Deadlock avoidance Bankers Algorithm

- The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for the 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
- **Why Banker's Algorithm is Named So?**
- The banker's algorithm is named so because it is used in the banking system to check whether a loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S.
- If a person applies for a loan then the bank first subtracts the loan amount from the total money that the bank has and if the remaining amount is greater than S then only the loan is sanctioned.
- It is done because if all the account holders come to withdraw their money then the bank can easily do it.

- It also helps the OS to successfully share the resources between all the processes. It is called the banker's algorithm because bankers need a similar algorithm- they admit loans that collectively exceed the bank's funds and then release each borrower's loan in installments.
- The banker's algorithm uses the notation of a safe allocation state to ensure that granting a resource request cannot lead to a deadlock either immediately or in the future.
  In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in a safe state always.
- The following **Data structures** are used to implement the Banker's Algorithm:
  Let **'n'** be the number of processes in the system and **'m'** be the number of resource types.
- **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 $\mathbf{R}_j$
- **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 $\mathbf{P}_i$ may request at most **'k'** instances of resource type $\mathbf{R}_j$.

- **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 $\mathbf{P_i}$ is currently allocated **‘k’** instances of resource type $\mathbf{R_j}$

- **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 $\mathbf{P_i}$ currently needs **‘k’** instances of resource type $\mathbf{R_j}$

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

  Allocation specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

  Banker’s algorithm consists of a Safety algorithm and a Resource request algorithm.

-

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i$ [j] = k means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

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, P_i 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 – Requesti
$Allocation_i$ = $Allocation_i$ + $Request_i$
$Need_i$ = $Need_i$ – $Request_i$

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. Suppose at time t_0 following snapshot of the system has been taken:

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	

What will be the content of the Need matrix?

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

So, the content of Need Matrix is:

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Q2.Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

Step 1 of Safety Algo

m=3, n=5

Work = Available

Work = 3 | 3 | 2

0	1	2	3	4
false	false	false	false	false

Finish = (above)

Step 2

For i = 0

Need$_0$ = 7, 4, 3

Finish [0] is false and Need$_0$ > Work (7,4,3 > 3,3,2)

So P$_0$ must wait

But Need $\leq$ Work

Step 2

For i = 1

Need$_1$ = 1, 2, 2

Finish [1] is false and Need$_1$ < Work (1,2,2 < 3,3,2)

So P$_1$ must be kept in safe sequence

Step 3

Work = Work + Allocation$_1$ (3, 3, 2 + 2, 0, 0)

A	B	C
5	3	2

Work = (above)

0	1	2	3	4
false	true	false	false	false

Finish = (above)

Step 2

For i = 2

Need$_2$ = 6 , 0, 0

Finish [2] is false and Need$_2$ > Work (6, 0, 0 > 5,3, 2)

So P$_2$ must wait

Step 2

For i=3

Need$_3$ = 0, 1, 1

Finish [3] = false and Need$_3$ < Work (0, 1, 1 < 5, 3, 2)

So P$_3$ must be kept in safe sequence

Step 3

Work = Work + Allocation$_3$ (5, 3, 2 + 2, 1, 1)

A	B	C
7	4	3

Work = (above)

0	1	2	3	4
false	true	false	true	false

Finish = (above)

Step 2

For i = 4

Need$_4$ = 4, 3, 1

Finish [4] = false and Need$_4$ < Work (4, 3, 1 < 7, 4, 3)

So P$_4$ must be kept in safe sequence

Step 3

Work = Work + Allocation$_4$ (7, 4, 3 + 0, 0, 2)

A	B	C
7	4	5

Work = (above)

0	1	2	3	4
false	true	false	true	true

Finish = (above)

Step 2

For i = 0

Need$_0$ = 7, 4, 3

Finish [0] is false and Need < Work (7, 4, 3 < 7, 4, 5)

So P$_0$ must be kept in safe sequence

Step 3

Work = Work + Allocation$_0$ (7, 4, 5 + 0, 1 , 0)

A	B	C
7	5	5

Work = (above)

0	1	2	3	4
true	true	false	true	true

Finish = (above)

Step 2

For i = 2

Need$_2$ = 6 , 0, 0

Finish [2] is false and Need$_2$ < Work (6, 0, 0 < 7, 5, 5)

