Report Of DSA project: Using Linked Lists

Submitted By:

Md Faizan Ansari [24PG00019]

Harshvardhan Manavalan [24PG00017]

Muskan Kumari [24PG00076]

Abhilash G [24PG00068]

Under the guidance of

Usha Subramanian

Course: MCA & MSc Data Science

I Semester 2024 - 2025

School of Engineering & Natural Science

1. Background

Sparse matrices are those special data structures applied in scenarios where most of the elements in a matrix are zeros. The usual ways of storing matrices consume memory for all the elements, though they are zero. Sparse matrices optimize this by storing only non-zero elements; thus, there is a significant saving of memory as well as of computational overhead.

In practical application, sparse matrices are very useful in representing huge datasets where values are few and scattered. For example:

- **User Information in Recommender Systems:** User ratings for movies, products, or services in applications like streaming services or e-commerce are naturally sparse because each user interacts with a very small fraction of items.
- **Graph Representations:** Large graphs have very few edges. Therefore, sparse matrix representation is more economical in terms of storage and computation.

IPL Data Context

This is the data analysis project on the Indian Premier League, an elite-class cricket tournament. The dataset includes the following:

- **Half-Centuries Matrix:** It displays whether a player has scored a half-century in a particular match. Since there are 120 players and 74 matches, the matrix has 8,880 cells, but in most of the matches, hardly any player scores half-centuries.

- **Wickets Matrix:** This matrix keeps track of whether a player took a wicket in a match. Similar to the half-centuries matrix, it has 8,880 cells but sparse non-zero entries because of the nature of the game.

Sparse matrix representation for IPL data has several advantages:

- **Memory Efficiency:** Instead of storing 8,880 cells per matrix, only the non-zero elements are stored, which drastically reduces storage requirements.
- **Scalability:** When extended to 17 IPL seasons, the data size grows exponentially. Sparse matrix representation ensures manageable storage even for such large datasets.
- **Ease of Analysis:** Sparse representation facilitates efficient computation and quick extraction of meaningful patterns, such as identifying high-performing players.

2. Problem Definition

The problem deals with the efficient storage and processing of IPL player performance data using sparse matrices and linked lists, wherein non-zero values represent half-centuries and wickets. Its objective is to create linked list representations, calculate transposed matrices, identify all-rounders, and analyze memory-efficient data storage.

Key Concepts in the Project

This project involves **Data Structures and Algorithms (DSA)** concepts applied to sparse matrices and linked lists. Here are the key concepts:

---

a. Sparse Matrix

A **sparse matrix** is a matrix where most elements are **zero**. Instead of storing the full matrix, only the **non-zero elements** are stored to **save memory**.

Why Use Sparse Matrices?

- Reduces memory usage.
- Faster computation by avoiding unnecessary zero values.
- Useful in applications like **graph representations, recommendation systems, and scientific computing**.

Example of a Sparse Matrix:

```
0 0 0 0 0
0 4 0 0 5
0 0 0 0 0
0 0 5 0 9
```

Instead of storing the full matrix, we store:

Row	Column	Value
1	1	4
1	4	5
3	2	5
3	4	9

---

b. Linked List Representation of Sparse Matrices

Since the matrix has **many zeros**, we use a **linked list** instead of a 2D array.

Node Structure:

Each node in the linked list stores:

- **Row index (Player Number)**
- **Column index (Match Number)**
- **Value (1 for half-century or wicket)**
- **Pointer to the next node**

This linked list representation allows efficient **storage and retrieval**.

---

c. Transpose of a Sparse Matrix

Matrix Transposition swaps **rows and columns**. If a matrix is stored in **row-major order**, transposing it requires **storing it in column-major order**.

For example, transposing:

1 0 3
0 5 0
7 0 9

Results in:

1 0 7
0 5 0
3 0 9

In this project, we:

- Create **linked lists** for the **transposed matrices**.
- Use **row-major or column-major format**.

---

d. Identifying All-Rounders Using Linked Lists

A player is considered an **all-rounder** if:

- **Scored at least one half-century**.
- **Took at least one wicket**.

