ROWS 100
#define MAX_COLS 100
#define MAX_NODES 10000

// Node structure for the Dancing Links matrix
typedef struct Node {
    struct Node *left, *right, *up, *down, *column;
    int row_id, col_id, node_count;
} Node;

// Structure to hold the Dancing Links matrix and solutions
typedef struct {
    Node *root;
    Node *headers[MAX_COLS];
    Node nodes[MAX_NODES];
    int node_count;
    int rows, cols;
    int solutions[MAX_ROWS][MAX_COLS];
    int solution_count;
} DLX;

// Function prototypes
void initialize_dlx(DLX *dlx, int rows, int cols);
void insert_node(DLX *dlx, int row, int col);
void cover_column(Node *col);
void uncover_column(Node *col);
void search(DLX *dlx, int k);
void print_matrix(DLX *dlx);
void print_solutions(DLX *dlx);
void input_matrix(DLX *dlx);

// Initialize the Dancing Links matrix
void initialize_dlx(DLX *dlx, int rows, int cols) {
    dlx->rows = rows;
    dlx->cols = cols;
    dlx->node_count = 0;
    dlx->solution_count = 0;
    dlx->root = &dlx->nodes[dlx->node_count++];
    dlx->root->left = dlx->root->right = dlx->root;
    dlx->root->up = dlx->root->down = dlx->root;

    // Initialize column headers
    for (int i = 0; i < cols; i++) {
        dlx->headers[i] = &dlx->nodes[dlx->node_count++];
        dlx->headers[i]->node_count = 0;
        dlx->headers[i]->col_id = i;
        dlx->headers[i->left = dlx->root->left;
        dlx->headers[i]->right = dlx->root;
        dlx->root->left->right = dlx->headers[i];
        dlx->root->left = dlx->headers[i];
        dlx->headers[i]->up = dlx->headers[i];
        dlx->headers[i]->down = dlx->headers[i];
    }
}

// Insert a node at (row, col) in the matrix
void insert_node(DLX *dlx, int row, int col) {
    Node *node = &dlx->nodes[dlx->node_count++];
    node->row_id = row;
    node->col_id = col;
    node->column = dlx->headers[col];

    // Link vertically
    node->down = dlx->headers[col];
    node->up = dlx->headers[col]->up;
    dlx->headers[col]->up->down = node;
    dlx->headers[col]->up = node;

    // Link horizontally
    Node *row_start = node;
    if (dlx->node_count > 1 && dlx->nodes[dlx->node_count-2].row_id == row) {
        row_start = &dlx->nodes[dlx->node_count-2];
        node->left = row_start;
        node->right = row_start->right;
        row_start->right->left = node;
        row_start->right = node;
    } else {
        node->left = node;
        node->right = node;
    }

    dlx->headers[col]->node_count++;
}

// Cover a column (remove it from consideration)
void cover_column(Node *col) {
    col->right->left = col->left;
    col->left->right = col->right;
    for (Node *i = col->down; i != col; i = i->down) {
        for (Node *j = i->right; j != i; j = j->right) {
            j->down->up = j->up;
            j->up->down = j->down;
            j->column->node_count--;
        }
    }
}

// Uncover a column (restore it)
void uncover_column(Node *col) {
    for (Node *i = col->up; i != col; i = i->up) {
        for (Node *j = i->left; j != i; j = j->left) {
            j->column->node_count++;
            j->down->up = j;
            j->up->down = j;
        }
    }
    col->right->left = col;
    col->left->right = col;
}

// Search for solutions using Algorithm X
void search(DLX *dlx, int k) {
    if (dlx->root->right == dlx->root) {
        dlx->solution_count++;
        return;
    }

