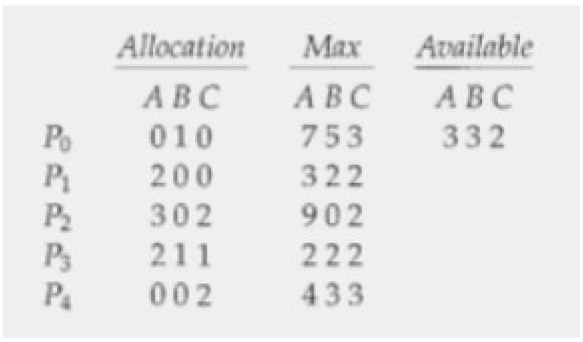
LAB NO: 8
DEADLOCK MANAGEMENT ALGORITHMS

1. Consider the following snapshot of the system. Write C/C++ program to implement Banker's algorithm for deadlock avoidance. The program has to accept all inputs from the user. Assume the total number of instances of A,B and C are 10,5 and 7 respectively.

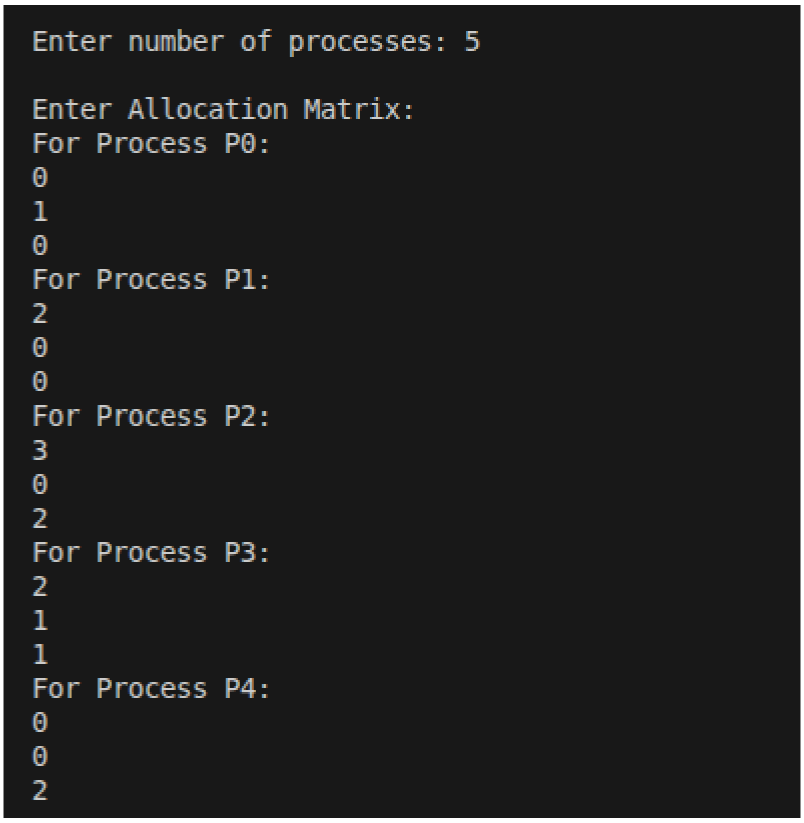


	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

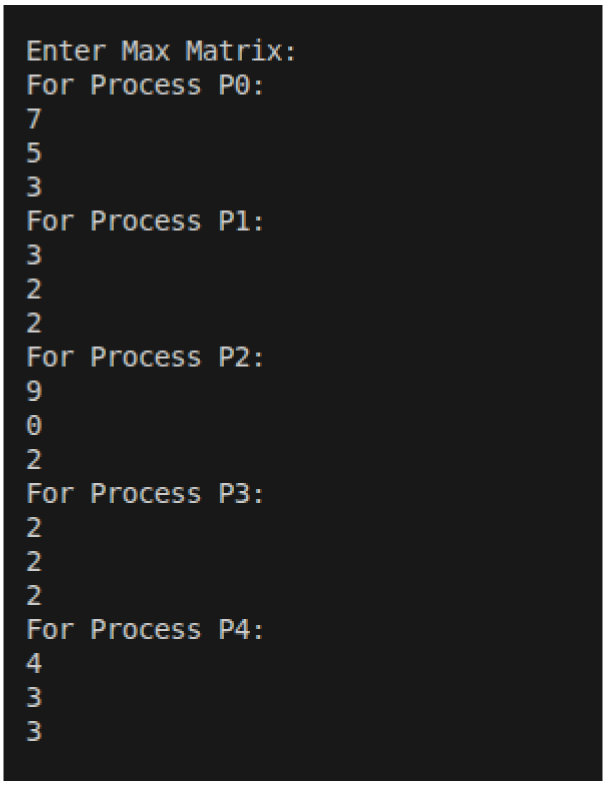
Program:

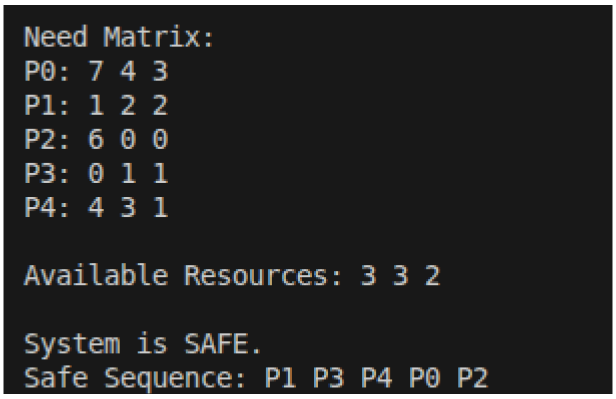
```
#include <stdio.h>
#include <stdbool.h>

int main() {
    int n;
    int total[3] = {10, 5, 7};

    printf("Enter number of processes: ");
    scanf("%d", &n);

    int alloc[n][3], max[n][3], need[n][3];
    int available[3];

    printf("\nEnter Allocation Matrix:\n");
    for(int i = 0; i < n; i++) {
        printf("For Process P%d:\n", i);
        for(int j = 0; j < 3; j++)
            scanf("%d", &alloc[i][j]);
    }

    printf("\nEnter Max Matrix:\n");
    for(int i = 0; i < n; i++) {
        printf("For Process P%d:\n", i);
        for(int j = 0; j < 3; j++)
            scanf("%d", &max[i][j]);
    }

    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    for(int j = 0; j < 3; j++) {
        int sum = 0;
    for(int i = 0; i < n; i++)
        sum += alloc[i][j];