A new **linked list** is created, storing:

- **Player number**
- **Total half-centuries**
- **Total wickets taken**

This helps in analyzing **which players performed well in both batting and bowling**.

---

e. Memory Efficiency & Real-World Applications

- **Memory Optimization**: Storing only non-zero values saves a significant amount of memory.
- **Graph Representation**: Adjacency matrices in graphs are often sparse.
- **Machine Learning**: Storing user-item interactions in recommendation systems.

---

f. **Problem Definition: Sparse Matrices and Memory Efficiency**

- A **sparse matrix** has very few nonzero elements, making traditional storage inefficient. These matrices are useful in graph algorithms, scientific computing, and machine learning. Using **linked lists** for storage helps manage memory efficiently by allocating space only for nonzero values, reducing storage needs and improving computational performance.

Objective of the Project:

The objective of this project is to store and manipulate IPL player performance data efficiently by using linked lists to represent sparse matrices. Implementing linked list-based representations would optimize memory usage while allowing effective retrieval of key statistics, such as half-centuries and wickets taken. It also includes generating transposed matrices and finding all-rounders to analyze the performance of the players.

Steps to Solve the Problem:

1. **Read Input Data:**
   - Load the two sparse matrices from the provided dataset, where one matrix represents half-centuries and the other represents wickets taken by IPL players across multiple matches.
2. **Create Linked List Representations:**
   - Construct two linked lists to store only non-zero elements of each sparse matrix.
   - Each node in the linked list should contain the row index (player number), column index (match number), and the corresponding value (1 for half-century or wicket).
3. **Generate Transpose of Sparse Matrices:**
   - Construct two new linked lists representing the transposed versions of both sparse matrices.
   - In the transposed version, rows and columns are interchanged while maintaining non-zero values.

4. **Identify and Store All-Rounders:**
    - Create a separate linked list to store players who have scored at least one half-century and taken at least one wicket.
    - Each node in this list should contain the player number, total half-centuries, and total wickets taken.
5. **Display Linked List Representations:**
    - Print the original and transposed sparse matrices using the linked list representation.
    - Display the all-rounders' list with player-wise statistics.
6. **Analyze Efficiency of Linked List Representation:**
    - Discuss how linked lists effectively store sparse matrices by reducing memory usage.
    - Provide insights on the real-world applications of sparse matrices and their benefits in handling large datasets efficiently.

Final Goal:

To represent IPL player performance data using linked lists for sparse matrices efficiently, which would allow for optimized storage, retrieval, and analysis of half-century and wicket-taking statistics while identifying all-rounders and evaluating memory efficiency.

3. Explanation of Work

3.1 Half-Centuries Linked List

Implementation:

The file half_centuries.txt holds information about whether a player has scored a half-century in a given IPL match. This is read into a linked list where each node contains the following:

- **Player Number:** It is a unique identifier for the player.
- **Match Number:** It refers to the match.
- **Value:** It is a binary value 1 for a half-century, and 0 otherwise.

Detailed Steps:

- **File Parsing:** It reads the file to pick non-zero values representing half-centuries.
- **Construct nodes:** For each non-zero entry, construct a node consisting of player details and match details.
- **Construction of Linked List:** Insert each of these nodes either in row major or column major order in the linked list.
- **Display:** Display the sparse matrix in its original form using the linked list.

3.2 Wickets Linked List

Implementation:

In wickets.txt, information is provided as to whether a player has taken a wicket in a match. A linked list is created just like the half-centuries list where each node contains:

- **Player Number:** This is the player ID.
- **Match Number:** This is the match number.
- **Value:** A binary value 1 if a wicket was taken; otherwise, it is 0.

Step-by-Step Details:

- **File Reading:** Read the file that identifies non-zero values for wickets-node
- **Creation:** Create a node for each non-zero value.
- **Construction of Linked List**: Construct the linked list using the same ordering convention used in the half-centuries list.
- **Display:** Reconstruct and display the sparse matrix for wickets using the linked list representation.

