Project Based Learning

B.Tech CSE IIIrd Semester
Session 2023-24
Name of the faculty: Dr. Yashu Shanker
Name of the course: Data Structure using C
Name of the activity: Project Based Learning
Students involved: 2nd year, B.Tech CSE

Event Description:

The students must design and implement the allotted project using data structures. Project-based learning's emphasis on practical applications is one of its distinctive qualities. Students can make the connection between what they learn in college and the real world by working on practical projects. PBL enables students to work on real-world issues, enhancing the relevance and meaning of their technical knowledge.

Data structures can be classified into the following basic types:

- Arrays
- Linked Lists
- Stacks
- Queues
- Trees
- Hash tables
- Graphs

Selecting the appropriate setting for your data is an integral part of the programming and problem-solving process. And you can observe that data structures organize abstract data types in concrete implementations. To attain that result, they make use of various algorithms, such as sorting, searching, etc.

Data structures are fundamental building blocks in computer science and programming. They are important tools that helps inorganizing, storing, and manipulating data efficiently. On top of that it provides a way to represent and manage information in a structured manner, which is essential for designing efficient algorithms and solving complex problems.

- It can store variables of various data types.
- It allows the creation of objects that feature various types of attributes.
- It allows reusing the data layout across programs.
- It can implement other data structures like stacks, linked lists, trees, graphs, queues, etc.

Required Skills:

- Basics of Data structure and its algorithms
- C Programming

Outcome of the activity:

- They help to solve complex real-time problems.
- They improve analytical and problem-solving skills.
- They help you to crack technical interviews.
- Topics in data structure can efficiently manipulate the data.

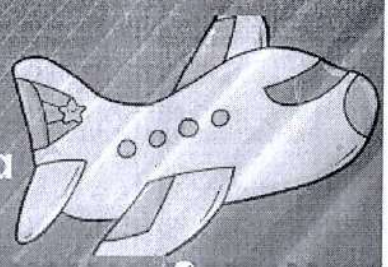
Students participated in the event:

S.No	Enrollment No	Student Name
1	12118002722	Vinay Prakash Suman
2	12218002722	Lakshit Verma
3	12318002722	Nitish Kumar
4	12418002722	Aman Tiwari
5	12518002722	Sarthak Jain
6	12618002722	Deep Shikha
7	12718002722	Gautam Kumar
8	12818002722	Irfan Ansari
9	12918002722	Satyam Kumar
10	13018002722	Ayush Verma
11	13118002722	Aditi Bansal
12	13218002722	Manish Prasad
13	13318002722	Navneet Chandra
14	13418002722	Aparna Tyagi
15	13518002722	Sonu Choudhary
16	13618002722	Paras Yadav
17	13718002722	Jai Chadha
18	13818002722	Aaryan
19	13918002722	Bhavyadeveshwar
20	14018002722	Santoshi Kumari
21	14118002722	Manish Kumar
22	14218002722	Sourabh
23	14318002722	Ishu Kumar
24	14418002722	Kaushal Rout
25	14518002722	Mitesh Kumar Singh
26	14618002722	Amit Kumar
27	14718002722	Kumar Tejaswa
28	35118002722	Harsh Rana
29	35218002722	Ujjwal Kumar
30	35318002722	Shamiullah Khan
31	35418002722	Shivam Baila

32	35518002722	Vaibhav Tiwari
33	35618002722	Jatin Chaudhary
34	35718002722	Lakshya Rautela
35	35818002722	Hemant Singh
36	35918002722	Priyanshu Kumar
37	36018002722	Ankush Pachori
38	36118002722	Harsh Sharma
39	36218002722	Aaditya Chand Ramola
40	36318002722	Anand Kumar Roy
41	36418002722	Kartik Sharma
42	36518002722	Harshvardhan
43	36618002722	Aman Kumar
44	36718002722	Anil
45	70118002722	Samrisha Chauhan
46	70218002722	Mansi
47	70318002722	Siddharth Sharma
48	70418002722	Tisha Choudhary
49	70518002722	Saksham
50	70618002722	Rakshit Malik
51	02718002722	Yashika Pratap Shukla
52	LE	Keshav Yadav
53	LE	Jatin Tanwar
54	LE	Gaurav Kumar
55	LE	Yash Kumar
56	LE	Vishnu Sharma
57	LE	Saksham Mohan Mishra
58	BRANCH CHANGE	Danish
59	BRANCH CHANGE	Riyansh Varshney
60	BRANCH CHANGE	Purwi Aggarwal
61	BRANCH CHANGE	Ashwani Kumar Sharma
62	BRANCH CHANGE	Vanshika Nagpal

