Deadlock Detection

Banker's Algorithm

Code:

```
class Process:

    def __init__(self,id,A_alloc,B_alloc,C_alloc,A_max,B_max,C_max):

         self.id = id

         self.A_alloc = A_alloc
         self.B_alloc = B_alloc
         self.C_alloc = C_alloc

         self.A_max = A_max
         self.B_max = B_max
         self.C_max = C_max

         self.A_need = A_max - A_alloc
         self.B_need = B_max - B_alloc
         self.C_need = C_max - C_alloc


class System :

     def __init__(self,A,B,C):
          self.A = A
          self.B = B
          self.C = C


#input
n = int(input("No of Process:"))
l=[]

for i in range(n):

     print(f"Process {i+1}")
     id=i+1

     a,b,c = map(int,input("Enter the allocated resources (A,B,C):").split())
     A,B,C = map(int,input("Enter the max resources(A,B,C):").split())

     l.append(Process(id,a,b,c,A,B,C))
     print()


a,b,c = map(int,input("Enter the available Resources(A,B,C):").split())
sys = System(a,b,c)



#bankers algo
```

```
final = []

while len(l):

     process = l[0]
     l.remove(process)

     if (process.A_need,process.B_need,process.C_need) <= (sys.A,sys.B,sys.C):
          final.append(process)
          sys.A+=process.A_alloc
          sys.B+=process.B_alloc
          sys.C+=process.C_alloc
     else:
          l.append(process)


#Output
print()
print("The sequence of the process are:")
print("--->".join([str(i.id) for i in final]))
```

Output:

```
No of Process:5
Process 1
Enter the allocated resources (A,B,C):0 1 0
Enter the max resources(A,B,C):7 5 3

Process 2
Enter the allocated resources (A,B,C):2 0 0
Enter the max resources(A,B,C):3 2 2

Process 3
Enter the allocated resources (A,B,C):3 0 2
Enter the max resources(A,B,C):9 0 2

Process 4
Enter the allocated resources (A,B,C):2 1 1
Enter the max resources(A,B,C):2 2 2

Process 5
Enter the allocated resources (A,B,C):0 0 2
Enter the max resources(A,B,C):4 3 3

Enter the available Resources(A,B,C):3 3 2

The sequence of the process are:
2--->4--->5--->1--->3
```

Wait for Graph (single instance of resource):

Code:

```
class WaitForGraph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, a, b):
        if a not in self.graph:
            self.graph[a] = set()
        self.graph[a].add(b)

    def detect_deadlock(self):
        visited = set()
        in_progress = set()

        def dfs(node):
            if node in in_progress:
                return True
            if node in visited:
                return False

            visited.add(node)
            in_progress.add(node)

            if node in self.graph:
                for neighbor in self.graph[node]:
                    if dfs(neighbor):
                        return True

            in_progress.remove(node)
            return False

        for node in self.graph.keys():
            if node not in visited:
                if dfs(node):
                    return True

        return False


graph = WaitForGraph()

print('Enter the allocation (u-->v),-1 to stop')

while True:
    k=input()
    if '-1' in k : break

    u,v = k.split()
    graph.add_edge(u,v)

if graph.detect_deadlock():
    print('Deadlock Detected')
else:
    print('No Deadlock')
```

Output:

```
Enter the allocation (u-->v),-1 to stop
p1 r1
r1 p2
p2 r2
r2 p1
-1
Deadlock Detected
```