3.3 All-Rounder Data Linked List

All-rounders are players who have scored at least one half-century and taken at least one wicket across matches.

Implementation:

Merge Linked Lists: Combine the half-centuries and wickets linked lists by comparing player numbers.

- **Filter Players:** Identify players appearing in both lists.
- **Node Creation:** Create nodes for each identified all-rounder containing:
- **Player Number**
- **Total Matches with Half-Centuries**
- **Total Matches with Wickets**
- **Consolidated List:** Build a new linked list specifically for all-rounder data.
- **Visualization:** Show a table summarizing each all-rounder's contribution.

3.4 Sparse Matrix Representation and Transpose

Original Matrix Representation: A linked list holds only the nonzero values of the sparse matrix. Each node in the linked list contains:

Row and column indices.

The nonzero value (e.g., 1 for half-centuries or wickets).

- **Matrix Transposition:** To analyze match-wise performance, the rows and columns of the sparse matrix are swapped. The transposition is done by:
- **Node Traversal**: Iterate through the original linked list.
- **Index Swapping:** Swap row and column indices for each node.
- **New Linked List**: Create a new linked list for the transposed matrix.
- **Visualization: Both** the original and transposed matrices are reconstructed using their respective linked list representations and displayed for analysis.

4.

What is a matrix with very few zero elements called? In what situations are such matrices useful? Can a linked list implementation help manage memory efficiently? Write a short note.

A matrix with very few zero elements is called a **sparse matrix**. Sparse matrices are particularly useful in situations where most elements of the matrix are zero, as storing only the non-zero elements significantly reduces memory usage and computational overhead.

Situations Where Sparse Matrices Are Useful:

1. **Graph Representation**: Sparse matrices can represent adjacency matrices for graphs with a large number of nodes but relatively few edges. For example, social networks where each user is connected to only a small subset of other users.
2. **Recommendation Systems**: Sparse matrices store user-product or user-movie ratings, as not every user interacts with every product or watches every movie.
3. **Scientific Computing**: Sparse matrices arise naturally in fields like finite element analysis and optimization problems where the systems involve many variables but few interactions between them.
4. **Image Compression**: Sparse matrices are used to store pixel values in images with large homogeneous regions, where most of the pixel values are zero.
5. **Machine Learning**: Feature matrices in machine learning often contain many zero entries, such as in natural language processing (NLP), where sparse matrices represent word embeddings or term-document matrices.

Linked List Implementation for Memory Efficiency:

The linked list representation is well-suited for sparse matrices because it stores only the non-zero elements. Each node in the linked list contains:

- **Row Index**: Position of the element in the matrix row.
- **Column Index**: Position of the element in the matrix column.
- **Value**: The non-zero value at the specified row and column.

By avoiding storage for zero values, the linked list implementation:

- **Reduces Memory Usage**: Memory is allocated only for the nodes representing non-zero elements, saving significant space when matrices are large.
- **Improves Traversal Efficiency**: Iterating over non-zero elements is faster compared to scanning through a full matrix with many zeros.

