NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION, AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, APPROVED BY AICTE & GOVT.OF KARNATAKA

COURSE LA2 REPORT

on

BANKERS (RESOURCE-REQUEST) ALGORITHM

Submitted in partial fulfilment of the requirement for the award of Degree of

Bachelor of Engineering

in

Computer Science and Engineering

Submitted by:

Anurag Nepal	1NT19CS036
Avishek Rijal	1NT19CS045
Baibhav Dhakal	1NT19CS048
Nabin Kumar K.C.	1NT19CS116

Department of Computer Science and Engineering

Nitte Meenakshi Institute of Technology

(AN AUTONOMOUS INSTITUTION AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM)
PB No. 6429, Yelahanka, Bangalore 560-064, Karnataka
Telephone: 080- 22167800, 22167860
Fax: 080 – 22167805

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
2020-21

CERTIFICATE

This is to certify that the Course Project titled "Banker's Algorithm(Resource-Request)" is an authentic work carried out by **Anurag Nepal(1NT19CS0363), Avishek Rijal(1NT19CS045), Baibhav Dhakal(1NT19CS048), Nabin Kumar K.C.(1NT19CS116).** Bonafede students of **Nitte Meenakshi Institute of Technology**, Bangalore in partial fulfilment for the award of the degree of ***Bachelor of Engineering*** in COMPUTER SCIENCE AND ENGINEERING of Visvesvaraya Technological University, Belagavi during the academic year ***2020-21***

Name Signature of the Faculty In charge **Name and Signature of the HOD**

DECLARATION

We hereby declare that

(i)This Report does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the report and in the References sections.
(ii) All corrections and suggestions indicated during the internal presentation have been incorporated in the report.
(iii) Content of the report has been checked for the plagiarism requirement

Name	USN	Signature
Anurag Nepal	1NT19CS036	
Avishek Rijal	1NT19CS045	
Baibhav Dhakal	1NT19CS048	
Nabin Kumar K.C.	1NT19CS116	

Date:

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our effort with success. We express our sincere gratitude to our Principal **Dr. H. C. Nagaraj**, Nitte Meenakshi Institute of Technology for providing facilities.

We thank our HoD**, Dr.Thippeswamy M.N** for the excellent environment created to further educational growth in our college. We also thank him for the invaluable guidance provided which has helped in the creation of a better technical report.

Thanks to our Subject Faculty, Deepthi Shetty.We also thank all our friends, teaching and non-teaching staff at NMIT, Bangalore, for all the direct and indirect help provided in the completion of the project.

Name	USN	Signature
Anurag Nepal	1NT19CS036	
Avishek Rijal	1NT19CS045	
Baibhav Dhakal	1NT19CS048	
Nabin Kumar K.C.	1NT19CS116	

Date:

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

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.

Following are the important data structures terms applied in the banker's algorithm:

Suppose n is the number of processes, and m is the number of each type of resource used in a computer system.

1. **Available**: It is an array of length 'm' that defines each type of resource available in the system. When Available[j] = K, means that 'K' instances of Resources type R[j] are available in the system.
2. **Max:** It is a [n x m] matrix that indicates each process P[i] can store the maximum number of resources R[j] (each type) in a system.
3. **Allocation:** It is a matrix of m x n orders that indicates the type of resources currently allocated to each process in the system. When Allocation [i, j] = K, it means that process P[i] is currently allocated K instances of Resources type R[j] in the system.
4. **Need:** It is an M x N matrix sequence representing the number of remaining resources for each process. When the Need[i] [j] = k, then process P[i] may require K more instances of resources type Rj to complete the assigned work.
   Need[i][j] = Max[i][j] - Allocation[i][j].
5. **Finish**: It is the vector of the order **m**. It includes a Boolean value (true/false) indicating whether the process has been allocated to the requested resources, and all resources have been released after finishing its task.

The Banker's Algorithm is the combination of the safety algorithm and the resource request algorithm to control the processes and avoid deadlock in a system:

Advantages of Banker's Algorithm:

