

BANKER'S ALGORITHM

Group members

Sudanshi Sehgal (2021A1R156)

Souhani Gandotra (2021A1R167)

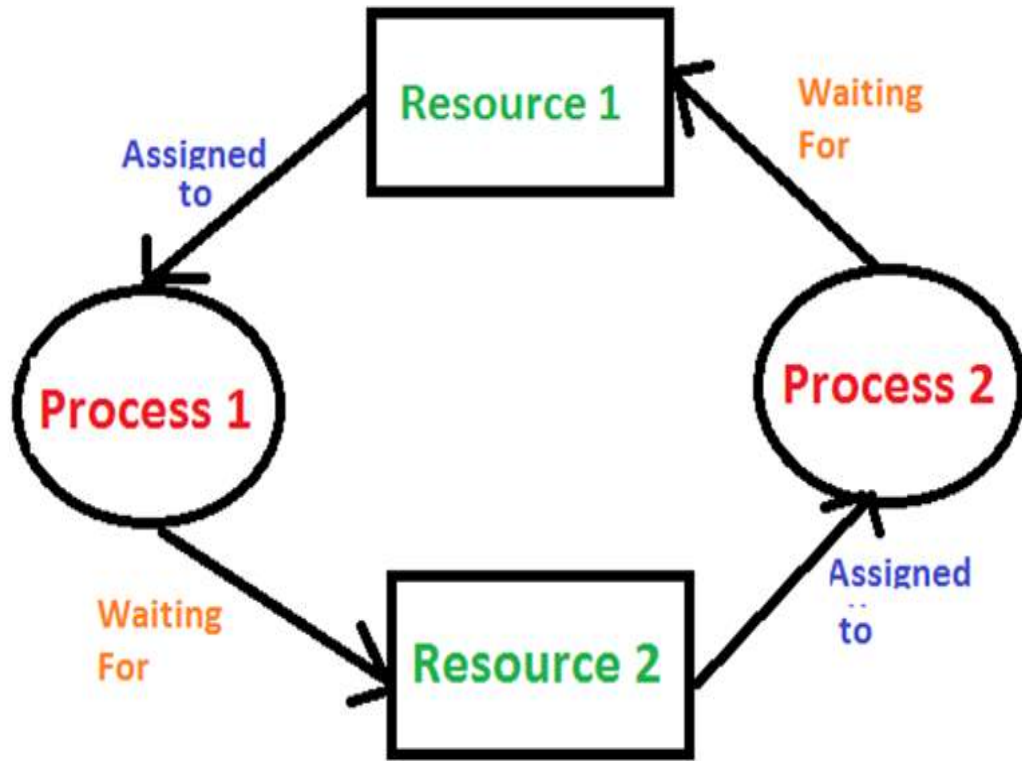
Simrat Singh (2021A1R170)

Anjali Salaria (2021A1R181)