5.Tables and Figures

Tables

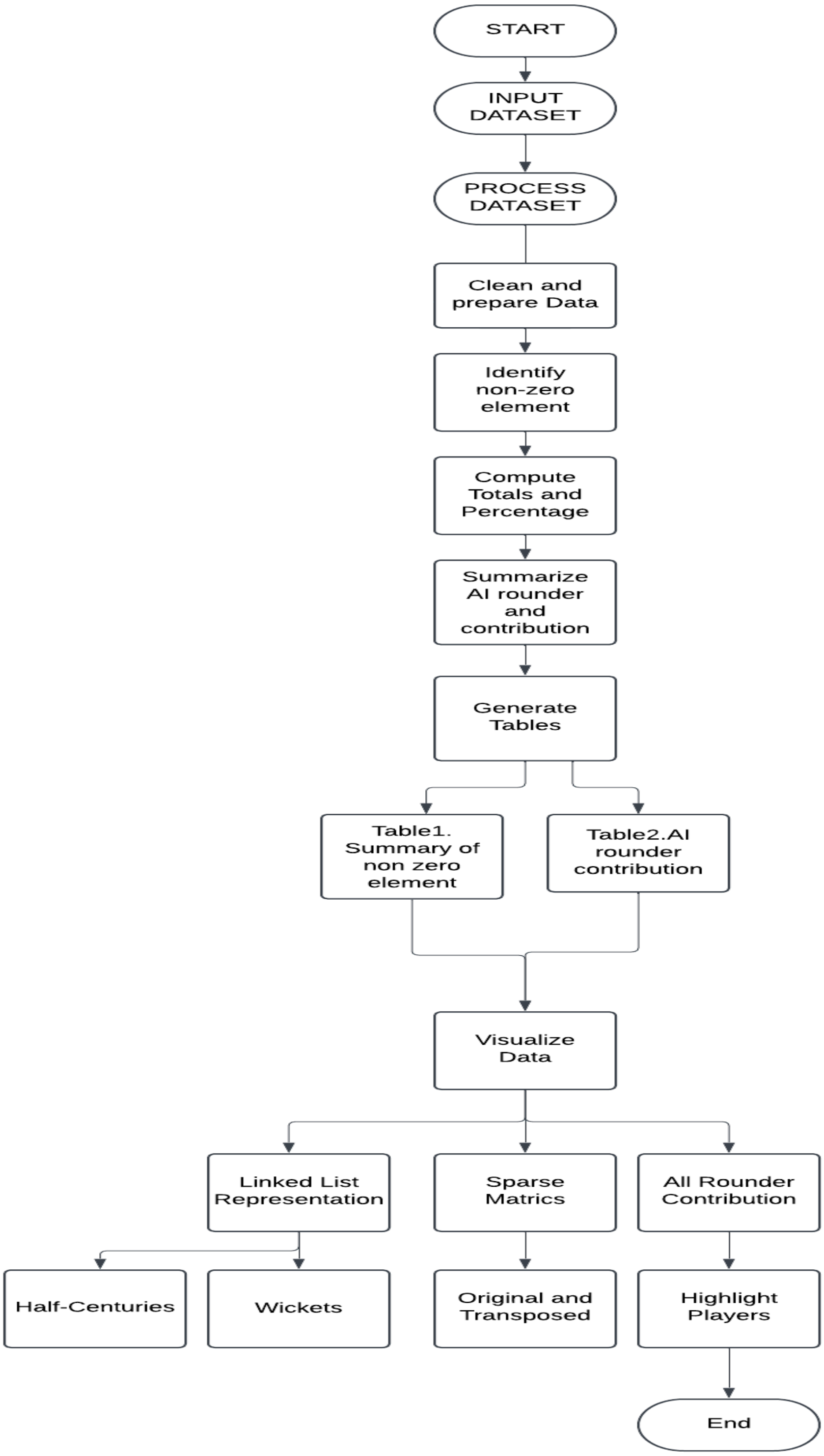
5.1 Summary of Non-Zero Elements

Data Type	Total Non-Zero Elements	Percentage Non-Zero
Half-Centuries	60	0.68%
Wickets	75	0.84%

5.2 All-Rounder Contributions

Player Number	Matches with Half-Centuries	Matches with Wickets
Player 1	5	0
Player 2	0	0
Player 3	4	0
….	….	….
Player 118	4	0
Player 120	0	6

6. Flowchart


START
INPUT DATASET
PROCESS DATASET
Clean and prepare Data
Identify non-zero element
Compute Totals and Percentage
Summarize Al rounder and contribution
Generate Tables
Table1. Summary of non zero element
Table2.Al rounder contribution
Visualize Data
Linked List Representation
Sparse Matrics
All Rounder Contribution
Half-Centuries
Wickets
Original and Transposed
Highlight Players
End

7. Runtime Performance

Overview

This report presents the time complexity analysis of various functionalities implemented in the program. The execution time of each operation has been recorded based on different test cases.