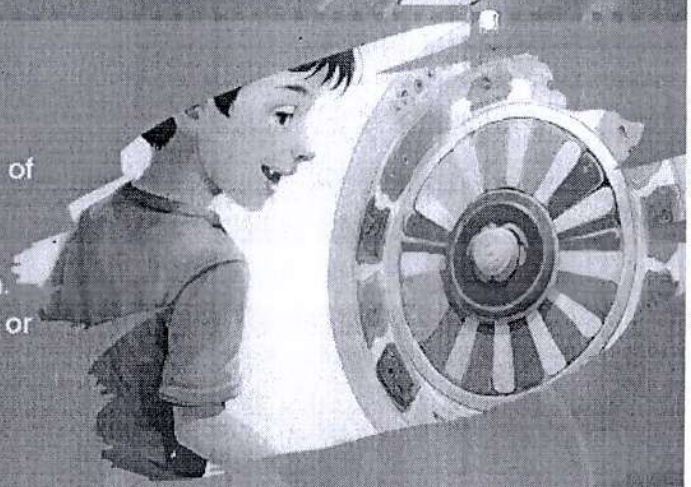
SPIN & WIN

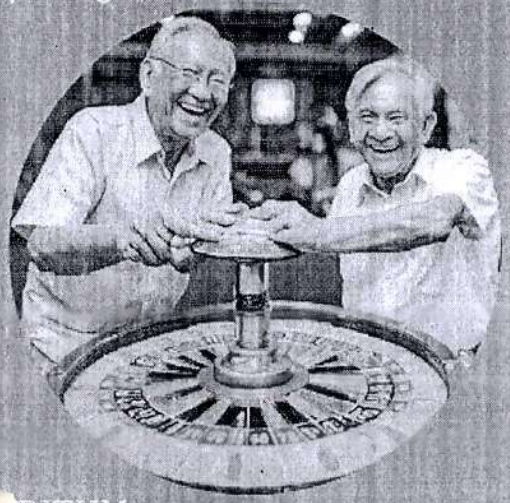
What is Spin & Win?
spin and win is a game of a chance where players spin a wheel to win prizes.



OW DOES IT WORKS?

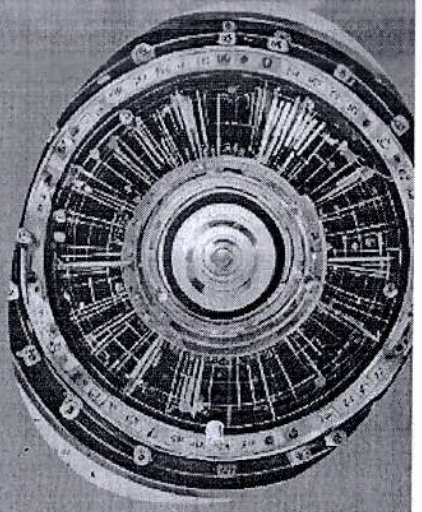
- The player enters the spin and win game.
- The player clicks a button to spin the wheel.
- The software application or website randomly selects one of the prizes or offers from the database.
- The wheel lands on the prize or offer that was selected.
- player wins the prize or offer that the wheel landed on.
- The prize or offer is delivered to the player (e.g., via email, or physical gift).





WHY IT IS FUN?

Spin and win games can be a lot of fun, especially when there are valuable prizes to be won. The excitement of spinning the wheel and seeing what prize you land on can be addictive.



