Q-Write a program to implement Banker's algorithm..

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// Function to check if the current state is safe or not
bool isSafe(vector<vector<int>>& max, vector<vector<int>>& alloc, vector<int>& avail, vector<int>& work, vector<bool>& finish) {
    int n = max.size(); // Number of processes
     // Temporary vectors to store the work and finish status
    vector<int> tempWork = work;
    vector<bool> tempFinish = finish;
 // Check if all processes have been marked finished
    bool allFinished = all_of(tempFinish.begin(), tempFinish.end(), [](bool val) { return val; });
    if (allFinished) {
        return true; // Safe state
    }
     // Loop through all processes
    for (int i = 0; i < n; ++i) {
        if (!tempFinish[i]) {
            // Check if the current process can be satisfied with available resources
            bool canExecute = true;
            for (int j = 0; j < avail.size(); ++j) {
                if (max[i][j] - alloc[i][j] > tempWork[j]) {
                    canExecute = false;
                    break;
                }
            }

            // If the process can be satisfied, execute it and update available resources
            if (canExecute) {
                tempFinish[i] = true;
                for (int j = 0; j < avail.size(); ++j) {
                    tempWork[j] += alloc[i][j];
                }
```

```
            return isSafe(max, alloc, avail, tempWork, tempFinish); // Recursively check for the rest of the processes
        }
      }
    }

    return false; // Unsafe state
}
// Function to execute Banker's algorithm
void bankersAlgorithm(vector<vector<int>>& max, vector<vector<int>>& alloc, vector<int>& avail) {
    int n = max.size(); // Number of processes
    int m = avail.size(); // Number of resource types
   // Calculate need matrix
    vector<vector<int>> need(n, vector<int>(m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Initialize work and finish vectors
    vector<int> work = avail;
    vector<bool> finish(n, false);
 // Check if the initial state is safe
    if (!isSafe(max, alloc, avail, work, finish)) {
        cout << "Unsafe state detected. Exiting..." << endl;
        return;
    }
  // Vector to store the sequence of execution
    vector<int> safeSequence;

    // Implement Banker's algorithm to find safe sequence
    int count = 0;
    while (count < n) {
        bool found = false;
        for (int i = 0; i < n; ++i) {
```

```
            if (!finish[i]) {
                bool canExecute = true;
                for (int j = 0; j < m; ++j) {
                    if (need[i][j] > work[j]) {
                        canExecute = false;
                        break;
                    }
                }
                     if (canExecute) {
                    for (int j = 0; j < m; ++j) {
                        work[j] += alloc[i][j];
                    }
                          safeSequence.push_back(i);
                    finish[i] = true;
                    count++;
                    found = true;
                }
            }
        }
      // If no process can be executed, the system is in an unsafe state
        if (!found) {
            cout << "Unsafe state detected. Exiting..." << endl;
            return;
        }
    }
   // Print the safe sequence
    cout << "Safe sequence: ";
    for (int i = 0; i < safeSequence.size(); ++i) {
        cout << safeSequence[i] << " ";
    }
    cout << endl;
}

int main() {
    int n, m; // Number of processes and resource types
```

```
    cout << "Enter number of processes: ";
    cin >> n;
    cout << "Enter number of resource types: ";
    cin >> m;

    // Maximum demand matrix
    vector<vector<int>> max(n, vector<int>(m));
    cout << "Enter maximum demand matrix:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Process " << i << ": ";
        for (int j = 0; j < m; ++j) {
            cin >> max[i][j];
        }
    }

    // Allocation matrix
    vector<vector<int>> alloc(n, vector<int>(m));
    cout << "Enter allocation matrix:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Process " << i << ": ";
        for (int j = 0; j < m; ++j) {
            cin >> alloc[i][j];
        }
    }

    // Available resources vector
    vector<int> avail(m);
    cout << "Enter available resources vector: ";
    for (int i = 0; i < m; ++i) {
        cin >> avail[i];
    }

    // Execute Banker's algorithm
    bankersAlgorithm(max, alloc, avail);
return 0;}
```