Banker's Algorithm in Operating System

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.

Why Banker's algorithm is named so?
Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not.
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 safe state always.

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 resources 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 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 P_i may request at most **'k'** instances of resource type 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 P_i is currently allocated **'k'** instances of resource type 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 P_i currently need **'k'** instances of resource type R_j
for its execution.

- Need [i, j] = Max [i, j] – Allocation [i, j]
$Allocation_i$ 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 Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

> 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

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:

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, 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			

Question1. 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
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = | 3 | 3 | 2 |
    0   1   2   3   4

Finish = | false | false | false | false | false |

For i = 0 ✗ Step 2
Need₀ = 7, 4, 3   7,4,3   3,3,2
Finish [0] is false and Need₀ > Work
So P₀ must wait   But Need ≤ Work

For i = 1 ✔ Step 2
Need₁ = 1, 2, 2   1,2,2   3,3,2
Finish [1] is false and Need₁ < Work
So P₁ must be kept in safe sequence

    3, 3, 2   2, 0, 0   Step 3
Work = Work + Allocation₁
    A B C
Work = | 5 | 3 | 2 |
    0   1   2   3   4

Finish = | false | true | false | false | false |

For i = 2 ✗ Step 2
Need₂ = 6 , 0, 0   6, 0, 0   5,3, 2
Finish [2] is false and Need₂ > Work
So P₂ must wait

For i=3 ✔ Step 2
Need₃ = 0, 1, 1   0, 1, 1   5, 3, 2
Finish [3] = false and Need₃ < Work
So P₃ must be kept in safe sequence

    5, 3, 2   2, 1, 1   Step 3
Work = Work + Allocation₃
    A B C
Work = | 7 | 4 | 3 |
    0   1   2   3   4

Finish = | false | true | false | true | false |

For i = 4 ✔ Step 2
Need₄ = 4, 3, 1   4, 3, 1   7, 4, 3
Finish [4] = false and Need₄ < Work
So P₄ must be kept in safe sequence

    7, 4, 3   0, 0, 2   Step 3
Work = Work + Allocation₄
    A B C
Work = | 7 | 4 | 5 |
    0   1   2   3   4

Finish = | false | true | false | true | true |

For i = 0 ✔ Step 2
Need₀ = 7, 4, 3   7, 4, 3   7, 4, 5
Finish [0] is false and Need < Work
So P₀ must be kept in safe sequence

    7, 4, 5   0, 1, 0   Step 3
Work = Work + Allocation₀
    A B C
Work = | 7 | 5 | 5 |
    0   1   2   3   4

Finish = | true | true | false | true | true |

For i = 2 ✔ Step 2
Need₂ = 6 , 0, 0   6, 0, 0   7, 5, 5
Finish [2] is false and Need₂ < Work
So P₂ must be kept in safe sequence

    7, 5, 5   3, 0, 2   Step 3
Work = Work + Allocation₂
    A B C
Work = | 10 | 5 | 7 |
    0   1   2   3   4

Finish = | true | true | true | true | true |

Finish [i] = true for 0 ≤ i ≤ n   Step 4
Hence the system is in Safe state

The safe sequence is P1, P3 , P4 , P0, P2

Question3. What will happen if process P₁ requests one additional instance of resource type A and two instances of resource type C?