L RITHM.

1. Initialize the wheel with prizes or rewards.
2. Set the player's initial balance to 0.
3. Allow the player to spin the wheel.
4. Generate a random result to determine the wheel's stopping position.
5. Retrieve the reward associated with the stopping position.
6. Display the result and update the player's balance.
7. Repeat steps 3 to 6 until the player decides to quit.
8. Display the player's final balance when they quit.

SUBMITTED BY
AMAN KUMAR
SEC. 'C'
ENROLLMRNT-36618002722

USES OF ARRAY

storing prizes
storing probabilities
Tracking player progress
Generating custom game ru

Affiliated to GGSIP University, New Delhi
Approved by AICTE & Council of Architecture

DELHI TECHNICAL CAMPUS
GREATER NOIDA
Department of Computer Science and Engineering
B-tech 2 Year (CSE)

Maze Game by Breadth First Search Algorithm

Co-ordinator: Dr. Yashu Shanker

Maze Game

Objective

Objective => Maze Frame using Breadth - First Search Algorithm

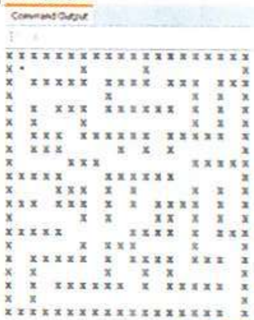
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define ROW 5
#define COL 5

// Structure to represent a cell in the maze
struct Cell {
    int row;
    int col;
};
```

```
int main() {
    int maze[ROW][COL] = {
        {1, 0, 1, 1, 1},
        {1, 1, 1, 0, 1},
        {0, 0, 0, 0, 1},
        {0, 1, 1, 1, 1},
        {1, 1, 1, 1, 1}
    };

    struct Cell src = {0, 0};
    struct Cell dest = {4, 4};

