

**NAME : ADARSH GOUR(B42)**

**REGISTRATION ID:11806478**

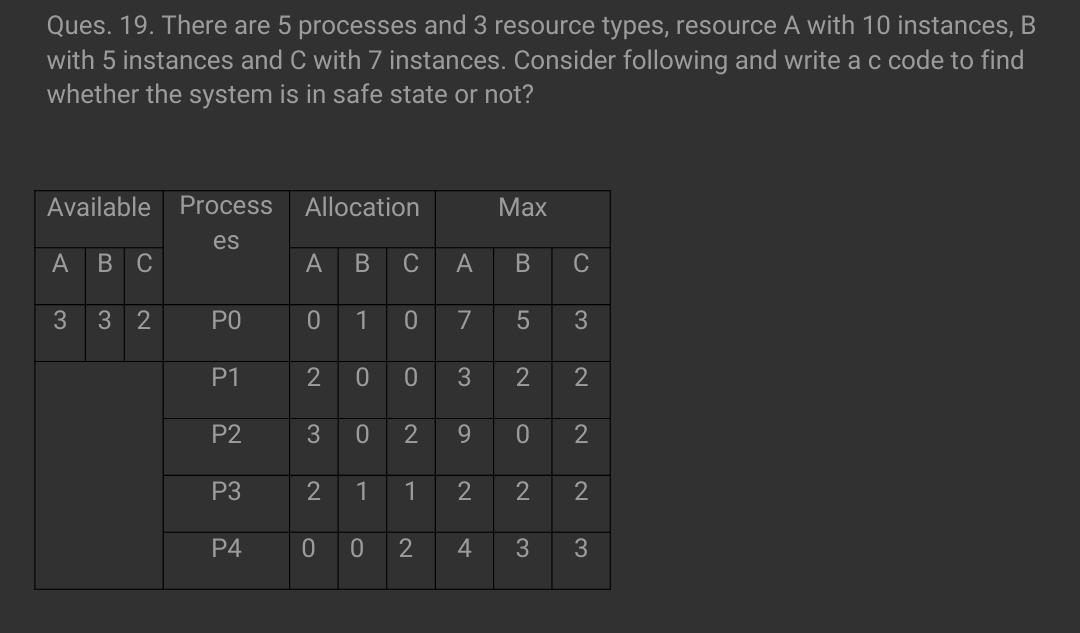
**SECTION:K18GE**

**GMAIL ID: [Adarsh.ag15@gmail.com](mailto:Adarsh.ag15@gmail.com)**

**GITHUB LINK:**

**QUES. CODE:Q19**

## PROBLEM:-



This process consists of deadlock avoidance , in order t do so the applicable concept is of ‘Banker’s algo.’.

**Concept:-**

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

**Data structures used:-**

n =no.of processes

m =no.of resources

**Available :**

>>It is a 1-d array of size **‘m’** indicating the number of available resources of each type.

>>Available[ j ] = k means there are **‘k’** instances of resource type ‘**Rj’.**

**Max :**

>>It is a 2-d array of size ‘**n\*m’** that defines the maximum demand of each process in a system.

>>Max[ i, j ] = k means process ‘**Pi’** may request at most **‘k’** instances of resource type ‘**Rj’.**

**Allocation :**

>>It is a 2-d array of size **‘n\*m’** that defines the number of resources of each type currently allocated to each process.

>>Allocation[ i, j ] = k means process ‘**Pi’** is currently allocated **‘k’** instances of resource type ‘**Rj’.**

**Need :**

>>It is a 2-d array of size **‘n\*m’** that indicates the remaining resource need of each process.

>>Need [ i,   j ] = k means process **Pi** currently need **‘k’** instances of resource type ‘**Rj’.**

for its execution.

>>Need [ i,   j ] = Max [ i,   j ] – Allocation [ i,   j ]

**Banker’s algorithm consists of Safety algorithm and Resource request algorithm**

**Safety Algorithm:**

1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.  
Initialize: Work = Available  
Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both  
a) Finish[i] = false  
b) Need i <= Work  
if no such i exists goto step (4)

3) Work = Work + Allocation[i]  
Finish[i] = true  
goto step (2)

4) if Finish [i] = true for all i  
then the system is in a safe state

**Resource-Request Algorithm:**

1) If Request i <= Need i  
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Request i <= Available  
Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as  
follows:  
Available = Available – Request i  
Allocation i = Allocation i + Request i  
Need i = Need i– Request i

## CODE:-

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the process names here

int n, m, i, j, k;

n = 5; // no. of processes

m = 3; // no. of resources

int alloc[5][3] = { { 0, 1, 0 }, // P0 // allocation matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } }; // P4

int max[5][3] = { { 7, 5, 3 }, // P0 // max matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // available resources

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

for (k = 0; k < 5; k++) {

for (i = 0; i < n; i++) {

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

printf(" SAFE sequence :-\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

return (0);

}