    Node *col = dlx->root->right;
    cover_column(col);

    for (Node *r = col->down; r != col; r = r->down) {
        dlx->solutions[dlx->solution_count][k] = r->row_id;
        for (Node *j = r->right; j != r; j = j->right) {
            cover_column(j->column);
        }
        search(dlx, k + 1);
        for (Node *j = r->left; j != r; j = j->left) {
            uncover_column(j->column);
        }
    }
    uncover_column(col);
```

```
}

// Print the current matrix
void print_matrix(DLX *dlx) {
    printf("\nCurrent Matrix (%d rows x %d cols):\n", dlx->rows, dlx->cols);
    int matrix[MAX_ROWS][MAX_COLS] = {0};
    for (int i = 0; i < dlx->node_count; i++) {
        if (dlx->nodes[i].row_id >= 0) {
            matrix[dlx->nodes[i].row_id][dlx->nodes[i].col_id] = 1;
        }
    }
    for (int i = 0; i < dlx->rows; i++) {
        for (int j = 0; j < dlx->cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

// Print all found solutions
void print_solutions(DLX *dlx) {
    printf("\nFound %d solution(s):\n", dlx->solution_count);
    for (int i = 0; i < dlx->solution_count; i++) {
        printf("Solution %d: Rows selected = ", i + 1);
        for (int j = 0; j < dlx->rows && dlx->solutions[i][j] != 0; j++) {
            printf("%d ", dlx->solutions[i][j]);
        }
        printf("\n");
    }
}

// Input matrix from user
void input_matrix(DLX *dlx) {
    int rows, cols;
    printf("Enter number of rows (max %d): ", MAX_ROWS);
    scanf("%d", &rows);
    printf("Enter number of columns (max %d): ", MAX_COLS);
    scanf("%d", &cols);
    if (rows <= 0 || cols <= 0 || rows > MAX_ROWS || cols > MAX_COLS) {
        printf("Invalid dimensions!\n");
        return;
    }

    initialize_dlx(dlx, rows, cols);
    printf("Enter the matrix (%d x %d, 0s and 1s only):\n", rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            int value;
            scanf("%d", &value);
            if (value == 1) {
                insert_node(dlx, i, j);
            }
        }
    }
    printf("Matrix input complete.\n");
}

int main() {
    DLX dlx;
    int choice;

    printf("Dancing Links Algorithm for Exact Cover Problem\n");
    while (true) {
        printf("\nMenu:\n");
        printf("1. Input a new matrix\n");
        printf("2. Solve the Exact Cover problem\n");
        printf("3. Display current matrix\n");
        printf("4. Display all solutions\n");
        printf("5. Exit\n");
        printf("Enter your choice (1-5): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                input_matrix(&dlx);
                break;
            case 2:
                dlx.solution_count = 0;
                search(&dlx, 0);
                printf("Solving complete. Found %d solution(s).\n",
dlx.solution_count);
                break;
            case 3:
                print_matrix(&dlx);
                break;
            case 4:
                print_solutions(&dlx);
                break;
            case 5:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice! Please enter 1-5.\n");
        }
    }
    return 0;
}
```

Output

Simple Exact Cover Problem

```
Dancing Links Algorithm for Exact Cover Problem

Menu:
```

```
1. Input a new matrix
2. Solve the Exact Cover problem
3. Display current matrix
4. Display all solutions
5. Exit
Enter your choice (1-5): 1
Enter number of rows (max 100): 3
Enter number of columns (max 100): 4
Enter the matrix (3 x 4, 0s and 1s only):
1 0 1 0
0 1 1 1
1 1 0 0
Matrix input complete.

Menu:
Enter your choice (1-5): 3
Current Matrix (3 x 4):
1 0 1 0
0 1 1 1
1 1 0 0

Menu:
Enter your choice (1-5): 2
Solving complete. Found 1 solution(s).