So P$_2$ must be kept in safe sequence

Step 3

Work = Work + Allocation$_2$ (7, 5, 5 + 3, 0, 2)

A	B	C
10	5	7

Work = (above)

0	1	2	3	4
true	true	true	true	true

Finish = (above)

Step 4

Finish [i] = true for 0 $\leq$ i $\leq$ n

Hence the system is in Safe state

The safe sequence is P$_1$,P$_3$, P$_4$,P$_0$,P$_2$

Q3. **What will happen if process P_1 requests one additional instance of resource type A and two instances of resource type C?**

$$\text{Request}_1 = \overset{A}{1}, \overset{B}{0}, \overset{C}{2}$$

To decide whether the request is granted we use Resource Request algorithm

Step 1

$\underset{\text{Request}_1}{1, 0, 2} < \underset{\text{Need}_1}{1, 2, 2}$

Step 2

$\underset{\text{Request}_1}{1, 0, 2} < \underset{\text{Available}}{3, 3, 2}$

Step 3

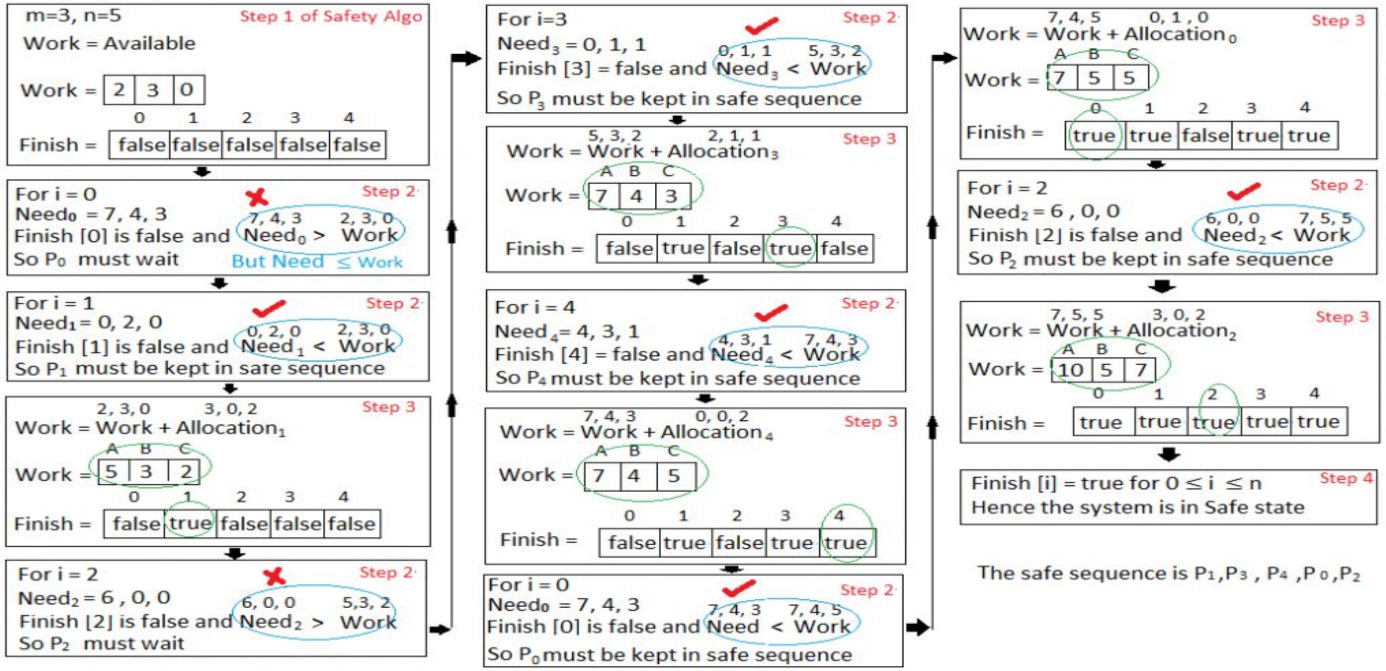
$\text{Available} = \text{Available} - \text{Request}_1$

$\text{Allocation}_1 = \text{Allocation}_1 + \text{Request}_1$

$\text{Need}_1 = \text{Need}_1 - \text{Request}_1$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



- Hence the new system state is safe, so we can immediately grant the request for process $\mathbf{P_1}$.