A B C
Request$_1$ = 1, 0, 2

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

Step 1
1, 0, 2   1, 2, 2 ✔
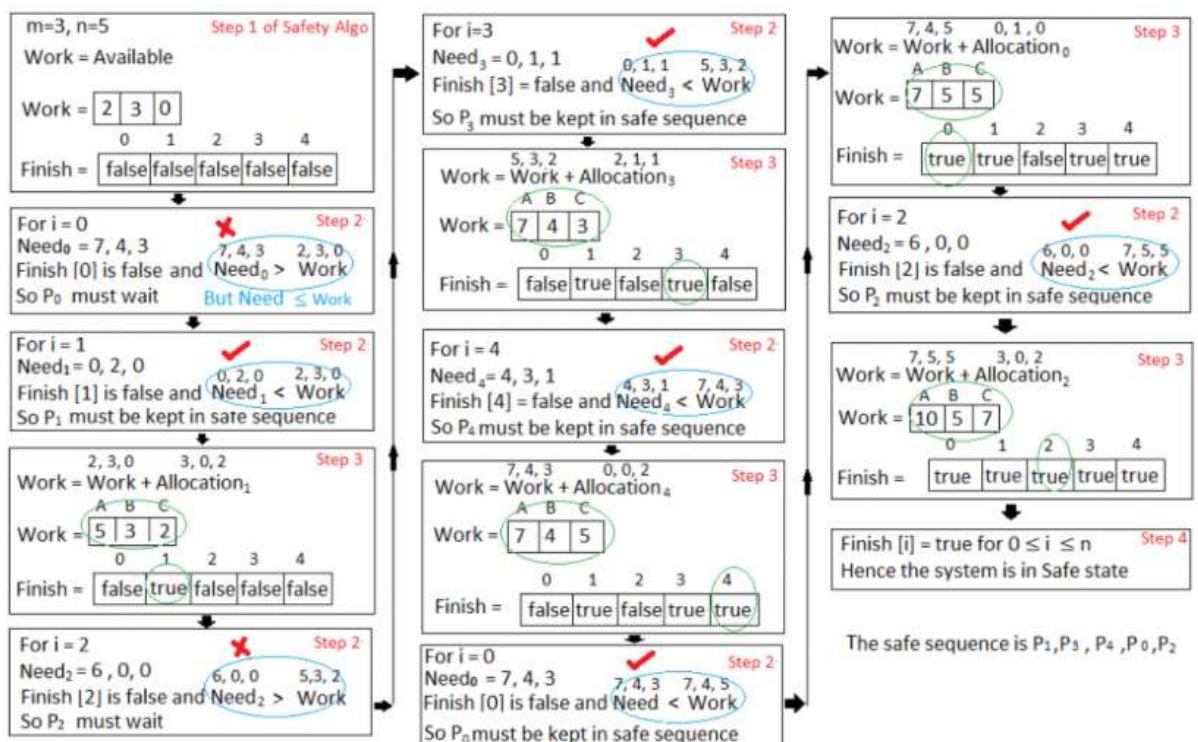Request$_1$ < Need$_1$

Step 2
1, 0, 2   3, 3, 2 ✔
Request$_1$ < Available

Step 3
Available = Available − Request$_1$
Allocation$_1$ = Allocation$_1$ + Request$_1$
Need$_1$ = Need$_1$ − 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.

Step 1 of Safety Algo
m=3, n=5
Work = Available
Work = 2 3 0
        0 1 2 3 4
Finish = false false false false false

For i = 0 ✘ **Step 2**
Need$_0$ = 7, 4, 3   7, 4, 3   2, 3, 0
Finish [0] is false and Need$_0$ > Work
So P$_0$ must wait   But Need ≤ Work

For i = 1 ✔ **Step 2**
Need$_1$ = 0, 2, 0   0, 2, 0   2, 3, 0
Finish [1] is false and Need$_1$ < Work
So P$_1$ must be kept in safe sequence

Step 3
2, 3, 0   3, 0, 2
Work = Work + Allocation$_1$
        A B C
Work = 5 3 2
        0 1 2 3 4
Finish = false true false false false

For i = 2 ✘ **Step 2**
Need$_2$ = 6, 0, 0   6, 0, 0   5, 3, 2
Finish [2] is false and Need$_2$ > Work
So P$_2$ must wait

For i=3 ✔ **Step 2**
Need$_3$ = 0, 1, 1   0, 1, 1   5, 3, 2
Finish [3] = false and Need$_3$ < Work
So P$_3$ must be kept in safe sequence

Step 3
5, 3, 2   2, 1, 1
Work = Work + Allocation$_3$
        A B C
Work = 7 4 3
        0 1 2 3 4
Finish = false true false true false

For i = 4 ✔ **Step 2**
Need$_4$ = 4, 3, 1   4, 3, 1   7, 4, 3
Finish [4] = false and Need$_4$ < Work
So P$_4$ must be kept in safe sequence

Step 3
7, 4, 3   0, 0, 2
Work = Work + Allocation$_4$
        A B C
Work = 7 4 5
        0 1 2 3 4
Finish = false true false true true

For i = 0 ✔ **Step 2**
Need$_0$ = 7, 4, 3   7, 4, 3   7, 4, 5
Finish [0] is false and Need < Work
So P$_0$ must be kept in safe sequence

Step 3
7, 4, 5   0, 1, 0
Work = Work + Allocation$_0$
        A B C
Work = 7 5 5
        0 1 2 3 4
Finish = true true false true true

For i = 2 ✔ **Step 2**
Need$_2$ = 6, 0, 0   6, 0, 0   7, 5, 5
Finish [2] is false and Need$_2$ < Work
So P$_2$ must be kept in safe sequence

Step 3
7, 5, 5   3, 0, 2
Work = Work + Allocation$_2$
        A B C
Work = 10 5 7
        0 1 2 3 4
Finish = true true true true true

Step 4
Finish [i] = true for 0 ≤ i ≤ n
Hence the system is in Safe state

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

Hence the new system state is safe, so we can immediately grant the request for process P$_1$.