Menu:
Enter your choice (1-5): 4
Found 1 solution(s):
Solution 1: Rows selected = 0 2

Menu:
Enter your choice (1-5): 5
Exiting program.
```

No Solution

```
Dancing Links Algorithm for Exact Cover Problem

Menu:
Enter your choice (1-5): 1
Enter number of rows (max 100): 2
Enter number of columns (max 100): 3
Enter the matrix (2 x 3, 0s and 1s only):
1 0 0
0 1 0
Matrix input complete.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 0 solution(s).
```

```
Menu:
Enter your choice (1-5): 4
Found 0 solution(s):

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Multiple Solutions

```
Dancing Links Algorithm for Exact Cover Problem

Menu:
Enter your choice (1-5): 1
Enter number of rows (max 100): 3
Enter number of columns (max 100): 3
Enter the matrix (3 x 3, 0s and 1s only):
1 0 1
0 1 1
1 1 0
Matrix input complete.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 2 solution(s).

Menu:
Enter your choice (1-5): 4
Found 2 solution(s):
Solution 1: Rows selected = 0
Solution 2: Rows selected = 1 2

Menu:
Enter your choice (1-5): 5
Exiting program.
```

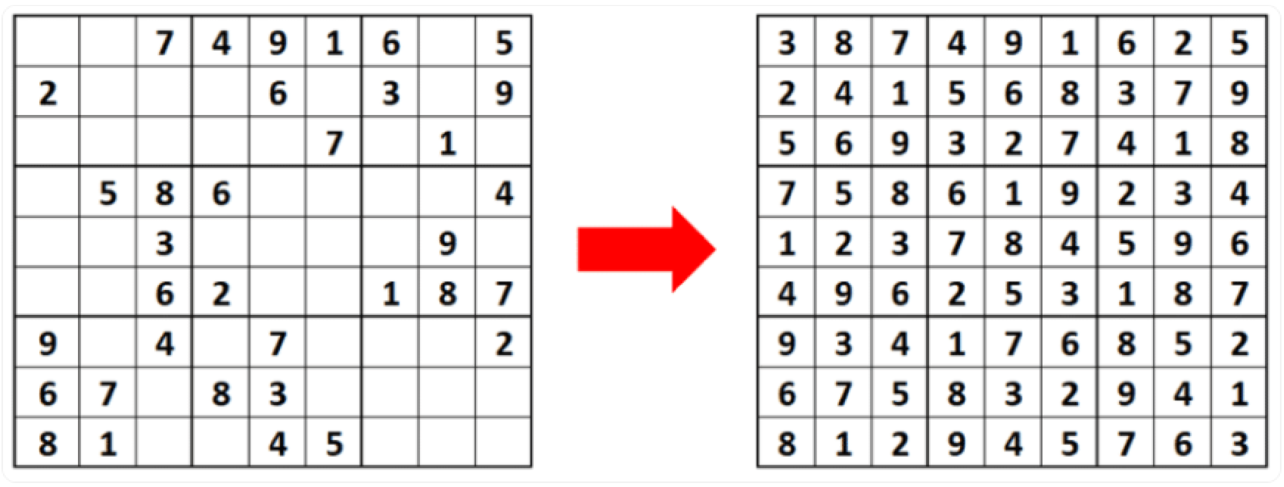
Invalid Input

```
Dancing Links Algorithm for Exact Cover Problem

Menu:
Enter your choice (1-5): 1
Enter number of rows (max 100): 101
Enter number of columns (max 100): 4
Invalid dimensions!

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Application Puzzles

Sudoku

Sudoku is a popular puzzle game where the goal is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids contain all of the digits from 1 to 9. In this article, we will learn how we can solve Sudoku puzzles using C programming language.



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

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

Left: *An unsolved Sudoku puzzle.* ***Right:*** *Solution to this Sudoku puzzle.*

Source Code: https://github.com/AbhishekBiswas76/Semester-Project/blob/main/code/sudoku.c

Output

Valid Sudoku with One Solution

```
Sudoku Solver Using Dancing Links (Algorithm X)

Menu:
1. Input a new Sudoku puzzle
2. Solve the Sudoku puzzle
3. Display current Sudoku grid
4. Display solution(s)
5. Exit
Enter your choice (1-5): 1
Enter the 9x9 Sudoku grid (0 for empty cells, 1-9 for filled cells):
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
```

```
0 0 0 0 8 0 0 7 9
Sudoku input complete.

