5rName:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	S. Y. B.Tech (Computer Engineering)
Course:	Operating System Laboratory
Course Code:	DJ19CEL403
Experiment No.:	06

AIM: **BANKER's ALGORITHM**

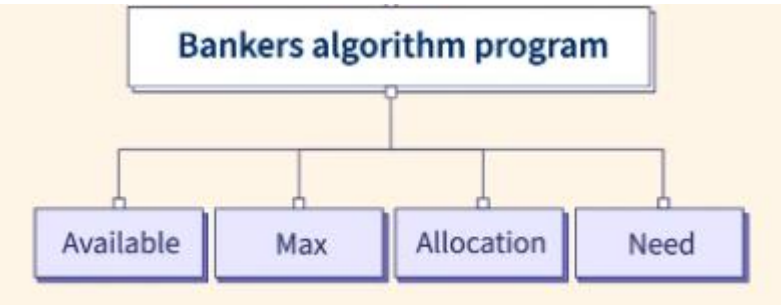
THEORY:

Why do we need Banker's Algorithm?

- A process in OS can request a resource, can use the resource, and can also release it.
- There comes a situation of deadlock in OS in which a set of processes is blocked because it is holding a resource and also requires some other resources at the same time that are being acquired by other processes.
- So to avoid such a situation of deadlock, we have the Bankers algorithm in Operating System.

What is Banker's Algorithm?

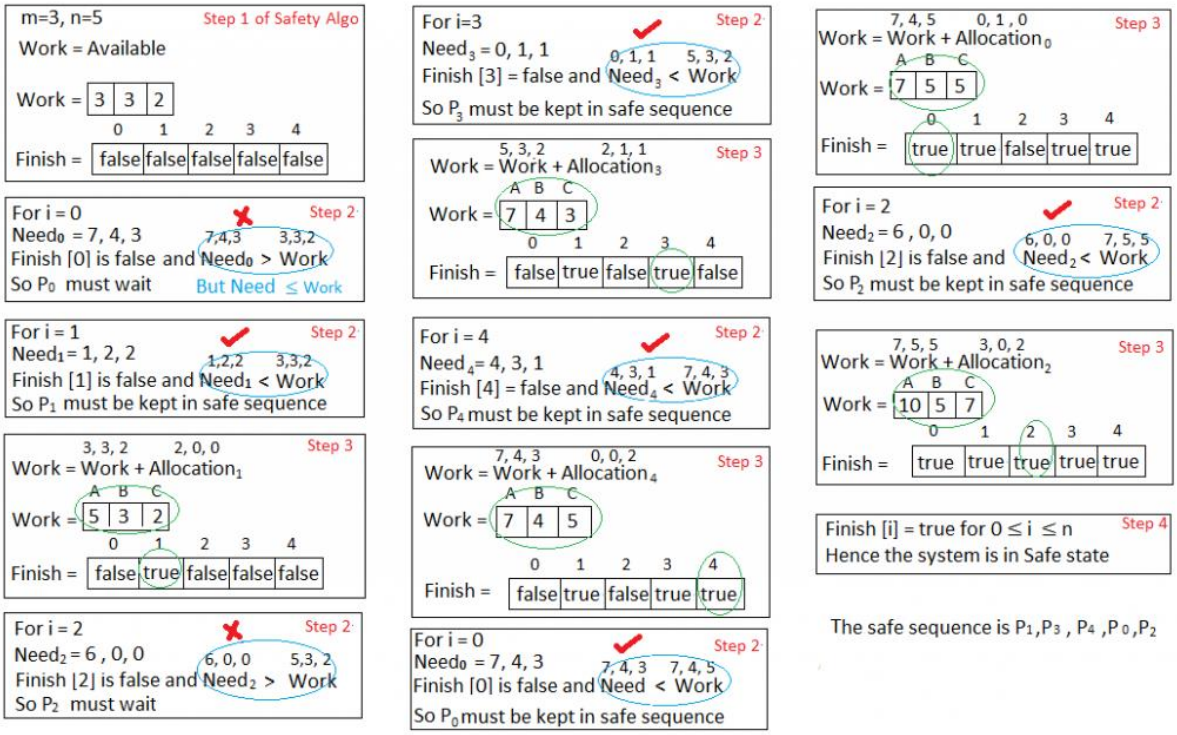
- Bankers' algorithm in OS is a deadlock avoidance algorithm and it is also used to safely allocate the resources to each process within the system.
- It was designed by Edsger Dijkstra.
- As the name suggests, Banker's algorithm in OS is mostly used in the banking systems to identify whether a loan should be given to a person or not.



How does it work?