Test Cases and Execution Time

1. Display Half Century

Operations:

- File parsing: **0.002000 seconds**
- Displaying results: **0.006000 seconds**

Total execution time: 0.008000 seconds

2. Display Wickets

Operations:

- File reading and processing: **0.000000 seconds**
- Displaying linked list: **0.019000 seconds**

Total execution time: 0.019000 seconds

3. Display All-Rounder

Operations:

- Parsing data files: **0.002000 seconds**
- Displaying summary: **0.001000 seconds**

Total execution time: 0.004000 seconds

4. Display Transpose Matrix

Operations:

- File parsing and matrix creation: **0.000000 seconds**
- Transposing matrices: **0.000000 seconds**
- Displaying matrices: **2.254000 seconds**

Total execution time: 2.254000 seconds

Complexity Analysis

Functionality	Observed Time (seconds)	Expected Complexity
Display Half Century	0.008000	O(n)
Display Wickets	0.019000	O(n)
Display All-Rounder	0.004000	O(n)
Transpose Matrix	2.254000	O(n*m)

8.Conclusion

The project shows that linked lists may be used effectively to represent sparse matrices, especially with IPL data in which most elements of the matrix are zero. Instead of conventional methods of matrix storage, in which all the elements, whether zero or otherwise, are stored, linked lists focus only on the non-zero elements. The memory usage goes down drastically as the size of the dataset increases, say, over many IPL seasons.

By using this method, we are able to ensure that only the required data, such as half-centuries and wickets, is stored while maintaining a clear structure which allows for easy retrieval and manipulation of data. This approach is highly useful for:

The team can easily identify all-rounders who can bat as well as bowl, as the data regarding batting and bowling are merged; hence one can pick out players who significantly contribute to their team's success.

- **Performance Analysis:** Since the matrix can be transposed, it is easy to analyze match-wise trends and individual performances. It gives an idea of how the players perform in different matches or against specific opponents.

- **Scalability:** It is scalable enough to process the data from all 17 IPL seasons without using too much memory. This exemplifies ho
- sparsity can efficiently adapt to bigger data sizes in sparse matrix representation.

9. Annexures

Annexure-9.1 Input File 1

The following is an input file half_centuries.txt, which is used by the Program.

half_centuries.txt

```
Player	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21
	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36	M37	M38	M39	M40	M41	M42
	M43	M44	M45	M46	M47	M48	M49	M50	M51	M52	M53	M54	M55	M56	M57	M58	M59	M60	M61	M62	M63
	M64	M65	M66	M67	M68	M69	M70	M71	M72	M73	M74
1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	0	1	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	1	0	0	0
2	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
3	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	1	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
4	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
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
6	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
7	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
8	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
9	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
10	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	0	0	0	1	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
11	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
12	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
13	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
14	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
```