Problem Statement:-

To implement the most representative banker's algorithm for
deadlock avoidance

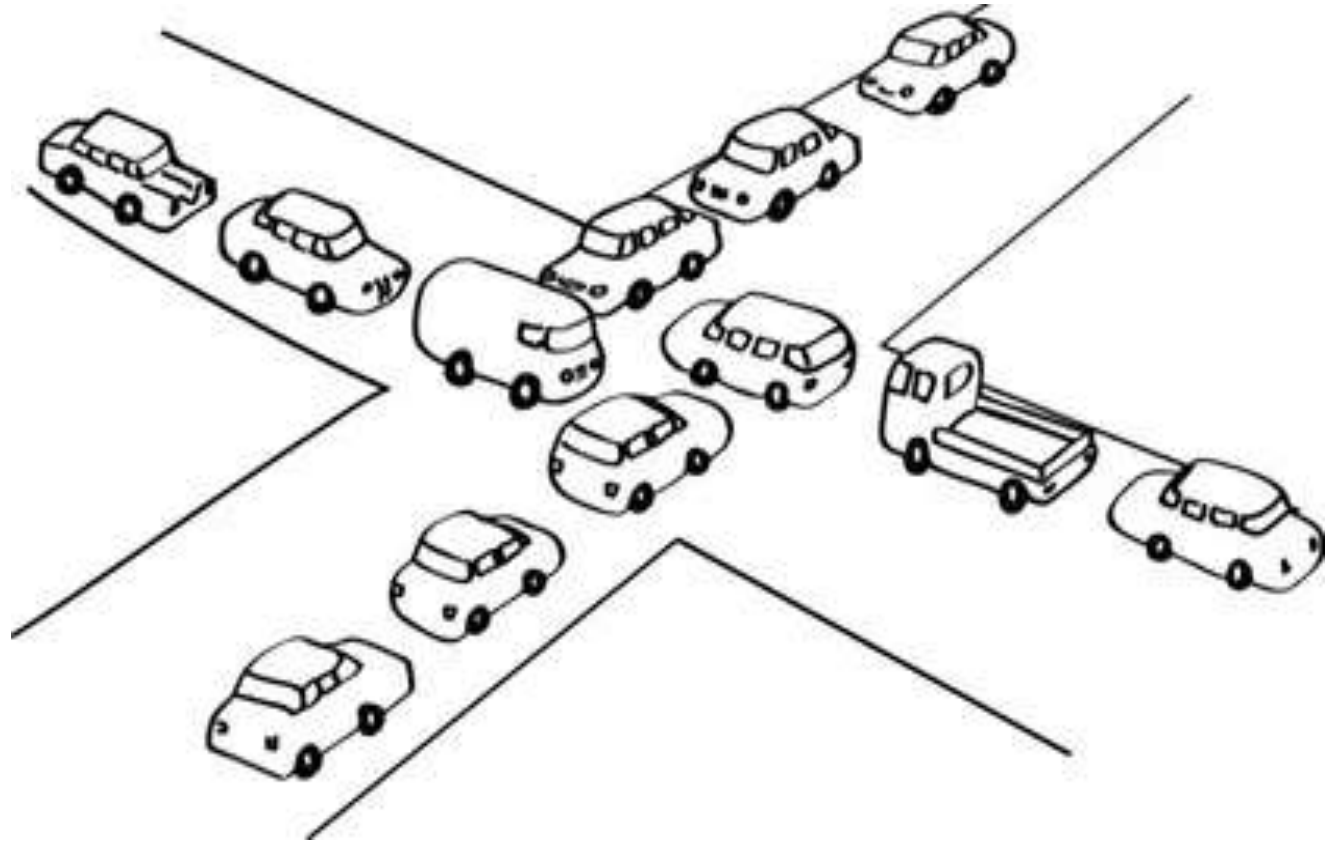


What is Deadlock?

- Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

A Real-World Example

- A real world example would be traffic, which is going only in one direction.
- Here, a bridge is considered a resource.
- So, when Deadlock happens, it can be easily resolved if one car backs up.
- Several cars may have to be backed up if a deadlock situation occurs.
- So starvation is possible.





The Conditions for Deadlock

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

1. Mutual Exclusion:

Only one process at a time may use a shared resource (i.e. critical section).

2. Hold and wait:

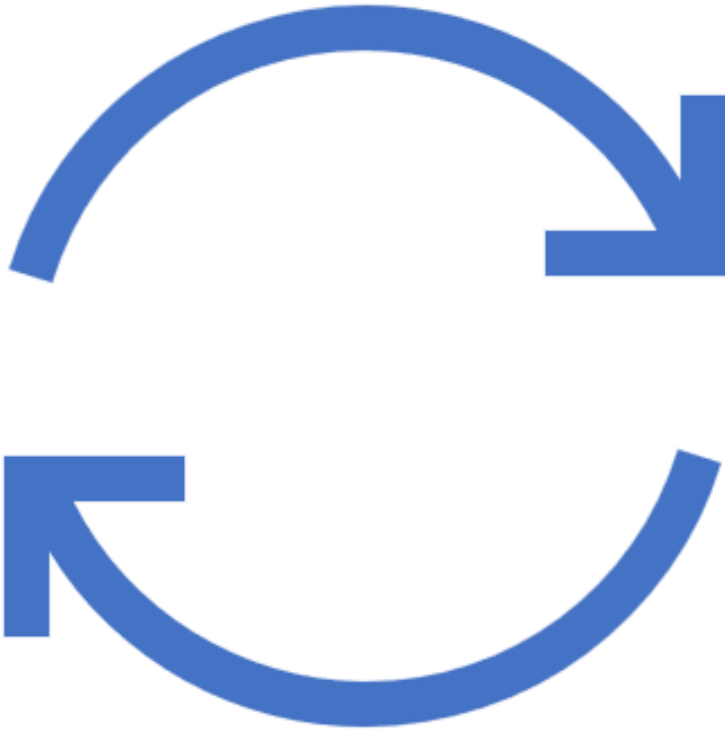
A process may hold allocated resources while awaiting assignment of others.