    printf("Maze:\n");
    for (int i = 0; i < ROW; ++i) {
        for (int j = 0; j < COL; ++j) {
            printf("%d ", maze[i][j]);
        }
        printf("\n");
    }
```

Breadth First Search Algorithm

•Breadth First Search (BFS) is an algorithm for Searching a tree data structure for a node that satisfies a given property.

It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.



Breadth-First Search (BFS) algorithm- Step-by-Step

Step 1: Define the Maze and Source/Destination Cells
Step 2: Initialize Data Structures
Step 3: Enqueue the Source Cell
Step 4: Perform BFS
Step 5: Print the Result

Presented By:
Siddharth Sharma
Saksham
Tisha Choudhary

Delhi Technical Campus

Affiliated to GGSIP University, New Delhi • Approved by AICTE & Council of Architecture

GROUP PROJECTS (SUDOKU GAME)

B.TECH (-CSE) 2-YEAR

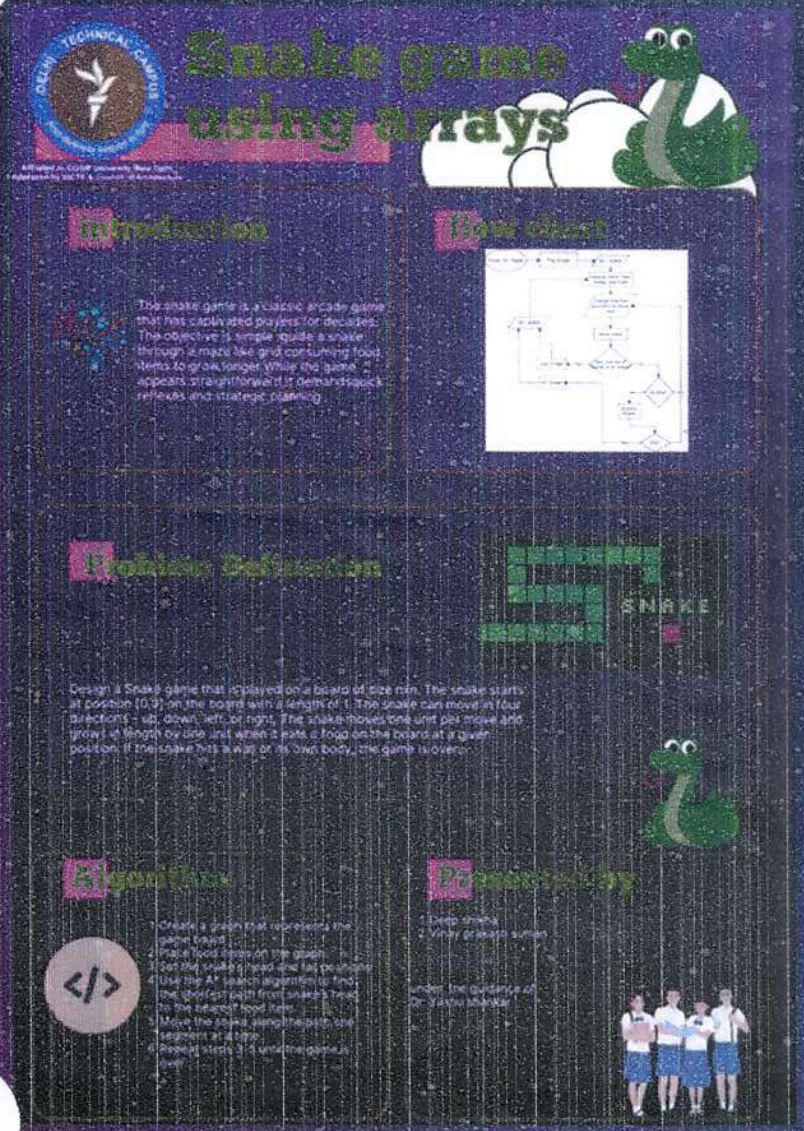
SUDOKU GAME

BACK-TRACKING (ALGORITHM)

CO-ORDINATOR: DR. YASHU SHANKER

PRESENTED BY :

- GAUTAM KUMAR
- LAKSHIT VERMA
- SANTOSHI KUMARI
- SATYAM KUMAR



DELHI TECHNICAL CAMPUS

GREATER NOIDA

Affiliated to GGSIPU and approved by AICTE & COA

SNAKE GAME

PRESENTED TO

YASHU SHANKER

PRESENTED BY

VINAY PRAKASH SUMAN

DEEPSHIKHA

DELHI TECHNICAL CAMPUS
GREATER NOIDA

Affiliated to GGSIPU and approved by AICTE & COA

MAZEGAME

-BY USING BREADTH FIRST SEARCH ALGORITHM

Co-ordinator : Dr. Yashu Shanker

OBJECTIVE

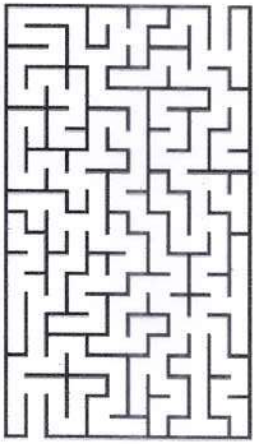
Objective => Maze Frame using Breadth - First Search Algorithm

BREADTH FIRST SEARCH ALGORITHM :

Breadth First Search (BFS) is a graph traversal algorithm that systematically explores all the neighbor nodes at the present depth before moving on to the nodes at the next depth level. It is widely used in pathfinding and puzzle-solving in game design.

KEY STEPS OF THE BFS ALGORITHM:

1. BEGIN WITH THE INITIAL NODE AND ENQUEUE IT INTO A QUEUE.
2. DEQUEUE A NODE FROM THE QUEUE AND MARK IT AS VISITED.
3. ENQUEUE ALL THE UNVISITED NEIGHBORING NODES OF THE CURRENT NODE.
4. REPEAT STEPS 2 AND 3 UNTIL THE QUEUE IS EMPTY.



BREADTH-FIRST SEARCH (BFS) ALGORITHM STEP-BY-STEP

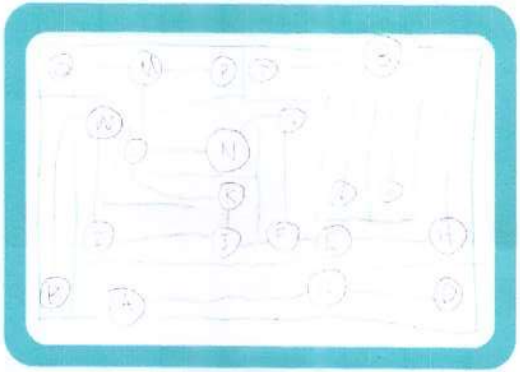
Step 1: Define the Maze and Source/Destination Cells

Step 2: Initialize Data Structures

Step 3: Enqueue the Source Cell

Step 4: Perform BFS

Step 5: Print the Result