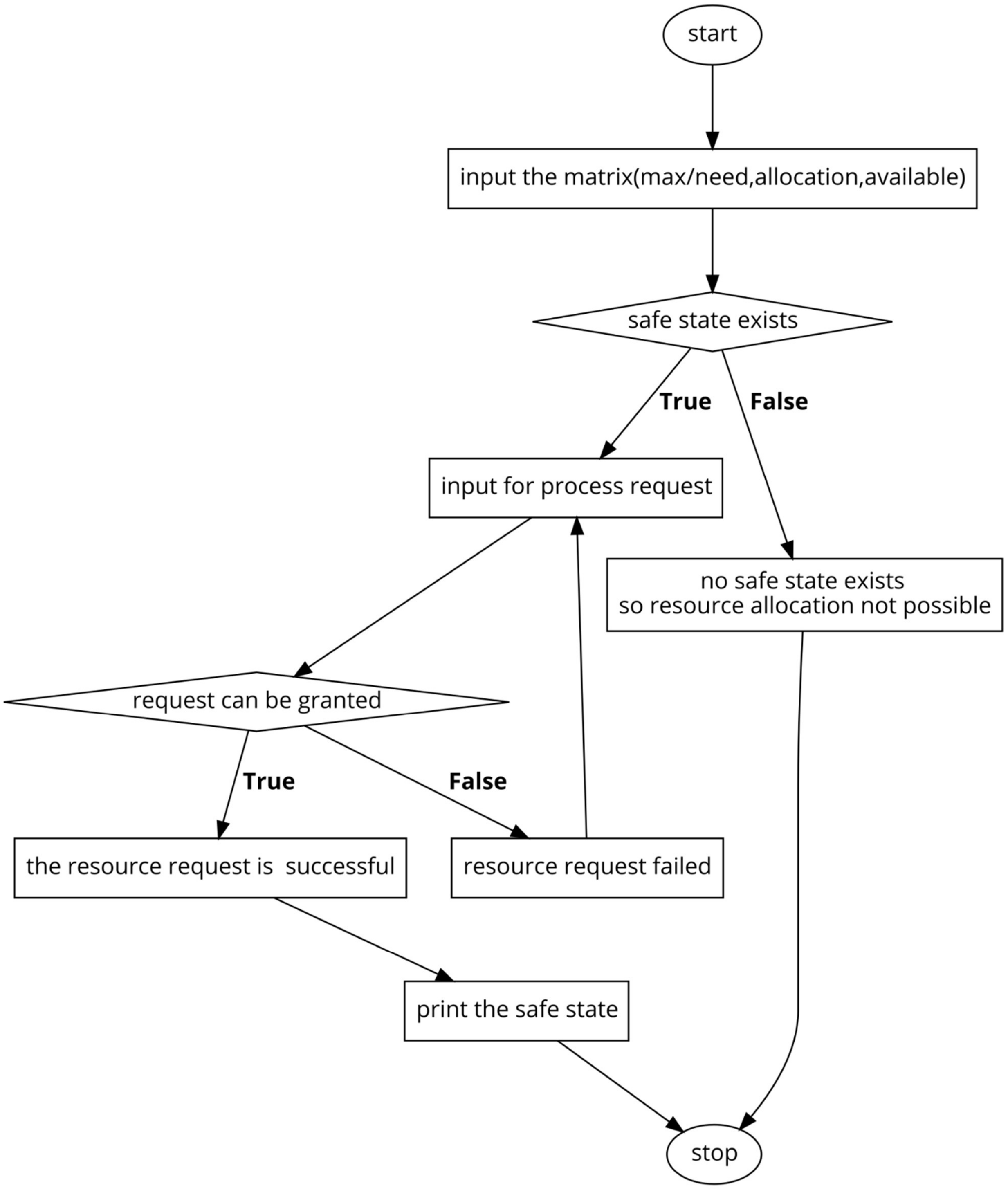
Here are the benefits/pros of using Banker's Algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that indicates each process can hold the maximum number of resources in a system.

Disadvantages of Banker's Algorithm:

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

Chapter 2 : FLOWCHART


start
input the matrix(max/need,allocation,available)
safe state exists
True
False
input for process request
no safe state exists
so resource allocation not possible
request can be granted
True
False
the resource request is successful
resource request failed
print the safe state
stop