- Supposing there are 'n' account holders in a particular bank and the total sum of their money is 'x'.
- Now, if a person applies for a loan (let's say for buying a car), then the loan amount subtracted from the total amount available in the bank gives us the remaining amount and that should be greater than 'x', then only the loan will be sanctioned by the bank.
- It is done keeping in mind about the worst case where all the account holders come to withdraw their money from the bank at the same time. Thus, the bank always remains in a safe state.
- Bankers' algorithm in OS works similarly.
- Each process within the system must provide all the important necessary details to the operating system like upcoming processes, requests for the resources, delays, etc. Based on these details, OS decides whether to execute the process or keep it in the waiting state to avoid deadlocks in the system.
- Thus, Banker's algorithm is sometimes also known as the Deadlock Detection Algorithm.

Implementation:



CODE:

```
public class Bankers {


   static int P = 5;
   static int R = 3;

   public static void main(String[] args) {

       int process[] = {0, 1, 2, 3, 4};

       int available_resources[] = {3, 3, 2};

       int maximum_need_resources[][] = {{7, 5, 3},
                                         {3, 2, 2},
                                         {9, 0, 2},
                                         {2, 2, 2},
                                         {4, 3, 3}};

       int alloted_resources[][] = {{0, 1, 0},
                                    {2, 0, 0},
                                    {3, 0, 2},
                                    {2, 1, 1},
```

```
                                    {0, 0, 2}};

        isSafe(process, available_resources, maximum_need_resources,
alloted_resources);
    }

    private static boolean isSafe(int[] process, int[] available_resources,
int[][] maximum_need_resources, int[][] alloted_resources) {

        int[][] need_of_resources = new int[P][R];

        calculateCurrentNeedOfResources(maximum_need_resources,
alloted_resources, need_of_resources);

        boolean[] finished_processes = new boolean[P];

        int[] safeSequence = new int [P];

        int[] current_status_of_resources = new int[R];
        for (int i = 0; i<R; i++){
            current_status_of_resources[i] = available_resources[i];
        }

        int count = 0;
        while (count < P){
            boolean found = false;

            for (int proc = 0; proc<P; proc++){

                if (finished_processes[proc] == false){
                    int j;
                    for (j=0; j<R; j++){
                        if (need_of_resources[proc][j] >
current_status_of_resources[j]){
                            break;  //we dont have sufficient resource
instances
                        }
                    }

                    if (j == R){
                        for (int k = 0; k<R; k++){
                            current_status_of_resources[k] +=
alloted_resources[proc][k];
                        }
```

```
                        safeSequence[count++] = proc;
                        finished_processes[proc] = true;
                        found = true;
                    }
                }
            }

            if (found == false){
                System.out.println("System encontering DEADLOCK!!");
                return false;
            }
        }

        System.out.println("All the processes can be executed with no
DEADLOCK!");
        System.out.print("Sequence is -> ");

        for (int processes = 0; processes<P; processes++){
            System.out.print("P"+safeSequence[processes]+"\t");
        }

        return true;
    }

    private static void calculateCurrentNeedOfResources(int[][]
maximum_need_resources, int[][] alloted_resources, int[][] need_of_resources)
{

        for (int i = 0; i<P; i++){
            for (int j = 0; j<R; j++){
                need_of_resources[i][j] = maximum_need_resources[i][j] -
alloted_resources[i][j];
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\Jadhav>  & 'C:\Users\Jadhav\Java\jdk-16\bin\java.exe' '-XX:+Sh
ta\Local\Temp\vscodesws_c370f\jdt_ws\jdt.ls-java-project\bin' 'Bankers'
All the processes can be executed with no DEADLOCK!
Sequence is -> P1       P3      P4      P0      P2
PS C:\Users\Jadhav>
```

CONCLUSION:

- Advantages
    - This algorithm helps in detecting and avoiding deadlock and also, helps in managing and controlling process requests of each type of resource within the system.
    - Each process should provide information to the operating system about upcoming resource requests, the number of resources, delays, and about how long the resources will be held by the process before release. This is also one of the main characteristics of the Bankers algorithm.
    - Various types of resources are maintained by the system while using this algorithm, that can fulfil the needs of at least one process type.
    - This algorithm also consists of two other advanced algorithms for maximum resource allocation.
- Disadvantages
    - This algorithm helps in detecting and avoiding deadlock and also, helps in managing and controlling process requests of each type of resource within the system.
    - Each process should provide information to the operating system about upcoming resource requests, the number of resources, delays, and about how long the resources will be held by the process before release. This is also one of the main characteristics of the Bankers algorithm.
    - Various types of resources are maintained by the system while using this algorithm, that can fulfil the needs of at least one process type.
    - This algorithm also consists of two other advanced algorithms for maximum resource allocation.
- This algorithm helps in detecting and avoiding deadlock and also, helps in managing and controlling process requests of each type of resource within the system.
- Each process should provide information to the operating system about upcoming resource requests, the number of resources, delays, and about how long the resources will be held by the process before release. This is also one of the main characteristics of the Bankers algorithm.
- Various types of resources are maintained by the system while using this algorithm, that can fulfil the needs of at least one process type.
- This algorithm also consists of two other advanced algorithms for maximum resource allocation.