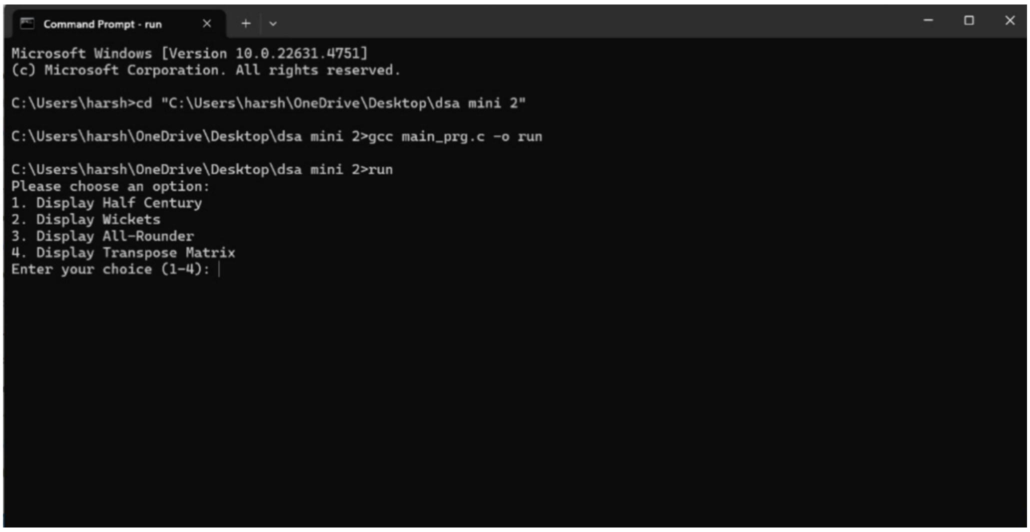
Annexure 9.2 Input File 2

The following is an input file two_plus_wickets.txt, which is used by the Program.

two_plus_wickets.txt

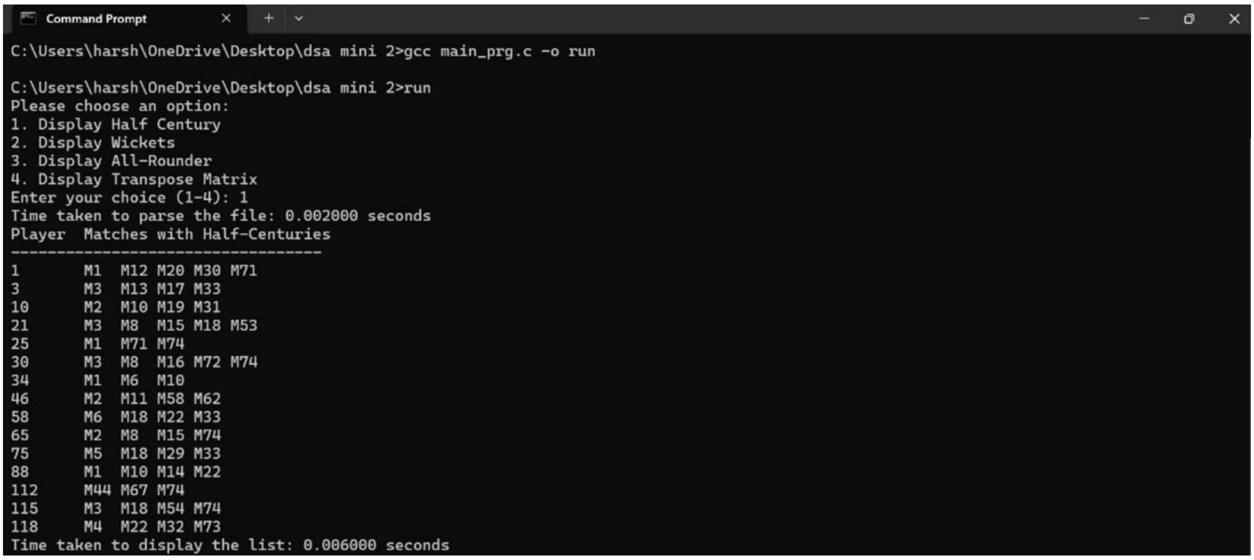
```
Player	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22
M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36	M37	M38	M39	M40	M41	M42	M43	M44	M45
M46	M47	M48	M49	M50	M51	M52	M53	M54	M55	M56	M57	M58	M59	M60	M61	M62	M63	M64	M65	M66	M67	M68
M69	M70	M71	M72	M73	M74
1	0	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
2	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
3	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
4	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
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
6	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
7	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
8	3	0	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	3	0
	0	0	0	0	3	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	3	0	3
9	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
10	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
11	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
12	0	0	3	0	0	0	0	0	0	0	3	0	2	0	0	0	0	0	0	0	0	0	4
	0	0	0	0	2	0	0	0	3	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
13	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
14	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
```

Annexure 9.3 output Of The code



The program is compiled using `gcc`, executed with `run`, and presents four options—Half Century, Wickets, All-Rounder, and Transpose Matrix. The user selects one option at a time to display the respective data, with processing done using linked lists and execution time measured using `clock()`.