3. No pre-emption:

A resource can be released only voluntarily by the process holding it, after that process holding it, after that process has completed its task

4. Circular Wait:

A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.





The Conditions for Deadlock

- Deadlock occurs if and only if the circular wait condition is unresolvable
- The circular wait condition is unresolvable if the first 3 policy condition hold
- So the 4 conditions taken together constitute necessary and sufficient conditions for deadlock



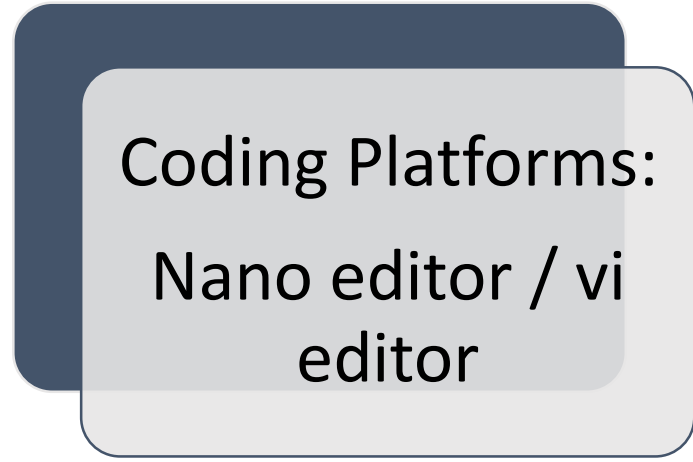
Requirements



Banker's
Algorithm



Linux Operating
System



Coding Platforms:
Nano editor / vi
editor

Banker's Algorithm

- Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resource, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it does not allow the request made by the process.
 - **Inputs to Banker's Algorithm:**
 1. Max need of resources by each process.
 2. Currently, allocated resources by each process.
 3. Max free available resources in the system.
 - **The request will only be granted under the below condition:**
 1. If the request made by the process is less than equal to max need to that process.
 2. If the request made by the process is less than equal to the freely available resource in the system.
-

Example of Banker's Algorithm

Example:

Considering a system with five processes P_0 through P_4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances.

Process	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	

Example (contd.)

Need = Max- Allocation

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

The system is in a safe state since the sequence $\langle P1, P3, P4, P2, P0 \rangle$ satisfies criteria.

Check that Request \leq Available
(that is, $(1,2,2) \leq (3,3,2)$) is true

Now P1 requests (1,2,2)

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	4	3	2	3	0
P1	3	0	2	0	2	0			
P2	3	0	2	6	0	0			
P3	2	1	1	0	1	1			
P4	0	0	2	4	3	1			

Pseudo Code:-

A safe sequence of processes,
or an empty sequence if no
such sequence exists.

- 1.while there exists a process P in N that has not been terminated do
- 2.if there exists a resource R in M that P can request and be granted without causing a deadlock then
- 3.request R from P
- 4.else if P can release a resource R without causing a deadlock then
- 5.release R from P
- 6.else
- 7.wait
- 8.end if
- 9.end while
- 10.return the safe sequence

Algorithm:

Safety Algorithm

1) Let Work and Finish be Arrays of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for $i=1, 2, 3, 4, \dots, n$

2) Find an i such that both

a) Finish[i] = false

b) Need i \leq Work

if no such i exists go to step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

Go to step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Algorithm:

- **Request Algorithm**

1) If Request $i \leq$ Need i

Go to step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Request $i \leq$ Available

Go to step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process P_i by modifying the state as

follows:

Available = Available – Request i

Allocation i = Allocation i + Request i

Need i = Need i – Request i

TECHNICAL DETAILS (CODING)

