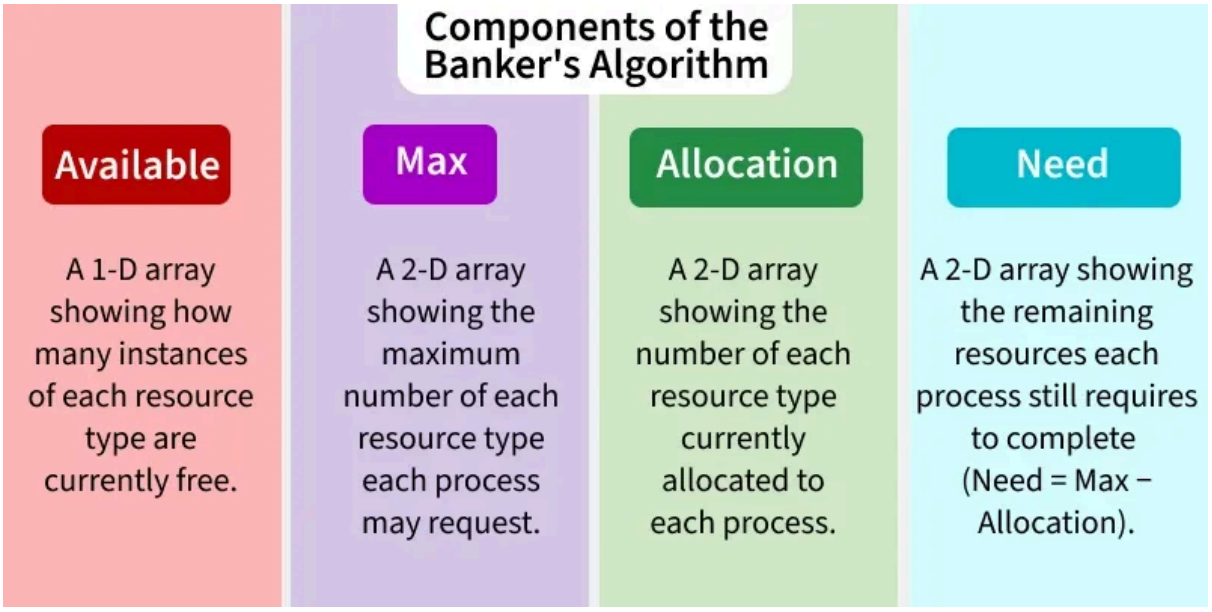
5.Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

Theory:

Banker's Algorithm is a **resource allocation and deadlock avoidance algorithm** used in operating systems. It ensures that a system remains in a safe state by carefully allocating resources to processes while **avoiding unsafe states that could lead to deadlocks**.

- The Banker's Algorithm is a smart way for computer systems to manage how programs use resources, like memory or CPU time.
- It helps prevent situations where programs get stuck and can not finish their tasks. This condition is known as deadlock.
- By keeping track of what resources each program needs and what's available, the banker algorithm makes sure that programs only get what they need in a safe order.



PROGRAM

```
#include <stdio.h>
#include <string.h>

int main() {
    int alloc[10][10], max[10][10];
    int avail[10], work[10], total[10];
    int i, j, k, n, need[10][10];
    int m;
    int count = 0;
    char finish[10];
```

```
    printf("Enter the number of processes and resources: ");
    scanf("%d%d", &n, &m);

    for (i = 0; i < n; i++) {
        finish[i] = 'n';
    }

    printf("Enter the maximum matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the allocation matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter the available vector: ");
    for (i = 0; i < m; i++) {
        scanf("%d", &total[i]);
    }

    // Initializing avail to total
    for (i = 0; i < m; i++) {
        avail[i] = total[i];
    }

    // Subtracting alloc from avail
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            avail[j] -= alloc[i][j];
        }
    }

    for (i = 0; i < m; i++) {
        work[i] = avail[i];
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

A:
    for (i = 0; i < n; i++) {
```

```
        int c = 0;
        for (j = 0; j < m; j++) {
            if (need[i][j] <= work[j] && finish[i] == 'n') {
                c++;
            }
        }

        if (c == m) {
            printf("All the resources can be allocated to Process %d\n", i + 1);
            printf("Available resources are: ");
            for (k = 0; k < m; k++) {
                work[k] += alloc[i][k];
                printf("%4d", work[k]);
            }
            printf("\n");
            finish[i] = 'y';
            printf("Process %d executed?: %c \n", i + 1, finish[i]);
            count++;
        }
    }

    if (count != n) {
        goto A;
    } else {
        printf("\nSystem is in a safe state\n");
    }

    printf("\nThe given state is a safe state\n");
    return 0;
}
```

OUTPUT:

```
Enter the no. of processes and resources: 4 3
Enter the maximum matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2 Available resources are: 6 2 3
Process 2 executed?:y
All the resources can be allocated to Process 3 Available resources are: 8 3 4
Process 3 executed?:y
All the resources can be allocated to Process 4 Available resources are: 8 3 6
```

Process 4 executed?:y
All the resources can be allocated to Process 1 Available resources are: 9 3 6
Process 1 executed?:y
System is in safe mode
The given state is safe state

VIVA Questions:

1.What is the safe state in the context of Banker's Algorithm?

2.How does Banker's Algorithm avoid Dead Lock?