```c
#include <stdio.h>

int main() {
    int maze[5][5] = {
        {1, 0, 1, 1, 1},
        {1, 1, 1, 0, 1},
        {0, 0, 0, 0, 1},
        {1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1}
    };
    printf("Maze: \n");
    for (int i = 0; i < 5; ++i) {
        for (int j = 0; j < 5; ++j) {
            printf("%d\t", maze[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```


Presented By:
Vishnu Sharma
(01218007223)

SPIN AND WIN

"Spin and Win" typically refers to a type of game or activity where participants have the opportunity to win prizes by spinning a wheel or a similar mechanism.

Use of Array :
spin and wheel can be implemented using array.
Various values can be stored in an array which then can be acessed by their index.

PSEUDO CODE

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int spinWheel(string prizes[], int numPrizes) {
 // Generate a random index to get a prize from the array
 int index = rand() % numPrizes;
 cout << "You spun the wheel and won: " << prizes[index]
<< endl;
 return index;
}
int main() {
 // Seed the random number generator with the current
time
 srand(static_cast<unsigned int>(time(0)));
 cout << "Welcome to Spin and Win Game!" << endl;
 cout << "Spin the wheel and see what you win." << endl;
 const int numPrizes = 6;
 string prizes[numPrizes] = {"Cash", "Free Spin", "Trip",
"Car", "Gadget", "No Prize"};
 char playAgain;
 do {
 int result = spinWheel(prizes, numPrizes);
 cout << "Do you want to spin again? (y/n): ";
 cin >> playAgain;
 } while (playAgain == 'y' || playAgain == 'Y');
 cout << "Thanks for playing Spin and Win Game!
Goodbye!" << endl;
 return 0;}
```

OUTPUT:

```
Welcome to Spin and Win Game!
Spin the wheel and see what you win.
You spun the wheel and won: Free Spin
Do you want to spin again? (y/n): y
You spun the wheel and won: Cash
Do you want to spin again? (y/n): n
Thanks for playing Spin and Win Game!
```