Menu:
Enter your choice (1-5): 3
Current Sudoku Grid:
---------------------
| 5 3 0 | 0 7 0 | 0 0 0 |
| 6 0 0 | 1 9 5 | 0 0 0 |
| 0 9 8 | 0 0 0 | 0 6 0 |
---------------------
| 8 0 0 | 0 6 0 | 0 0 3 |
| 4 0 0 | 8 0 3 | 0 0 1 |
| 7 0 0 | 0 2 0 | 0 0 6 |
---------------------
| 0 6 0 | 0 0 0 | 2 8 0 |
| 0 0 0 | 4 1 9 | 0 0 5 |
| 0 0 0 | 0 8 0 | 0 7 9 |
---------------------

Menu:
Enter your choice (1-5): 2
Solving complete. Found 1 solution(s).

Menu:
Enter your choice (1-5): 4
Found 1 solution(s):
Solution 1:
---------------------
| 5 3 4 | 6 7 8 | 9 1 2 |
| 6 7 2 | 1 9 5 | 3 4 8 |
| 1 9 8 | 3 4 2 | 5 6 7 |
---------------------
| 8 5 9 | 7 6 1 | 4 2 3 |
| 4 2 6 | 8 5 3 | 7 9 1 |
| 7 1 3 | 9 2 4 | 8 5 6 |
---------------------
| 9 6 1 | 5 3 7 | 2 8 4 |
| 2 8 7 | 4 1 9 | 6 3 5 |
| 3 4 5 | 2 8 6 | 1 7 9 |
---------------------

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Unsolvable Sudoku

```

Sudoku Solver Using Dancing Links (Algorithm X)

Menu:
```

```
Enter your choice (1-5): 1
Enter the 9x9 Sudoku grid (0 for empty cells, 1-9 for filled cells):
5 5 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 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 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
Sudoku input complete.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 0 solution(s).

Menu:
Enter your choice (1-5): 4
No solutions found.

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Empty Sudoku (Multiple Solutions, Limited to One)

```
Sudoku Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter the 9x9 Sudoku grid (0 for empty cells, 1-9 for filled cells):
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 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 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
Sudoku input complete.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 1 solution(s).

Menu:
Enter your choice (1-5): 4
Found 1 solution(s):
Solution 1:
```

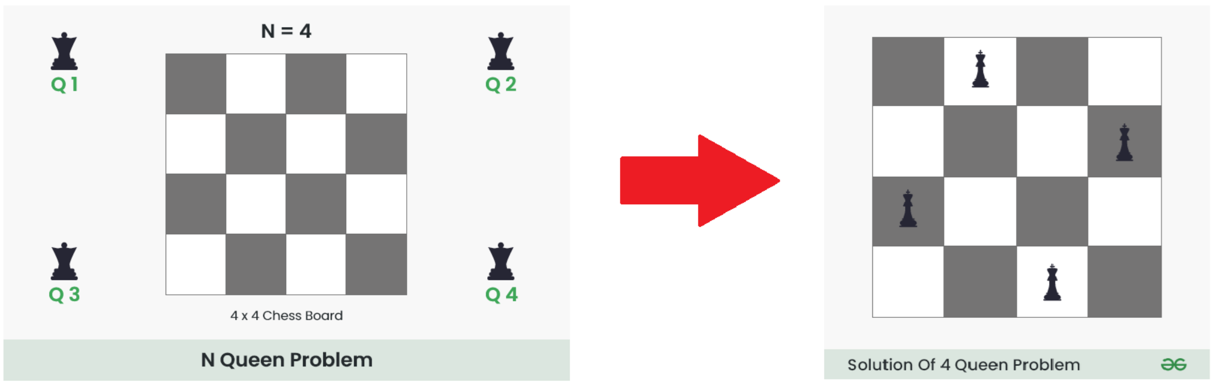
```
---------------------
| 1 2 3 | 4 5 6 | 7 8 9 |
| 4 5 6 | 7 8 9 | 1 2 3 |
| 7 8 9 | 1 2 3 | 4 5 6 |
---------------------
| 2 1 4 | 5 6 7 | 8 9 3 |
| 5 6 7 | 8 9 3 | 2 1 4 |
| 8 9 3 | 2 1 4 | 5 6 7 |
---------------------
| 3 4 5 | 6 7 8 | 9 1 2 |
| 6 7 8 | 9 1 2 | 3 4 5 |
| 9 1 2 | 3 4 5 | 6 7 8 |
---------------------