Annexure 9.3.1



```
C:\Users\harsh\OneDrive\Desktop\dsa mini 2>gcc main_prg.c -o run

C:\Users\harsh\OneDrive\Desktop\dsa mini 2>run
Please choose an option:
1. Display Half Century
2. Display Wickets
3. Display All-Rounder
4. Display Transpose Matrix
Enter your choice (1-4): 1
Time taken to parse the file: 0.002000 seconds
Player  Matches with Half-Centuries
----------------------------------
1       M1  M12 M20 M30 M71
3       M3  M13 M17 M33
10      M2  M10 M19 M31
21      M3  M8  M15 M18 M53
25      M1  M71 M74
30      M3  M8  M16 M72 M74
34      M1  M6  M10
46      M2  M11 M58 M62
58      M6  M18 M22 M33
65      M2  M8  M15 M74
75      M5  M18 M29 M33
88      M1  M10 M14 M22
112     M44 M67 M74
115     M3  M18 M54 M74
118     M4  M22 M32 M73
Time taken to display the list: 0.006000 seconds
```

Pressing "1" selects the option to display players along with the matches in which they scored a half-century.

Annexure 9.3.2

```
C:\Users\harsh\OneDrive\Desktop\dsa mini 2>run
Please choose an option:
1. Display Half Century
2. Display Wickets
3. Display All-Rounder
4. Display Transpose Matrix
Enter your choice (1-4): 2
Time for reading and processing the file: 0.000000 seconds
Linked List of Non-Zero Values Data:
Player  Matches with Non-Zero Values
-----------------------------------
8       M1 (3) M6 (3) M11(3) M22(3) M28(3) M72(3) M74(3)
12      M3 (3) M11(3) M13(2) M23(4) M28(2) M32(3)
21      M3 (5) M15(4) M18(4) M23(3) M71(2) M74(2)
24      M19(2) M26(2) M33(4) M41(2) M46(2) M57(1) M72(3) M74(3)
46      M2 (3) M11(2) M17(3) M22(2) M25(2) M28(2) M32(2) M74(1)
60      M3 (3) M13(1) M21(4) M29(5) M72(2) M73(3)
75      M2 (2) M6 (2) M13(2) M18(2) M23(2) M31(3) M39(1) M71(2) M74(2)
78      M4 (2) M17(2) M25(2) M29(2) M33(2) M44(2) M49(2) M54(2) M73(2) M74(1)
115     M2 (3) M9 (2) M16(2) M22(1) M28(1) M31(1) M65(2) M72(2) M74(3)
120     M2 (4) M13(4) M35(3) M42(3) M68(3) M74(3)
Time for displaying the linked list: 0.019000 seconds
```

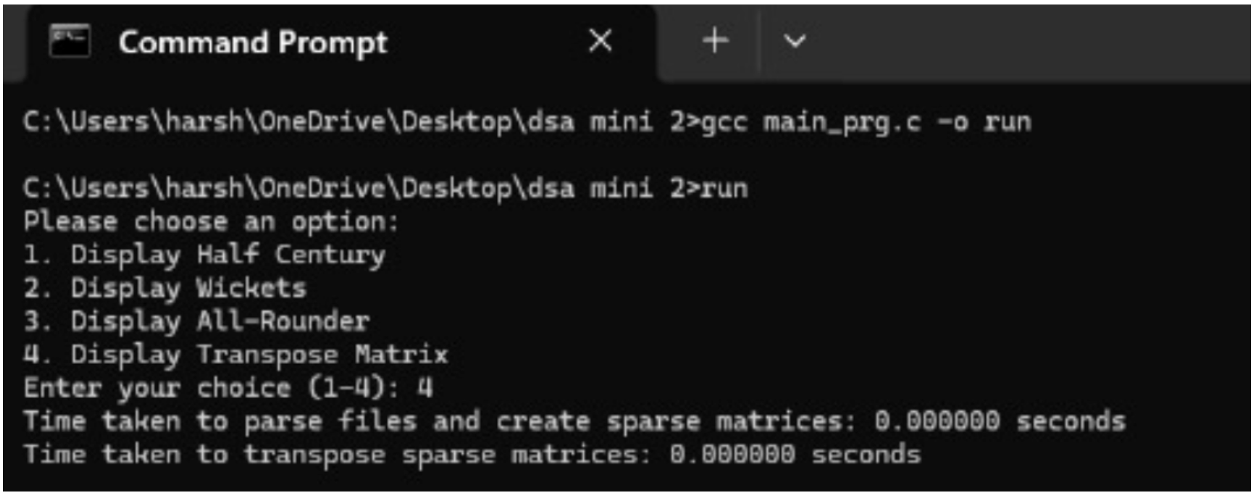
Pressing "2" selects the option to display players along with the matches in which they took wickets.