Activities Text Editor Dec 21 12:31 project1.c ~/Desktop


```
1 #include <stdio.h>
2
3 int m, n, i, j, al[10][10], max[10][10], av[10], need[10][10], temp, z, y, p, k;
4
5 void main() {
6
7     printf("\n Enter no of processes : ");
8     scanf("%d", &m); // enter numbers of processes
9
10    printf("\n Enter no of resources : ");
11    scanf("%d", &n); // enter numbers of resources
12
13    for (i = 0; i < m; i++) {
14        for (j = 0; j < n; j++) {
15            printf("\n Enter instances for al[%d][%d] = ", i,j); // al[][] matrix is for allocated instances
16            scanf("%d", &al[i][j]);
17            // al[i][j]=temp;
18        }
19    }
20
21    for (i = 0; i < m; i++) {
22        for (j = 0; j < n; j++) {
23            printf("\n Enter instances for max[%d][%d] = ", i,j); // max[][] matrix is for max instances
24            scanf("%d", &max[i][j]);
25        }
26    }
27
28    for (i = 0; i < n; i++) {
29        printf("\n Available Resource for av[%d] = ",i); // av[] matrix is for available instances
30        scanf("%d", &av[i]);
31    }
32
33    // Print allocation values
34    printf("Allocation Values :\n");
35    for (i = 0; i < m; i++) {
36        for (j = 0; j < n; j++) {
37            printf(" \t %d", al[i][j]); // printing allocation matrix
38        }
39        printf("\n");
40    }
41
42    printf("\n\n");
43
44    // Print max values
```