```
pin the wheel and see what you win.
ou spun the wheel and won: Trip
o you want to spin again? (y/n): y
ou spun the wheel and won: Cash
o you want to spin again? (y/n): y
ou spun the wheel and won: Cash
o you want to spin again? (y/n): y
ou spun the wheel and won: Car
o you want to spin again? (y/n): y
ou spun the wheel and won: Cash
o you want to spin again? (y/n): n
hanks for playing Spin and Win Game! Goodbye!
```


PRESENTED BY:
SOURABH (14218002722)
ANIL (36718002722)

MUSIC PLAYLIST PROJECT

What is our project?

It is a way to manage and manipulate a list of songs. A doubly linked list is a data structure that consists of nodes, and each node has a reference to the next and the previous node. This makes it easy to navigate both forwards and backwards through the playlist.

What we will be doing here?

In this code, we define a Node class to represent each song and a playlist class that manages the doubly linked list of songs. You can add songs, play the current song, skip to the next or previous song, and display the entire playlist. Elements we will be using:

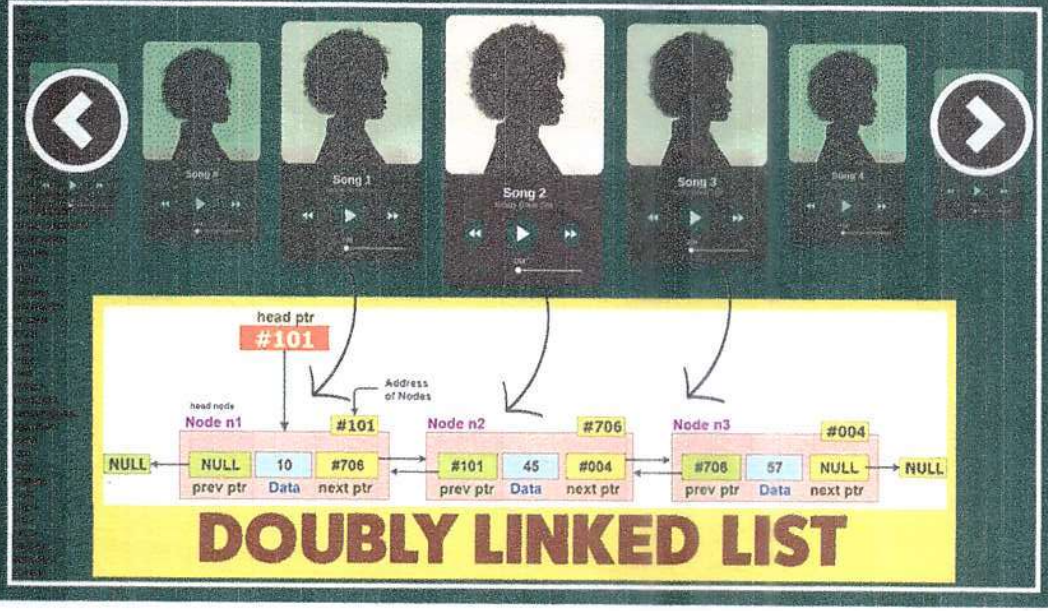
1)Node Class :-
Nodes represent songs. Include attributes for song data and references to the next and previous songs.

2)Playlist Class :
Manages the playlist using DLL. Maintains head, tail, and current song pointers. Implement methods for adding songs, playing the current song, and navigating between songs.

Future Scopes:

- Add features like song removal, shuffling, repeating, and metadata storage.
- Store playlists in files for future use.
- Doubly linked lists facilitate efficient song navigation and playlist management.

WORKING EXAMPLE



Delhi Technical Campus

Affiliated to GGSIP University, New Delhi • Approved by AICTE & Council of Architecture

PSEUDO CODE

```
struct Song
    title: string (up to 100 characters)
    artist: string (up to 100 characters)
    prev: reference to previous song
    next: reference to next song

struct Playlist
    head: reference to the first song
    tail: reference to the last song

function createSong(title: string, artist: string) -> Song
    newSong = allocate Song
    set newSong.title and artist
    return newSong

function addSong(playlist: Playlist, song: Song)
    if playlist.head is NULL
        set playlist.head and tail to song
    else
        set song.prev to playlist.tail
        set playlist.tail.next to song
        set playlist.tail to song