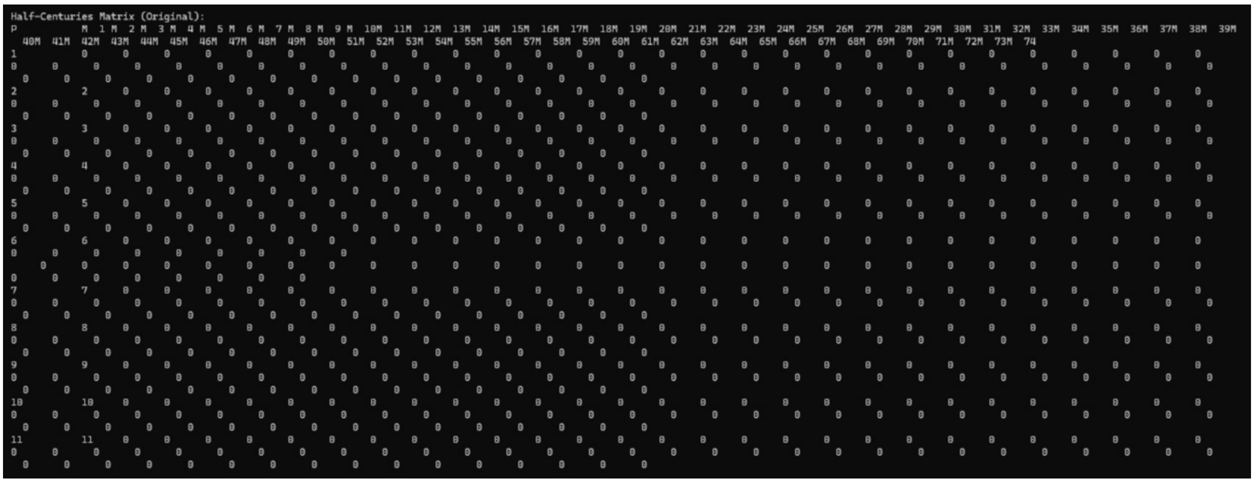
Annexure 9.3.3

```
C:\Users\harsh\OneDrive\Desktop\dsa mini 2>run
Please choose an option:
1. Display Half Century
2. Display Wickets
3. Display All-Rounder
4. Display Transpose Matrix
Enter your choice (1-4): 3
Time for parsing data files: 0.002000 seconds

All-Rounder Summary (Player, Total Half-Centuries, Total Wickets)
--------------------------------------------------------------
Player: 21       Half-Centuries: 5   Wickets: 20
Player: 46       Half-Centuries: 4   Wickets: 17
Player: 75       Half-Centuries: 4   Wickets: 18
Player: 115      Half-Centuries: 4   Wickets: 17

Time for displaying summary: 0.001000 seconds
Total runtime of the program: 0.004000 seconds
```

Pressing "3" selects the option to display players along with the matches in which they performed as all-rounders.

Annexure 9.3.4





Pressing "4" selects the option to display the transpose of the matrix.

10. References

- **YouTube:**

  https://youtu.be/yzs7GOS3nKY?si=jQTjdmDmZEG3T7Ku
- https://www.geeksforgeeks.org/basics-file-handling-c/
- https://chatgpt.com/
- https://gemini.google.com/app
- Google
- ChatGPT by OpenAI (2025) provides assistance with report creation and coding queries.