1. **Initialization**: The algorithm starts by initializing some data structures. These include:
   * **Available**: Represents the number of available resources of each type.
   * **Max**: Represents the maximum demand of each process.
   * **Allocation**: Represents the resources currently allocated to processes.
   * **Need**: Represents the remaining resources that a process needs to complete its task.
2. **Need Calculation**: The Need matrix is calculated as **Need[i][j] = Max[i][j] - Allocation[i][j]**. This matrix shows the remaining resources needed by each process to complete its task.
3. **Safety Check**:
   * The algorithm simulates the allocation of resources to processes by considering the **Available** and **Need** matrices.
   * It checks whether it's possible to allocate resources to all processes such that no deadlock occurs. If such a sequence exists, the system is in a safe state; otherwise, it's in an unsafe state.
   * The **is\_safe** function uses a loop to iterate through processes, checks if the resources needed by a process can be satisfied by the **Available** resources. If yes, it allocates resources to that process and releases its resources, making them available for other processes.
4. **Safety or Deadlock Detection**: If the algorithm can find a safe sequence, the system is in a safe state; otherwise, it's not, indicating a deadlock.

In essence, the Banker's Algorithm ensures that the system can allocate resources in such a way that processes can finish their execution without entering a deadlock state, ensuring both resource utilization and system safety.

**Data Structures:**

1. **Available**: A vector storing the number of available resources of each type.
2. **Max**: A 2D vector representing the maximum resources that each process may request.
3. **Allocation**: A 2D vector indicating the resources currently allocated to each process.
4. **Need**: A 2D vector representing the resources that each process still needs to complete its task.

**Functions:**

1. **calculate\_need(int n, int m)**

This function calculates the **Need** matrix for each process by subtracting the **Allocation** matrix from the **Max** matrix. The **Need** matrix represents the additional resources that each process needs to complete its task.

2. **is\_safe(int n, int m)**

This function checks whether the system is in a safe state or not by simulating the resource allocation and verifying if all processes can complete their tasks without causing a deadlock.

* **Work**: A vector initialized with the available resources.
* **Finish**: A vector to keep track of processes that have finished.

The algorithm runs a loop until all processes are finished or the system is deemed unsafe. It iterates over each process and checks if it can be allocated the resources it needs. If a process can be allocated resources, those resources are released (added back to the available pool), and the process is marked as finished. If no process can be allocated resources in a loop iteration, the system is considered unsafe.

**Main Function:**

1. **Input**: The user inputs the number of processes (**n**) and resource types (**m**), available resources, maximum resource requirements, and allocated resources for each process.
2. **Validation**: The inputs are validated to ensure that the maximum resource requests do not exceed available resources and that allocated resources do not exceed the maximum requirements.
3. **Calculation**: The **Need** matrix is calculated using the **calculate\_need()** function.
4. **Safety Check**: The system's safety is checked using the **is\_safe()** function.

**Error Handling:**

* If the user inputs violate the constraints (Max > Available or Allocation > Max), an error message is displayed, and the program exits with an error code (**1**).

**Output:**

* The program outputs whether the system is in a safe state or not based on the Banker's algorithm.

**Conclusion:**

The provided code is a straightforward implementation of the Banker's algorithm in C++. It takes user inputs for available resources, maximum resource requirements, and allocated resources, validates the inputs, calculates the **Need** matrix, and checks if the system is in a safe state.