Banker's Algorithm Simulator

1. Purpose

The Banker's Algorithm Simulator is a web-based educational tool designed to help users visualize and understand the working of the Banker's Algorithm in operating systems. This simulator demonstrates how resource allocation and deadlock avoidance are managed, providing interactive steps for better comprehension.

2. Concept Overview

The Banker's Algorithm is a crucial deadlock-avoidance algorithm in operating systems. It ensures that the system remains in a safe state when allocating resources to multiple processes. The algorithm works by simulating resource allocation and checking if all processes can complete without causing a deadlock.

Key operations:

- **Input Tables:** Users specify the total resources, currently allocated resources, and the maximum resources needed for each process.
- **Need Calculation:** The difference between maximum and allocated resources.
- **Available Resources:** Total resources minus allocated resources.
- **Safe Sequence:** Determines a sequence of process executions ensuring no deadlock occurs.

3. Design and Implementation

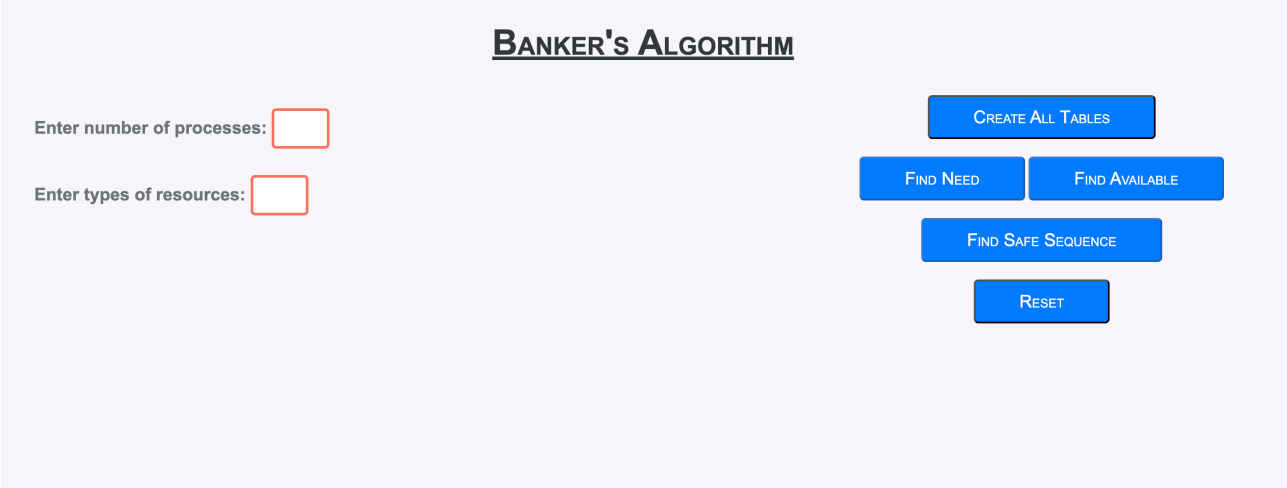
3.1 Project Structure

- **`gg.html`:** Defines the layout and structure of the simulator interface.
- **`gg.js`:** Contains all logic for table generation, need calculation, safety checks, and safe sequence determination.
- **`ggstyle.css`:** Provides the styling for an intuitive and visually appealing interface.

3.2 Simulator Interface

The interface consists of:

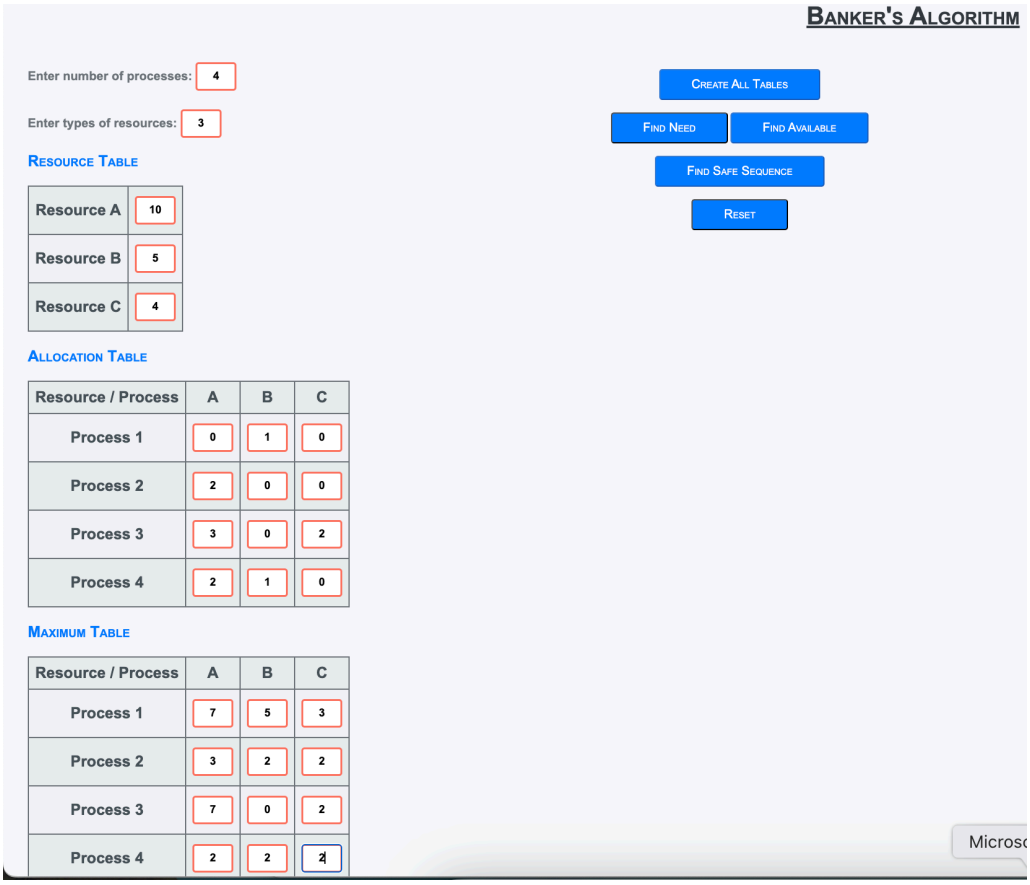
- **Input Fields:** Users enter the number of processes and resource types.
- **Action Buttons:** Generate tables, calculate needs and availability, and find safe sequences.
- **Dynamic Tables:** Users populate tables with allocation and maximum resource values.



3.3 Core Functions

- **`createTables()`:** Generates input tables for Allocation, Maximum, and Resource data.
- **`findNeed()`:** Calculates and displays the Need Table.
- **`findAvailable()`:** Computes available resources and displays the Available Table.
- **`generateSafeSeq()`:** Runs the safety algorithm to determine and display the safe sequence.

EXAMPLE :



Need Table

Resource / Process	A	B	C
Process 1	7	4	3
Process 2	1	2	2
Process 3	4	0	0
Process 4	0	1	2

Need (Process 1) = Max (7,5,3) - Allocation (0,1,0) = (7,4,3)

Need (Process 2) = Max (3,2,2) - Allocation (2,0,0) = (1,2,2)

Need (Process 3) = Max (7,0,2) - Allocation (3,0,2) = (4,0,0)

Need (Process 4) = Max (2,2,2) - Allocation (2,1,0) = (0,1,2)

Fin

Available Table

Resource A	3
Resource B	3
Resource C	2

Available (Resource A) = Total (10) - Total allocated (0 + 2 + 3 + 2) = 3

Available (Resource B) = Total (5) - Total allocated (1 + 0 + 0 + 1) = 3

Available (Resource C) = Total (4) - Total allocated (0 + 0 + 2 + 0) = 2

Safe Sequence

2	3	4	1

Process 2 : Need (1,2,2) <= Available (3,3,2) -> New Available (5,3,2)

Process 3 : Need (4,0,0) <= Available (5,3,2) -> New Available (8,3,4)

Process 4 : Need (0,1,2) <= Available (8,3,4) -> New Available (10,4,4)

Process 1 : Need (7,4,3) <= Available (10,4,4) -> New Available (10,5,4)

4. Usage Instructions

1. Open `gg.html` in a web browser.
2. Enter the number of processes and resource types.
3. Click "Create All Tables" to generate the input tables.
4. Fill in Allocation and Maximum tables.
5. Click "Find Need" to calculate the Need Table.
6. Click "Find Available" to compute available resources.
7. Click "Find Safe Sequence" to check if a safe sequence exists.

If a safe sequence is found, it will be displayed. Otherwise, the system is declared unsafe.

Link for example question : https://www.naukri.com/code360/library/banker-s-algorithm

5. Challenges Faced

- **Input Validation:** Ensuring all fields are filled and valid before calculations.
- **Edge Cases:** Handling scenarios with zero allocations or maximum values equal to allocations.

6. Future Scope

1. **Enhanced Visualization:** Add graphical representations of resource allocations and state transitions.
2. **Mobile Responsiveness:** Improve the UI for better usability on mobile devices.
3. **Detailed Error Handling:** Provide clearer error messages and user guidance.
4. **Process Prioritization:** Allow users to assign priorities for more realistic simulations.

7. Conclusion

This project successfully simulates the Banker's Algorithm, offering an interactive way to explore resource allocation and deadlock avoidance in operating systems. It is a valuable educational tool for students, educators, and professionals seeking a deeper understanding of this essential algorithm.