Activities Text Editor

Open

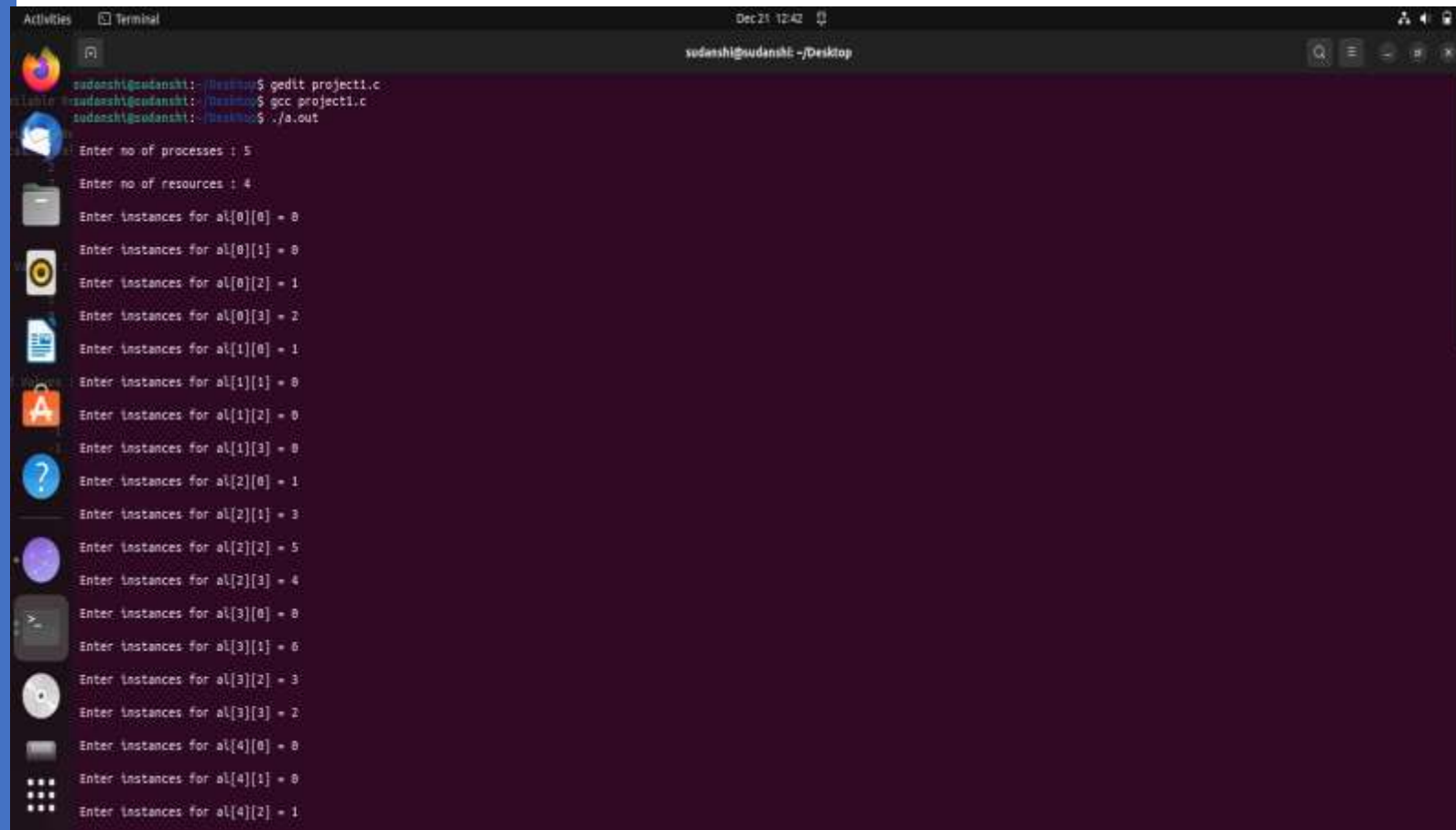
```
44 // Print max values
45 printf("Max Values :\n");
46 for (i = 0; i < m; i++) {
47     for (j = 0; j < n; j++) {
48         printf(" \t %d", max[i][j]); // printing max matrix
49     }
50     printf("\n");
51 }
52
53 printf("\n\n");
54
55 // Print need values
56 printf("Need Values :\n");
57 for (i = 0; i < m; i++) {
58     for (j = 0; j < n; j++) {
59         need[i][j] = max[i][j] - al[i][j]; // calculating need matrix
60         printf("\t %d", need[i][j]); // printing need matrix
61     }
62     printf("\n");
63 }
64
65 p = 1; // used for terminating while loop
66 y = 0;
67 while (p != 0) {
68     for (i = 0; i < m; i++) {
69         z = 0;
70         for (j = 0; j < n; j++) {
71             if (need[i][j] <= av[j] &&
72                 (need[i][0] != -1)) { // comparing need with available instance and
73                                     // checking if the process is done
74                                     // or not
75                                     // counter if condition TRUE
76                                     z++;
77             }
78         }
79         if (z == n) { // if need<=available TRUE for all resources then condition
80                     // is TRUE
81
82                     for (k = 0; k < n; k++) {
83                         av[k] += al[i][k]; // new work = work + allocated
84                     }
85                     printf("\n Executed process %d", i); // Print the Process
86                     need[i][0] = -1; // assign -1 if Process done
87                     // cont if process done
88 }
```

```
85     }
86     printf("\n Executed process %d", i); // Print the Process
87     need[i][0] = -1;                    // assign -1 if Process done
88     y++;                                // cont if process done
89     }
90
91 } // end for loop
92
93 if (y == m) { // if all done then
94     p = 0;    // exit while loop
95 }
96
97 } // end while
98
99 printf("\n");
100
101 }
```

OUTPUT

NO DEADLOCK



```
Activities Terminal Dec 21 12:42 sudanishi@sudanishi: ~/Desktop
sudanishi@sudanishi: ~/Desktop$ gedit project1.c
sudanishi@sudanishi: ~/Desktop$ gcc project1.c
sudanishi@sudanishi: ~/Desktop$ ./a.out
Enter no of processes : 5
Enter no of resources : 4
Enter instances for a[0][0] = 0
Enter instances for a[0][1] = 0
Enter instances for a[0][2] = 1
Enter instances for a[0][3] = 2
Enter instances for a[1][0] = 1
Enter instances for a[1][1] = 0
Enter instances for a[1][2] = 0
Enter instances for a[1][3] = 0
Enter instances for a[2][0] = 1
Enter instances for a[2][1] = 3
Enter instances for a[2][2] = 5
Enter instances for a[2][3] = 4
Enter instances for a[3][0] = 0
Enter instances for a[3][1] = 0
Enter instances for a[3][2] = 3
Enter instances for a[3][3] = 2
Enter instances for a[4][0] = 0
Enter instances for a[4][1] = 0
Enter instances for a[4][2] = 1
```