Chapter 3: CODE

```
/*The program provided down is Bankers Algorithm(Resource-Request)*/

// C program to implement Shortest Job first with Arrival Time

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

        int alloc[10][10],max[10][10],need[10][10];

        int avble[1][10];

        int exct[10];

        int flag=0,flag_safe=0;

        int n,m,i,j,a[10],k,x=0;

  char input[4];

  int reqmat[1][10];

  int ret, pid;

  int safety(int aloc[][10],int neede[][10],int avble[1][10],int n,int m,int exct[]){

        int able[1][10];

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

                exct[i]=0;

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

                able[0][i]=avble[0][i];


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

                for(i=0;i<n;i++){
                        if(exct[i]==0){

                                flag=0;//reset flag

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

                                        if(neede[i][j]>able[0][j]) flag=1;

                                }

                                if(flag==0&&exct[i]==0){

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

      {

       able[0][j]+=alloc[i][j];

      }

                                        exct[i]=1;//executed

                                        flag_safe++;//Count the number of processes executed

                                        a[x++]=i;

                                }

                        }

                }

        }

 return flag_safe;

 }


int res_request(int alloc[][10],int need[][10],int avble[1][10],int pid,int m){

        int reqmat[1][10];

        int aloc[10][10], neede[10][10],able[1][10];

        int i;
        int id=pid-1;

        printf("\n Enter request :- \n");

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

                scanf("%d",&reqmat[0][i]);

        }

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

                able[0][i]=avble[0][i];


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

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

        aloc[i][j]=alloc[i][j];

        neede[i][j]=need[i][j];

        }}


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

                if(reqmat[0][i] > neede[id][i]){

                        printf("\nRequest greater than need. Exiting...\n");

                        return(-1);

        }


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

                if(reqmat[0][i] > able[0][i]){

                        printf("\n Resources unavailable.Exiting...\n");

                        return (-1);
                }

        for(i=0;i<m;i++){
                able[0][i]-=reqmat[0][i];
                aloc[id][i]+=reqmat[0][i];
                neede[id][i]-=reqmat[0][i];
        }
        int ans=safety(aloc,neede,able,n,m,exct);
        return ans;
}


int main(){

        printf("\n DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM\n");

        printf("\n Enter total no. of processes: ");
        scanf("%d",&n);

        printf("\n Enter total no. of resources: ");
        scanf("%d",&m);

  printf("\n Enter the Allocation Matrix:");
        for(i=0;i<n;i++){
```