function display(playlist: Playlist)
    current = playlist.head
    while current is not NULL
        print "Title: " + current.title
        print "Artist: " + current.artist
        print newline
        set current to current.next
```


Presented to - Dr. Yashu Shanker

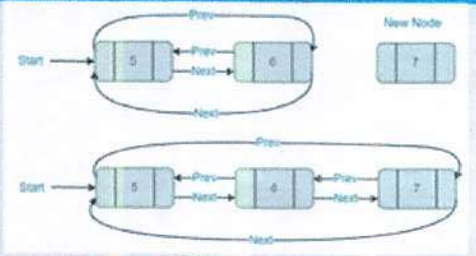
By -

- Keshava Yadav
- Jatin Tanwar
- Yash Kumar

Music Playlists

& DOUBLY-LINKED LISTS

A simple and efficient way to manage your music playlists.



1 WHAT IS A DOUBLY-LINKED LIST?

- Each node has a data field, a previous pointer, and a next pointer
- Allows for traversal in both directions
- More complex to implement than singly linked lists, but more flexible and efficient for certain operations

2 HOW DOES IT WORKS?

- Node has data, previous, and next pointers
- Allows traversal in both directions
- More complex than singly linked lists, but more flexible and efficient for certain operations

3 LINKED LIST & PLAYLIST

1. Start at the start node and add it to the queue.
2. Remove the first node from the queue and explore it.
3. If the node is the goal node, then the algorithm has found a solution and it terminates.
4. Otherwise, add all of the node's neighbors to the queue.
5. Repeat steps 2-4 until the queue is empty.

4 WHY LINKED LIST FOR PLAYLISTS?

- Adding a new song to the playlist: To add a new song to the beginning of the playlist, you can simply create a new node containing the song data and set its next pointer to the first node in the playlist. Then, you update the pointer of the previous node to point to the new node.
- Removing a song from the playlist: To remove a song from the playlist, you can simply update the pointers of the previous and next nodes to point around the removed node.
- Shuffle the playlist: To shuffle the playlist, you can randomly reorder the pointers of the nodes in the list. This can be done using a variety of different algorithms.
- Repeat the playlist: To implement repeat, you can create a circular doubly linked list, where the last node points to the first node. This allows you to play the playlist over and over again.

5 ALGORITHM

[Head] <-> [Node 1] <-> [Node 2] <-> [Node 3] <-> [Tail]

```
def traverse_doubly_linked_list(head):
    node = head
    while node is not None:
        # Do something with node.data
        node = node.next

def traverse_doubly_linked_list(tail):
    node = tail
```

DELHI TECHNICAL CAMPUS

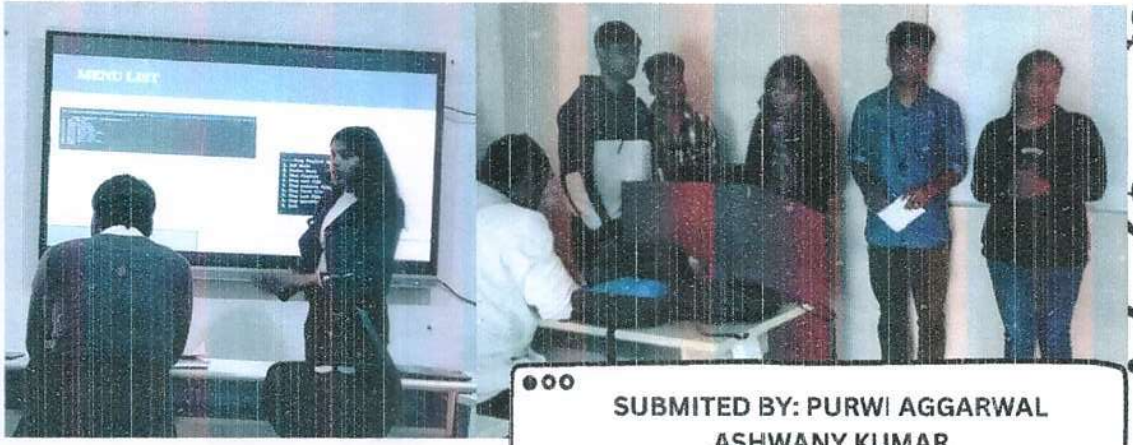
GREATER NOIDA

Affiliated to GGSIPU and approved by AICTE & COA

Affiliated to GGSIP University, New Delhi
Approved by AICTE & Council of Architecture

MUSICAL PLAYLIST

SUBMITED TO:YASHU SHANKER



SUBMITED BY: PURWI AGGARWAL
ASHWANY KUMAR
GAURAV KUMAR
RAKSHIT MALIK
VANSHIKA NAGPAL

DELHI TECHNICAL CAMPUS
GREATER NOIDA

Affiliated to GGSIPU and approved by AICTE & COA

Affiliated to GGSIP University, New Delhi
Approved by AICTE & Council of Architecture

Objective

To make sudoku solver using backtracking method

INPUT

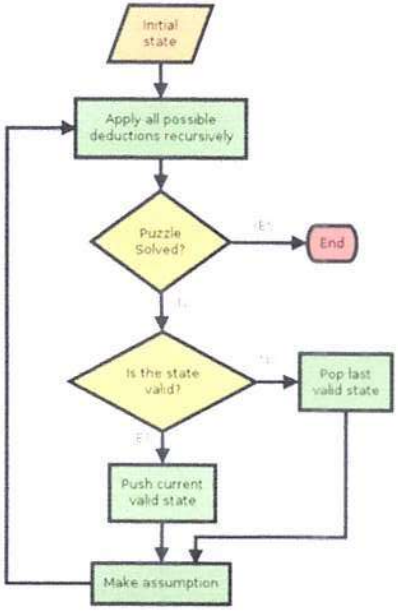
```
int main(){
int grid[N][N] =
 { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
 { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
 { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
 { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
 { 9, 0, 0, 8, 0, 3, 0, 0, 5 },
 { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
 { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
 { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
 { 0, 0, 5,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