    available[j] = total[j] - sum;
}

printf("\nNeed Matrix:\n");
for(int i = 0; i < n; i++) {
    printf("P%d: ", i);
    for(int j = 0; j < 3; j++)
        printf("%d ", need[i][j]);
    printf("\n");
}

printf("\nAvailable Resources: ");
for(int j = 0; j < 3; j++)
    printf("%d ", available[j]);
printf("\n");

bool finish[n];
int safeSeq[n];
int work[3];

for(int i = 0; i < n; i++)
    finish[i] = false;

for(int j = 0; j < 3; j++)
    work[j] = available[j];

int count = 0;

while(count < n) {
    bool found = false;

    for(int i = 0; i < n; i++) {
        if(!finish[i]) {
            bool safe = true;

            for(int j = 0; j < 3; j++) {
                if(need[i][j] > work[j]) {
                    safe = false;
                    break;
                }
            }

            if(safe) {
                for(int j = 0; j < 3; j++)
                    work[j] += alloc[i][j];

                safeSeq[count++] = i;
                finish[i] = true;
                found = true;
            }
            }
        }

        if(!found) {
            printf("\nSystem is NOT SAFE\n");
            return 0;
        }
    }

    printf("\nSystem is SAFE.\nSafe Sequence: ");
    for(int i = 0; i < n; i++)
        printf("P%d ", safeSeq[i]);

    printf("\n");

    return 0;
}
```

Output:

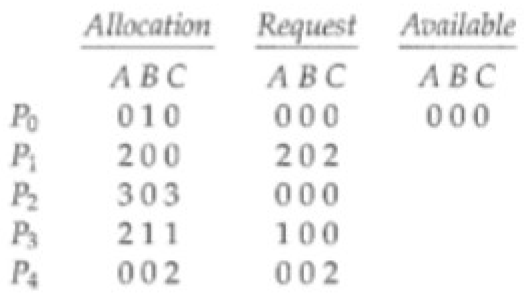


```
Enter number of processes: 5

Enter Allocation Matrix:
For Process P0:
0
1
0
For Process P1:
2
0
0
For Process P2:
3
0
2
For Process P3:
2
1
1
For Process P4:
0
0
2
```


```
Enter Max Matrix:
For Process P0:
7
5
3
For Process P1:
3
2
2
For Process P2:
9
0
2
For Process P3:
2
2
2
For Process P4:
4
3
3
```


```
Need Matrix:
P0: 7 4 3
P1: 1 2 2
P2: 6 0 0
P3: 0 1 1
P4: 4 3 1

Available Resources: 3 3 2

System is SAFE.
Safe Sequence: P1 P3 P4 P0 P2
```

2. Consider the following snapshot of the system. Write C/C++ program to implement deadlock detection algorithm.



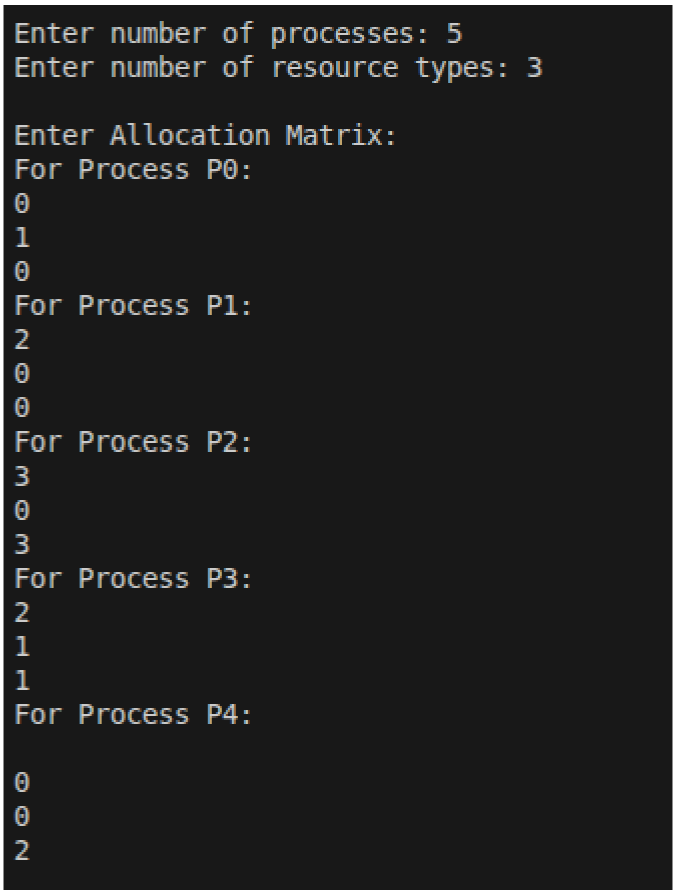
	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Program:

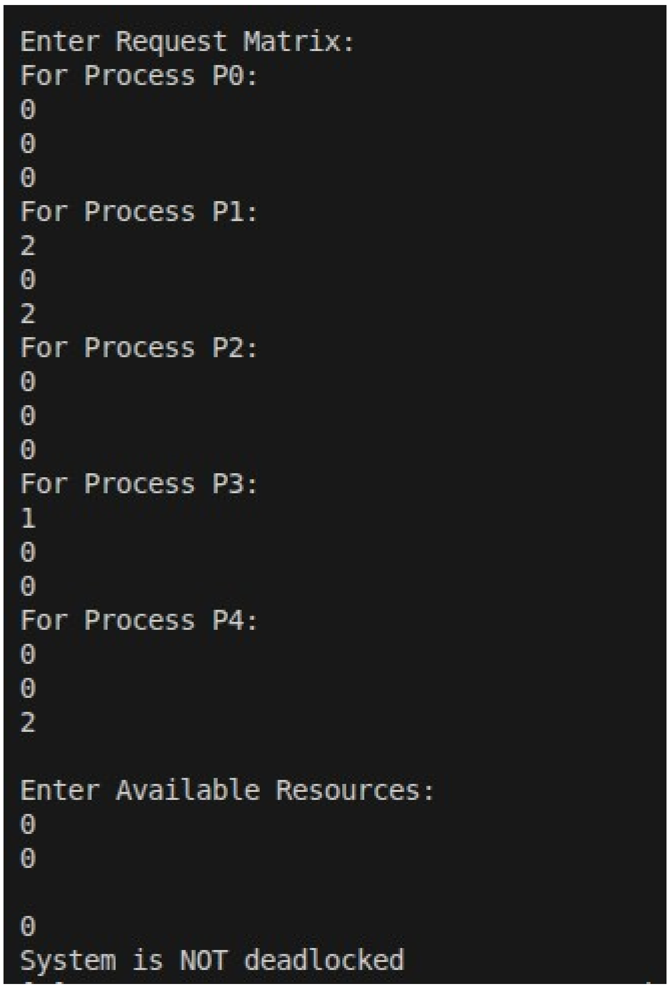
```
#include <stdio.h>
#include <stdbool.h>

int main() {
    int n, m;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter number of resource types: ");
    scanf("%d", &m);

    int allocation[n][m];
    int request[n][m];
    int available[m];

    printf("\nEnter Allocation Matrix:\n");
    for(int i = 0; i < n; i++) {
        printf("For Process P%d:\n", i);
        for(int j = 0; j < m; j++)
            scanf("%d", &allocation[i][j]);
    }

    printf("\nEnter Request Matrix:\n");
    for(int i = 0; i < n; i++) {
        printf("For Process P%d:\n", i);
        for(int j = 0; j < m; j++)
            scanf("%d", &request[i][j]);
    }

    printf("\nEnter Available Resources:\n");
    for(int j = 0; j < m; j++)
        scanf("%d", &available[j]);

    bool finish[n];
    int work[m];
```

```
for(int i = 0; i < n; i++) {
    bool zeroAlloc = true;
    for(int j = 0; j < m; j++) {
        if(allocation[i][j] != 0) {
            zeroAlloc = false;
            break;
        }
    }
    finish[i] = zeroAlloc;
}

for(int j = 0; j < m; j++)
    work[j] = available[j];

bool found;
do {
    found = false;

    for(int i = 0; i < n; i++) {
        if(!finish[i]) {
            bool possible = true;

            for(int j = 0; j < m; j++) {
                if(request[i][j] > work[j]) {
                    possible = false;
                    break;
                }
            }

            if(possible) {
                for(int j = 0; j < m; j++)
                    work[j] += allocation[i][j];

                finish[i] = true;
                found = true;
            }
        }
    }

} while(found);

bool deadlock = false;

for(int i = 0; i < n; i++) {
    if(!finish[i]) {
        printf("Process P%d is deadlocked\n", i);
        deadlock = true;
    }
}

if(!deadlock)
    printf("System is NOT deadlocked\n");
    else
        printf("System is in DEADLOCK state\n");

    return 0;
}
```

Output:



```
Enter number of processes: 5
Enter number of resource types: 3

Enter Allocation Matrix:
For Process P0:
0
1
0
For Process P1:
2
0
0
For Process P2:
3
0
3
For Process P3:
2
1
1
For Process P4:

0
0
2
```



```
Enter Request Matrix:
For Process P0:
0
0
0
For Process P1:
2
0
2
For Process P2:
0
0
0
For Process P3:
1
0
0
For Process P4:
0
0
2

Enter Available Resources:
0
0

0
System is NOT deadlocked
```