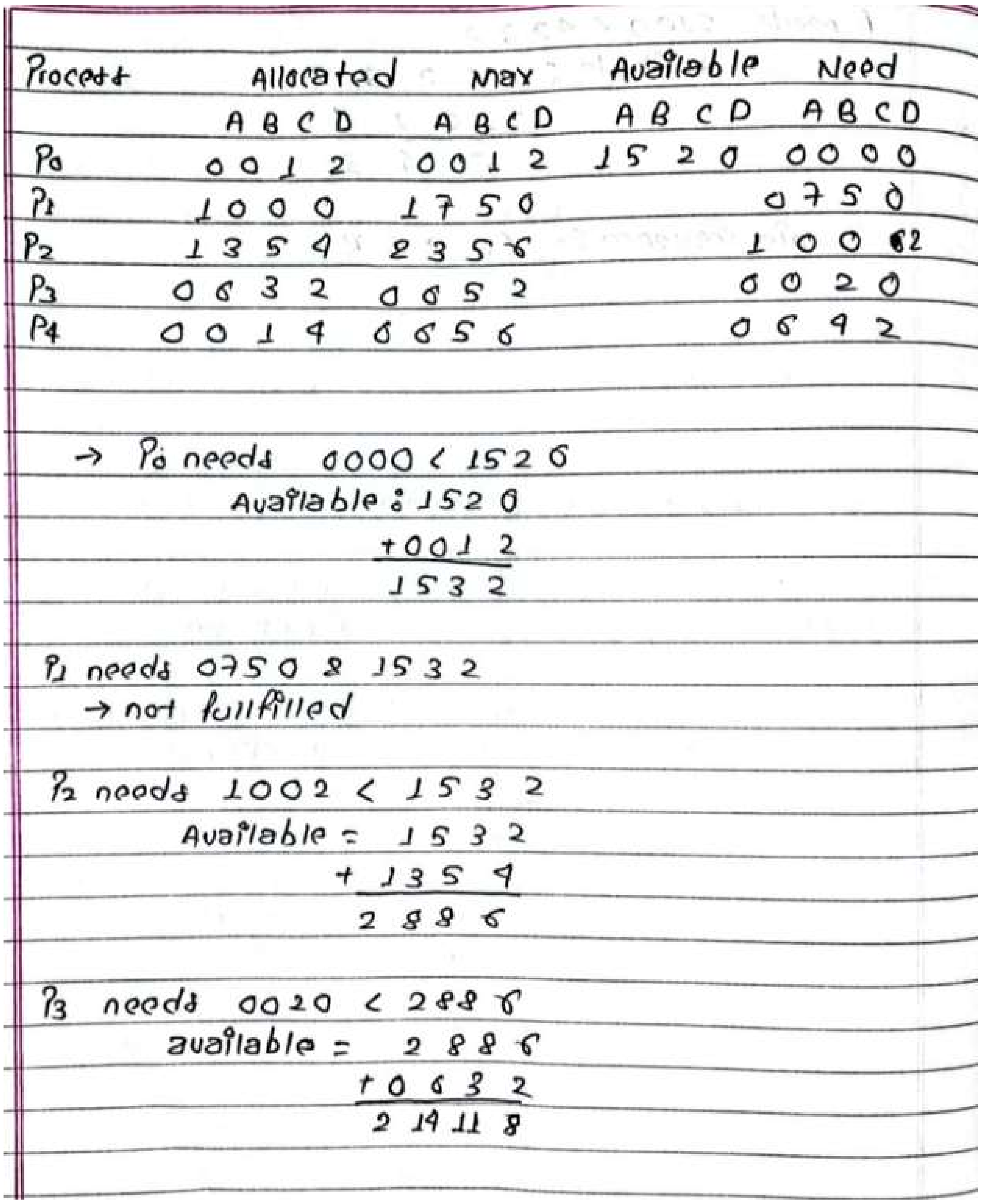
```
                    for(j=0;j<m;j++){

                              scanf("%d",&alloc[i][j]);

  }

}

retry:

printf("\nDo you want to enter the max matrix or the need matrix?: ");

scanf("%s",input);


for (i = 0;i<strlen(input);i++)

{

  input[i] = tolower((unsigned char)input[i]);

}

if(strcmp(input,"max")==0)

{

  printf("\nEnter the Maximum Matrix: ");

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

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

                              scanf("%d",&max[i][j]);

                    }

          }


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

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

                              need[i][j]=max[i][j]-alloc[i][j];//Calculate Need Matrix
```

```
}


else if(strcmp(input,"need")==0)

{

  printf("\nEnter the Need Matrix:");



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

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

     {scanf("%d",&need[i][j]);

                       }

}

else

{

 printf("Not a valid choice");

 goto retry;

}



           printf("\nEnter the Available resources:");

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

                      scanf("%d",&avble[0][i]);

           }


           printf("\n Allocation Matrix:");

           for(i=0;i<n;i++){
                printf("\n");
                for(j=0;j<m;j++){
                        printf("%d\t",alloc[i][j]);
                }
        }

if(strcmp(input,"max")==0)
        {
 printf("\n Maximum Matrix:");
         for(i=0;i<n;i++)
  {
                 printf("\n");
                 for(j=0;j<m;j++)
   {
                         printf("%d\t",max[i][j]);
                 }
         }
}

        printf("\n Need Matrix:");
        for(i=0;i<n;i++)
{
                printf("\n");
                for(j=0;j<m;j++)
 {
                          printf("%d\t",need[i][j]);
                 }
         }

ret=safety(alloc,need,avble,n,m,exct);
if(ret!=0)
{ printf("\n A safe sequence has been detected");
         again: printf("\nDo you want to request for any processes(YES||NO): ");
 scanf("%s",input);

 for (i = 0;i<strlen(input);i++)
 {
  input[i] = tolower((unsigned char)input[i]);
 }
 if (strcmp(input,"yes")==0)
 {
  printf("\nEnter your request:");
                 printf("\n Enter Process no.: ");
                 scanf("%d",&pid);
                 int check=res_request(alloc,need,avble,pid,m);
                 if(check==-1)  goto again;
           if(check!=0)
   {
        printf("\n A safe sequence has been detected.No Deadlocks\n");
                        for(i=0;i<n;i++)
        {
          printf("P%d",a[i]);
          if(i!=n-1) printf("--->");
        }
      }
    }
    else
    {
      printf("\n A safe sequence has been detected.No Deadlocks\n");
                      for(i=0;i<n;i++)
      {
        printf("P%d",a[i]);
        if(i!=n-1) printf("--->");
      }
    }
  }
         else
  {
                      printf("\n Deadlock has occured.Exiting...\n");
                      exit(0);
            }
}
```

Chapter 4: Code Explanation

We have imported the required header files for the various operations needed during the program.We have an executed status matrix and flag to check if the process can be executed and we have flag_safe to count the number of executed processes. Then, we have a basic variable for the loop to be executed and an array to store the safe sequence, after that we ask for user inputs on max/need and yes or no. At line eleven is the resource request followed by basic variables on line twelve. Then, we take input form the user for the resource input. After we take the input, we check the first condition if the request is greater than need, we exit. After that we check the second condition if the request is greater than available, we exit. After that we have a for loop to check if the conditions satisfy, if it does, we find the new values for available allocated and need for the process. Then we check the safety state processes in one pass even if a process is skipped, we can get back to it without leaving the third loop. Then we have an exct=0 condition meaning the process hasn't been executed yet, so we reset the flag by putting the flag as 0. Furthermore we check if the need is greater than available then we set the flag as 1 which means it cannot be executed yet furthermore if the flag is zero and hasn't been executed it simply means we can execute the certain process. After the execution we add the available and allocated to form the new available vector. After that is done, we mark the current process as executed and with the help of exct it marks the current process as executed. After that is done flag_Safe++ is used to increment the counter to count the number of processes executed and a[x++]=i is used to store the process index in the answer array. After that with the help of return flag_safe we return the number of processes executed. After that we move to the main part of the program where we ask for the user to enter the number of processes, after that we take input for the total number of resources, furthermore we take input for the allocation matrix and we ask for need or max. When we do that, we change the input to lowercase just in case someone enters the input in uppercase as described in the header part. We then use stringcompare to check if max is entered and ask for the max matrix and calculate the need matrix. We also then calculate the need matrix while putting the condition to print if the wrong choice is given. Accordingly we take the available resources input and print the allocated matrix. After that we see if the max matrix is entered. If it is true, we print the max then we print the need matrix. After that process is done, we ask if the process needs additional resources or not if it does then we ask for a request and take input for the request. After that we ask for the process id and then run a resource request module to find if it can be granted. Furthermore, we check the safety state from the new values and if the safety module returns the same number as there are number of processes there is a safe space else it is on deadlock.