OUTPUT

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

```
int isSafe(int grid[N][N], int row,
                        int col, int num)
{
int solveSudoku(int grid[N][N], int row, int col)
```

FLOWCHART



ALGORITHM

1. Initialize Grid: Start with an unsolved Sudoku puzzle.
2. Find Empty Cell: Look for an empty cell; if none, the puzzle is solved.
3. Try Values: Assign numbers (1 to 9) to the empty cell.
4. Check Validity: Verify if the assignment adheres to Sudoku rules.
5. Recursive Backtracking: If valid, move to the next empty cell recursively. If the grid is filled and valid, return true.
6. Backtrack on Failure: If no valid number for the cell, backtrack to the previous and try the next value.
7. Repeat Until Solution or Exhausted: Continue the process until the grid is filled or all possibilities are exhausted without a solution.

Faculty: Dr. Yashu Shanker

Presented by:
DANISH (20118002722)
RIYANSH(2041002722)

Department of Computer Science and Engineering
Delhi Technical Campus, Greater Noida

MINIMIZING CASH FLOW USING GRAPHS AND HEAPS

Introduction

Cash flow minimization is a crucial aspect of financial management, particularly in business settings. It involves optimizing the movement of funds to ensure a steady flow of cash, minimizing expenses, and maximizing profits. The primary goal of cash flow minimization is to reduce the overall cost of financing operations and enhance profitability.

Pseudocode

```
function greedyCashFlowMinimization(graph): NCFHeap = initializeNCFHeap(graph) while NCFHeap is not empty: maxCreditor = extractMax(NCFHeap) maxDebtor = extractMin(NCFHeap) transferAmount = minimum(maxCreditor.NCF, maxDebtor.NCF) updateNCF(maxCreditor, -transferAmount) updateNCF(maxDebtor, transferAmount) if maxCreditor.NCF == 0: removeVertexFromNCFHeap(NCFHeap, maxCreditor) if maxDebtor.NCF == 0: removeVertexFromNCFHeap(NCFHeap, maxDebtor)
```

Problem Definition

The cash flow minimization problem involves finding the optimal sequence of transactions between individuals or entities to minimize the overall cost of financing operations. It aims to efficiently transfer funds between creditors (those with excess funds) to debtors (those with insufficient funds) while minimizing the total amount of money transferred.

Cash Flow Minimization Algorithm

Initialize Heap → Identify Maximum Creditor and Maximum Debto → Calculate Transfer Amount → Update Net Cash Flow → Check for Elimination → Repeat

Key Concepts and Takeaways

Cash flow minimization is essential for reducing financing costs. Graphs and heaps effectively represent and solve cash flow problems, with greedy algorithms providing an efficient solution. Net cash flow guides optimal transaction sequencing.

Group Name:

Manish Kumar
sourabh Kumar
Ishu Kumar

Under the guidance of:
Dr. yashu shankar