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Invalid Input

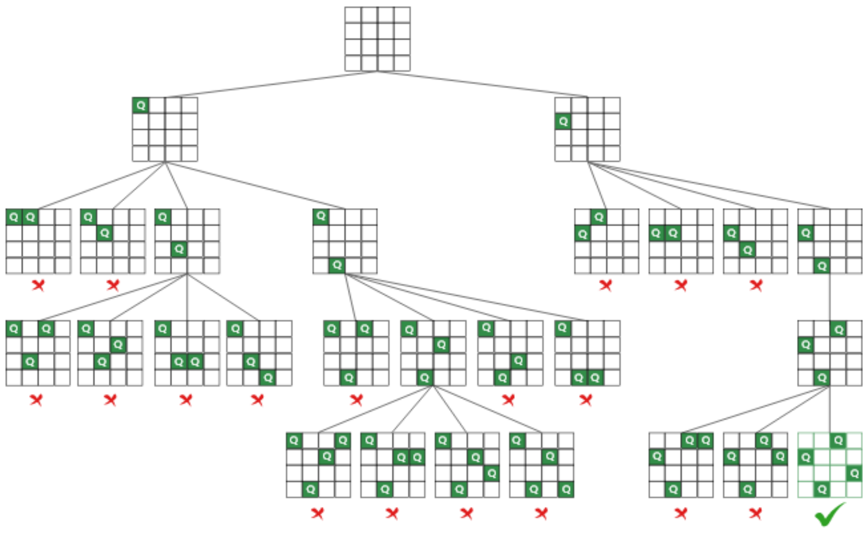
```
Sudoku Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter the 9x9 Sudoku grid (0 for empty cells, 1-9 for filled cells):
10 0 0 0 0 0 0 0 0
Invalid input! Use 0-9 only.
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 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 0 0 0 0 0 0 0 0
Sudoku input complete.

Menu:
Enter your choice (1-5): 3
Current Sudoku Grid:
---------------------
| 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 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 0 0 | 0 0 0 | 0 0 0 |
| 0 0 0 | 0 0 0 | 0 0 0 |
---------------------

Menu:
Enter your choice (1-5): 5
Exiting program.
```

N Queens

The n-queens puzzle is the problem of placing n queens on a (n × n) chessboard such that no two queens can attack each other. Note that two queens attack each other if they are placed on the same row, the same column, or the same diagonal.