Solved Examples:



Process	Allocated	Max	Available	Need
	A B C D	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0	0 0 0 0
P1	1 0 0 0	1 7 5 0		0 7 5 0
P2	1 3 5 4	2 3 5 6		1 0 0 2
P3	0 6 3 2	0 6 5 2		0 0 2 0
P4	0 0 1 4	0 6 5 6		0 6 4 2

→ P0 needs 0000 < 1520

Available : 1520
+0012
1532

P1 needs 0750 $\nless$ 1532
→ not fullfilled

P2 needs 1002 < 1532

Available = 1532
+ 1354
2886

P3 needs 0020 < 2886

available = 2886
+0632
2 14 11 8

P_4 needs 0 6 4 2 $\leq$ 2 14 11 8

$\therefore$ Available =

```
   2  14  11  8
 + 0   0   1  4
 --------------
   2  14  12  12
```

P_1 needs 0 7 5 0 < 2 14 12 12

Available

```
   2  14  12  12
 + 1   0   0   0
 ---------------
   3  14  12  12
```

$\therefore$ safe sequence = P_0 P_2 P_3 P_4 P_1

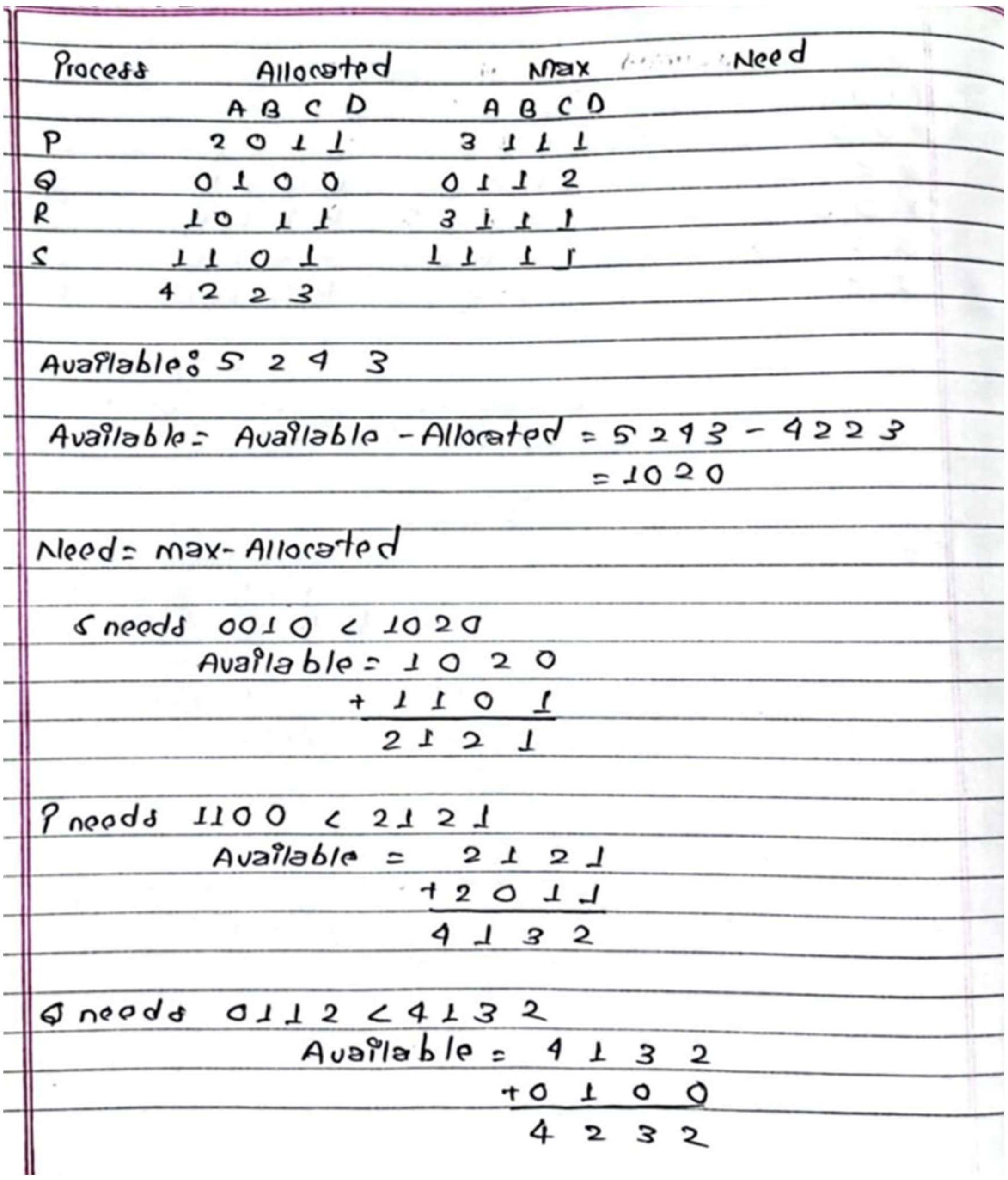
$\Rightarrow$ A request for (0,4,2,0)

