5 processes P_0 through P_4

3 resource types:

A (10 instances), B (5 instances), and C (7 instances).

Snapshot at time T_0:

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3			
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Available		
A	B	C
3	3	2

Data Structures
For Banker's Algorithm

```
n: integer    # of processes
m: integer    # of resource-types
Available[1:m]
 #Available[i] is # of avail resources of type i
Max[1:n,1:m]
 #Max demand of each Pi for each Ri
Allocation[1:n,1:m]
 #current allocation of resource Rj to Pi
Need[1:n,1:m]
 #max # resource Rj that Pi may still request
 #Need[i,j] = Max[i,j] – Allocation[i,j]
```

Banker's Algorithm

```
If request[i] > need[i]
      error (asked for too much)
If request[i] > available[i]
      wait(can't supply it now)
Resources are available to satisfy the request
      Let's assume that we satisfy the request, then
       available = available – request[i]
       allocation[i] = allocation[i] + request[i]
       need[i] = need[i] – request[i]
Now check if this would leave us in a safe state:
   If yes, grant the request
   If no, leave the state as is and cause process to wait
```

Safety Algorithm

```
Initialize:
  Work[1:m] = Available[1:m] //how many resources available
  Finish[1:n] = false      //none of the processes finished yet
Step 1:Find a process i such that both:
      (a) Finish[i] = false
      (b) Need[i] ≤ Work
      If no such i exists, go to step 3.
Step 2: Found an i
      Finish[i] = true      //done with this process
      Work = Work + Allocation[i]
      go to step 1
Step 3:
     If Finish[i] == true for all i, then
        the system is in a safe state.
     Else
   Not safe
```