Activities Terminal Dec 21 12:43 sudanshi@sudanshi: ~/Desktop

```
Enter instances for a1[4][2] = 1
Enter instances for a1[4][3] = 4
Enter instances for max[0][0] = 0
Enter instances for max[0][1] = 0
Enter instances for max[0][2] = 1
Enter instances for max[0][3] = 2
Enter instances for max[1][0] = 1
Enter instances for max[1][1] = 7
Enter instances for max[1][2] = 5
Enter instances for max[1][3] = 6
Enter instances for max[2][0] = 2
Enter instances for max[2][1] = 3
Enter instances for max[2][2] = 5
Enter instances for max[2][3] = 0
Enter instances for max[3][0] = 0
Enter instances for max[3][1] = 0
Enter instances for max[3][2] = 5
Enter instances for max[3][3] = 2
Enter instances for max[4][0] = 0
Enter instances for max[4][1] = 0
Enter instances for max[4][2] = 5
Enter instances for max[4][3] = 0
Available Resource for av[0] = 8
```

```
Activities Terminal Dec 21 12:43 sudanshi@sodanshi: ~/Desktop

Enter instances for max[4][0] = 0
Enter instances for max[4][1] = 0
Enter instances for max[4][2] = 5
Enter instances for max[4][3] = 0
Available Resource for av[0] = 8
Available Resource for av[1] = 8
Available Resource for av[2] = 8
Available Resource for av[3] = 8
Allocation Values :
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4

Max Values :
0 0 1 2
1 7 5 6
2 3 5 0
0 6 5 2
0 6 5 0

Need Values :
0 0 0 0
0 7 5 6
1 0 0 2
0 0 2 0
0 0 4 2

Executed process 0
Executed process 1
Executed process 2
Executed process 3
Executed process 4
sudanshi@sodanshi: ~/Desktop
```

OUTPUT

DEADLOCK

```
Activities  Terminal  Dec 21 12:44  sudasht@sudasht: ~/Desktop
sudasht@sudasht:~/Desktop$ gedit project1.c
sudasht@sudasht:~/Desktop$ gcc project1.c
sudasht@sudasht:~/Desktop$ ./a.out
Enter no of processes : 4
Enter no of resources : 3
Enter instances for a1[0][0] = 2
Enter instances for a1[0][1] = 3
Enter instances for a1[0][2] = 2
Enter instances for a1[1][0] = 2
Enter instances for a1[1][1] = 1
Enter instances for a1[1][2] = 3
Enter instances for a1[2][0] = 1
Enter instances for a1[2][1] = 2
Enter instances for a1[2][2] = 3
Enter instances for a1[3][0] = 1
Enter instances for a1[3][1] = 2
Enter instances for a1[3][2] = 3
Enter instances for max[0][0] = 1
Enter instances for max[0][1] = 2
Enter instances for max[0][2] = 1
Enter instances for max[1][0] = 1
Enter instances for max[1][1] = 0
Enter instances for max[1][2] = 2
Enter instances for max[2][0] = 0
```

```
ACMIDES Terminal Dec 21 12:45 sudanshi@sudanshi ~/Desktop

Enter instances for max[0][2] = 1
Enter instances for max[1][0] = 1
Enter instances for max[1][1] = 0
Enter instances for max[1][2] = 2
Enter instances for max[2][0] = 0
Enter instances for max[2][1] = 1
Enter instances for max[2][2] = 2
Enter instances for max[3][0] = 0
Enter instances for max[3][1] = 1
Enter instances for max[3][2] = 2
Available Resource for av[0] = 5
Available Resource for av[1] = 5
Available Resource for av[2] = 5
Allocation Values :
  2  3  2
  2  1  3
  4  2  3
  1  2  3

Max Values :
  1  2  1
  1  0  2
  0  1  2
  0  1  2

Need Values :
  -1  -1  -1
  -1  -1  -1
  -1  -1  -1
  -1  -1  -1
```

”

THANK YOU