S_1 = If request < need
i.e (0,4,2,0) $\leq$ (0,7,5,0)
$\rightarrow$ satisfied

S_2 : If request $\leq$ available
i.e (0,4,2,0) $\leq$ (1,5,2,0)
$\rightarrow$ satisfied.

S_3 : Available = (1,5,2,0) - (0,4,2,0)
= (1,1,0,0)

Need = ~~(0,4,2,0)~~ (0,7,5,0) - (0,4,2,0)
=> (0,3,3,0)



Process	Allocated	Max	Need
	A B C D	A B C D	
P	2 0 1 1	3 1 1 1	
Q	0 1 0 0	0 1 1 2	
R	1 0 1 1	3 1 1 1	
S	1 1 0 1	1 1 1 1	
	4 2 2 3		

Available: 5 2 4 3

Available = Available - Allocated = 5 2 4 3 - 4 2 2 3
= 1 0 2 0

Need = max - Allocated

S needs 0 0 1 0 < 1 0 2 0

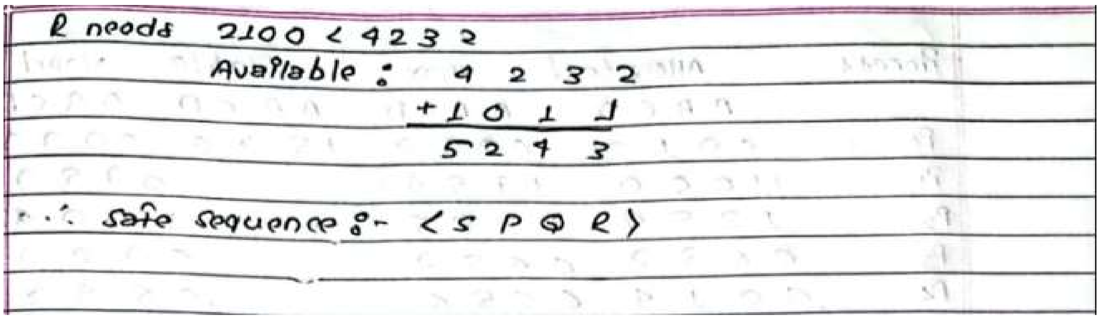
```
Available = 1 0 2 0
          + 1 1 0 1
          ---------
            2 1 2 1
```

P needs 1 1 0 0 < 2 1 2 1

```
Available =  2 1 2 1
            +2 0 1 1
            --------
             4 1 3 2
```

Q needs 0 1 1 2 < 4 1 3 2

```
Available =  4 1 3 2
            +0 1 0 0
            --------
             4 2 3 2
```

