Banker's algorithm

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.

Example

suppose we have five processes P_0 through P_4 and three resource types A, B, and C. Resource type A has 10 instances, resource type B has 5 instances, and resource type C has 7 instances. Suppose that, at time To, the following snapshot of the system has been taken:

Process	Allocation A B C	Max A B C	Available 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	

In the following code, we first judge if the state is safe. Then suppose P_1 request for [1,0,2], we want to find whether this requestion is granted.

Code

```
from multiprocessing import Process
import copy
import numpy as np

class process:
    def __init__(self, allocation, maxr):
        self.allocation = allocation
        self.maxr = maxr
        self.need = self.maxr- self.allocation

def safety_algorithm(available,processes):
```

```
    work = copy.deepcopy(available)
    finish = np.zeros((len(processes),))
    flag = []
    for j in range(len(processes)):
        for i in range(len(processes)):
            if finish[i] == 0 and (processes[i].need <= work).all():
                work += processes[i].allocation
                finish[i] = 1
                flag.append(i)
    if (finish == 1).all():
        print('safe state')
        print('sequence:', flag)
        return True
    else:
        print('unsafe')
        return False

def resource_request_algorithm(request_i, processes, available,i):
    if (request_i > processes[i].need).all():
        print('error process has exceeded its maximum claim')
        return False
    else:
        if (request_i > available[i]).all():
            print('request must wait')
            return False
        else:
            available -= request_i
            processes[i].allocation += request_i
            processes[i].need -= request_i
            if safety_algorithm(available, processes):
                return True
            else:
                available += request_i
                processes[i].allocation -= request_i
                processes[i].need += request_i
                return False

def bankerAlgorthm():
    available = np.array([3,3,2])
    Max = np.array([[7,5,3],[3,2,2],[9,0,2],[2,2,2],[4,3,3]])
    allocation = np.array([[0,1,0],[2,0,0],[3,0,2],[2,1,1],[0,0,2]])
    processes = []
    for i in range(len(Max)):
        processes.append(process(allocation[i],Max[i]))
    safety_algorithm(available, processes)
    request_i = np.array([1,0,2])
    print('\nProcess request [1,0,2]')
    if resource_request_algorithm(request_i, processes, available, 1):
        print('request granted')
```

```
bankerAlgorthm()
```

Output

```
safe state
sequence: [1, 3, 4, 0, 2]

Process request [1,0,2]
safe state
sequence: [1, 3, 4, 0, 2]
request granted
```

Conclution

From the output, we know that this state is safe now, and the requestion will be granted.