Recursive Tree Approach



Source Code: https://github.com/AbhishekBiswas76/Semester-Project/blob/main/code/n-queens.c

Output

4-Queens (Multiple Solutions)

```
N-Queens Solver Using Dancing Links (Algorithm X)

Menu:
1. Input board size (N)
2. Solve N-Queens problem
3. Display current board
4. Display all solutions
5. Exit
Enter your choice (1-5): 1
Enter board size N (1 to 20): 4
Board size 4x4 set.

Menu:
Enter your choice (1-5): 3
Current 4x4 Board:
. . . .
. . . .
. . . .
. . . .

Menu:
Enter your choice (1-5): 2
Solving complete. Found 2 solution(s).

Menu:
Enter your choice (1-5): 4
Found 2 solution(s) for 4x4 N-Queens:
Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .

Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .

Menu:
Enter your choice (1-5): 5
Exiting program.
```

3-Queens (No Solutions)

```
N-Queens Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter board size N (1 to 20): 3
```

```
Board size 3x3 set.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 0 solution(s).

Menu:
Enter your choice (1-5): 4
Found 0 solution(s) for 3x3 N-Queens:

Menu:
Enter your choice (1-5): 5
Exiting program.
```

8-Queens (Larger Board)

```
N-Queens Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter board size N (1 to 20): 8
Board size 8x8 set.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 92 solution(s).

Menu:
Enter your choice (1-5): 4
Found 92 solution(s) for 8x8 N-Queens:
Solution 1:
Q . . . . . . .
. . . . Q . . .
. . . . . . Q .
. . Q . . . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .

Solution 2:
Q . . . . . . .
. . . . . Q . .
. . . . . . . Q
. . Q . . . . .
. . . . . . Q .
. . . . Q . . .
. Q . . . . . .
. . . Q . . . .

[... 90 more solutions ...]
```

```
Menu:
Enter your choice (1-5): 5
Exiting program.
```

1-Queen (Trivial Solution)

```
N-Queens Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter board size N (1 to 20): 1
Board size 1x1 set.

Menu:
Enter your choice (1-5): 2
Solving complete. Found 1 solution(s).

Menu:
Enter your choice (1-5): 4
Found 1 solution(s) for 1x1 N-Queens:
Solution 1:
Q

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Invalid Input

```
N-Queens Solver Using Dancing Links (Algorithm X)

Menu:
Enter your choice (1-5): 1
Enter board size N (1 to 20): 21
Invalid board size! Must be between 1 and 20.

Menu:
Enter your choice (1-5): 3
No board size set.

Menu:
Enter your choice (1-5): 2
Please set board size first (Option 1).

Menu:
Enter your choice (1-5): 5
Exiting program.
```

Conclusion

Using C, we were able to connect and apply the knowledge that we have learned in Data Structures and Algorithms. We understood the importance of such concepts which are then used to solve real world problems.

We have built this project from the following concepts below.

DSA

- [x] Array
- [x] Linked List
- [x] Tree
- [x] Sparse Matrix (Both Array and Linked List implementations)
- [x] Circular Linked List
- [x] Doubly Linked List
- [x] Circular Doubly Linked List

References

- "Complete Guide On Complexity Analysis - DSA Tutorial," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/complete-guide-on-complexity-analysis/
- "Arrays in C," *GeeksforGeeks*. https://www.geeksforgeeks.org/c-sharp/arrays-in-c-sharp/
- "Linked List Data Structure," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/linked-list-data-structure/
- "Tree Data Structure," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/tree-data-structure/
- "Sparse Matrix Representations," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/sparse-matrix-representation/
- "Introduction to Circular Linked List," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/circular-linked-list/
- "Doubly Linked List in C," *GeeksforGeeks*. https://www.geeksforgeeks.org/c/doubly-linked-list-in-c/
- "Introduction to Circular Doubly Linked List," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/introduction-to-circular-doubly-linked-list/
- "Implementation of Exact Cover Problem and Algorithm X using DLX," *GeeksforGeeks*. https://www.geeksforgeeks.org/cpp/implementation-of-exact-cover-problem-and-algorithm-x-using-dlx/
- "Sudoku in C," *GeeksforGeeks*. https://www.geeksforgeeks.org/c/sudoku-in-c/
- "N Queen Problem," *GeeksforGeeks*. https://www.geeksforgeeks.org/dsa/n-queen-problem-backtracking-3/

Thank You For Your Attention