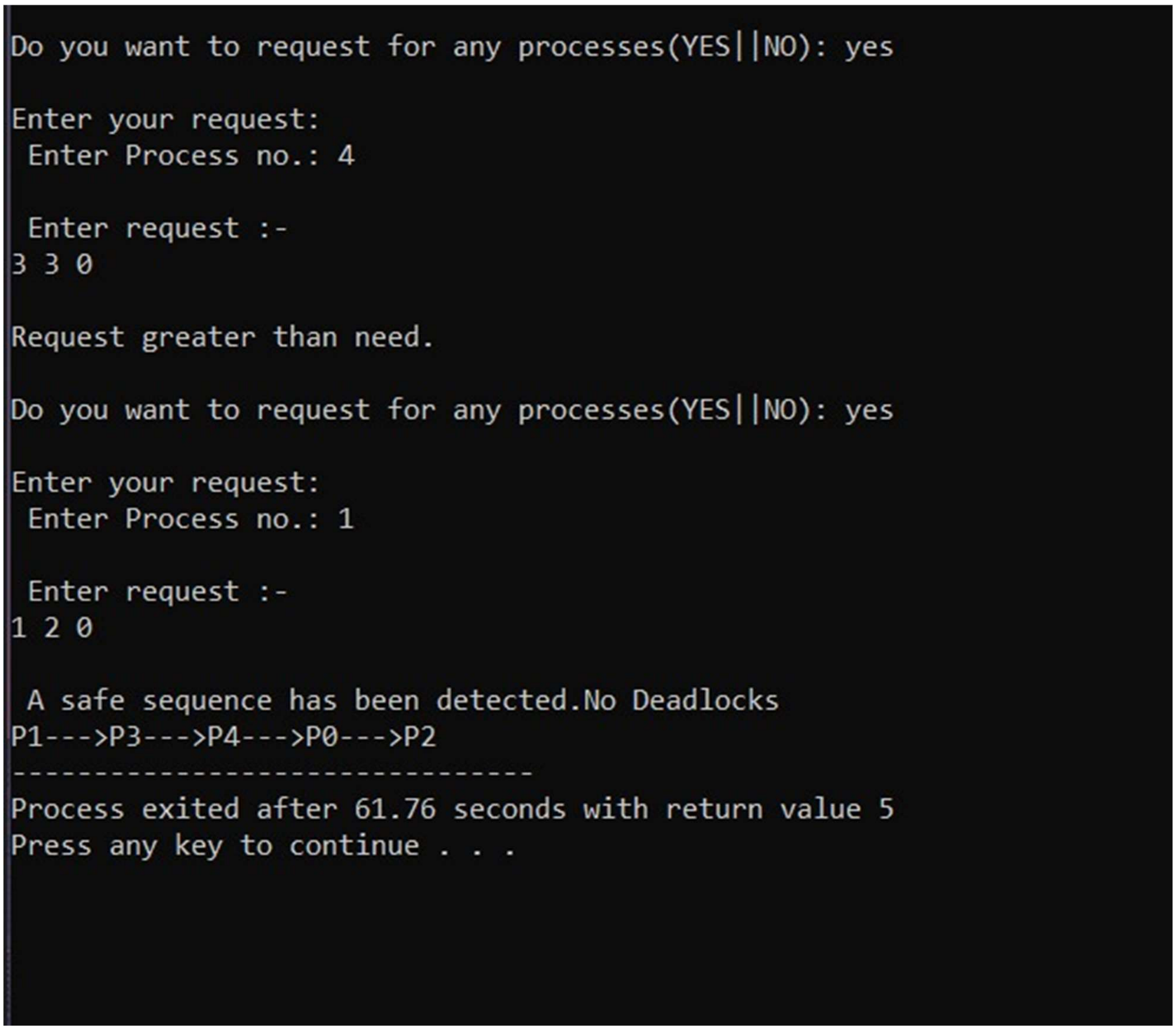


R needs 2 1 0 0 < 4 2 3 2

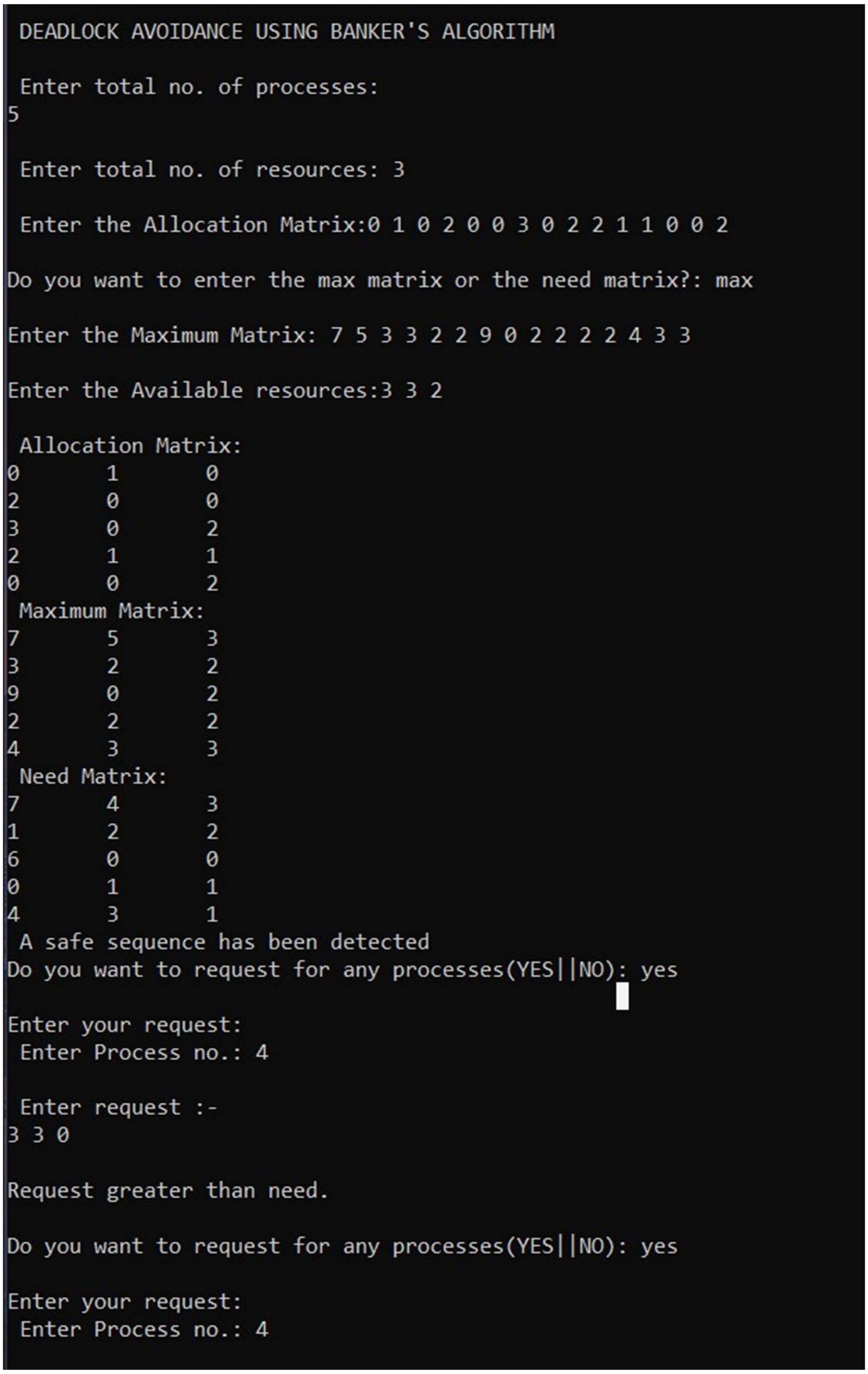
```
Available :  4 2 3 2
            +1 0 1 1
            --------
             5 2 4 3
```

∴ Safe sequence :- < S P Q R >

Chapter 5:Screenshots



```
Do you want to request for any processes(YES||NO): yes

Enter your request:
 Enter Process no.: 4

 Enter request :-
3 3 0

Request greater than need.

Do you want to request for any processes(YES||NO): yes

Enter your request:
 Enter Process no.: 1

 Enter request :-
1 2 0

 A safe sequence has been detected.No Deadlocks
P1--->P3--->P4--->P0--->P2
--------------------------------
Process exited after 61.76 seconds with return value 5
Press any key to continue . . .
```



```
 DEADLOCK AVOIDANCE USING BANKER'S ALGORITHM

 Enter total no. of processes:
5

 Enter total no. of resources: 3

 Enter the Allocation Matrix:0 1 0 2 0 0 3 0 2 2 1 1 0 0 2

Do you want to enter the max matrix or the need matrix?: max

Enter the Maximum Matrix: 7 5 3 3 2 2 9 0 2 2 2 2 4 3 3

Enter the Available resources:3 3 2

 Allocation Matrix:
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
 Maximum Matrix:
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3
 Need Matrix:
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
 A safe sequence has been detected
Do you want to request for any processes(YES||NO): yes

Enter your request:
 Enter Process no.: 4

 Enter request :-
3 3 0

Request greater than need.

Do you want to request for any processes(YES||NO): yes

Enter your request:
 Enter Process no.: 4
```

Chapter6: Hardware and Software Requirement

HARDWARE REQUIREMENTS

Processor: Intel Pentium IV

RAM: 512 MB

Hard Disk: 40GB

SOFTWARE REQUIREMENTS

Operating System: Windows 98, 2000, XP, 7, 8, 8.1, 10

Tools: Dev-C++ 5.11

Technologies: DOS 7.0

Chapter 7: References

https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system-2/

https://www.javatpoint.com/bankers-algorithm-in-operating-system

Computer Concepts And C Programming (As Per VTU) **Book by Rajiv Khanna**