1. Consider the following resource allocation graph: the following problems:
2. Convert it to the matrix representation (i.e., Allocation, Request and Available).
    1. Allocation Matrix:

	R1	R2	R3	R4	R5
P1	0	0	0	1	0
P2	1	0	0	0	0
P3	0	0	1	0	0
P4	0	1	0	0	0
P5	0	0	0	0	1
Total	1	1	1	1	1

    2. Request Matrix:

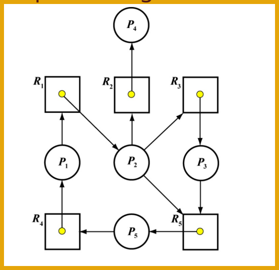
	R1	R2	R3	R4	R5
P1	1	0	0	0	0
P2	0	1	1	0	1
P3	0	0	0	0	1
P4	0	0	0	0	0
P5	0	0	0	1	0

    3. Available Matrix:

R1	R2	R3	R4	R5
0	0	0	0	0

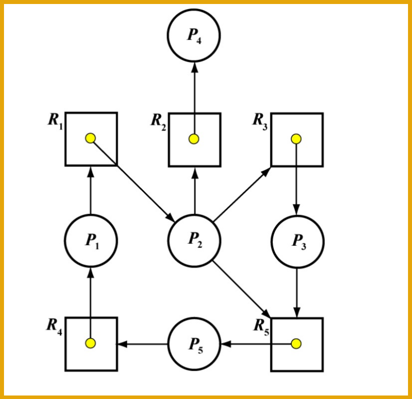
3. Do a step-by-step execution of the deadlock detection algorithm. For each step, add and remove the directed edges, and redraw the resource allocation graph.
4. A deadlock occurs when processes wait resources in a circular manner. Take for example two processes i and j and two resources x and y. If i requests x and j requests y, no issues occur, but if i then requests y and j requests x before each process frees the resources, no process will ever be able to make progress on their tasks, and wait forever for one another. In essence, they have “blocked” each other’s execution. One representation of the allocation of resources can be seen as follows. Take a bipartite graph of sets P (the processes) and R (resources). If a resource is requested by a process but not currently used there is a directed edge from the process to the resource. Similarly, if a resource is currently being used by a process, there is a directed edge from the resource to the process. If a cycle exists and each resource only has a single instance, then a circular wait has occurred. First observe that each resource may only be used by a single process at a time. This means that finding a cycle will suffice to detect a deadlock (i.e. a circular wait). This can be done easily in polynomial time using a depth-first-search (DFS) of the resource allocation graph.

Start with process P1, following the DFS takes process R1, to process P2, to process R5, to process P5 to process R4, back to process P1. Since we have visited a vertex which we already visited while doing DFS, this means we have found a cycle in this graph, in this case indicating a circular wait. Note that this does not require change the resource allocation graph at all. For completeness, I redraw the graph again below.



5. Is there a deadlock? If there is a deadlock, which processes are involved?

    1. Yes, there is a deadlock involving the processes P1, P2, and P5 requesting resources R1, R4, and R5. First note the definition of a deadlock. A deadlock occurs when processes wait resources in a circular manner. Take for example two processes i and j and two resources x and y. If i requests x and j requests y, no issues occur, but if i then requests y and j requests x before each process frees the resources, no process will ever be able to make progress on their tasks, and wait forever for one another. In essence, they have "blocked" each other's execution. One representation of the allocation of resources can be seen as follows. Take a bipartite graph of sets P (the processes) and R (resources). If a resource is requested by a process but not currently used there is a directed edge from the process to the resource. Similarly, if a resource is currently being used by a process, there is a directed edge from the resource to the process. If a cycle exists and each resource can only have a single process using it at a time, then a circular wait has occurred. The resource allocation graph shown below for this problem contains a cycle involving the processes aforementioned.

6. Consider the following resource allocation graph: Do the following:

7. Convert it to the matrix representation (i.e., Allocation, request and Available).

    1. Allocation Matrix:

	R1	R2	R3	R4
P1	1	0	0	0
P2	0	1	0	0
P3	0	0	1	0

P4	0	1	0	1
P5	0	0	0	1
Total	1	2	1	1

   2. Request Matrix:

	R1	R2	R3	R4
P1	0	0	0	0
P2	0	1	1	0
P3	0	0	0	0
P4	1	0	0	0
P5	0	0	0	1

   3. Available Matrix:

R1	R2	R3	R4
2	0	0	0

8. Do a step-by-step execution of the deadlock detection algorithm. For each step, add and remove the directed edges, and redraw the resource allocation graph.

   1. Initial Step

```
flowchart TB
subgraph Process
P1;
P2;
P3;
P4;
P5;
end
subgraph Resource
R1;
R2;
R3;
```

```
R4;
end
R1 --> P1
P1 --> R2
R2 --> P2
R3 --> P3
R4 --> P4
R4 --> P5
P4 --> R1
R2 --> P4
P2 --> R3
P3 --> R4
```

First we can run Process 5 with our current resources

```
flowchart TB
subgraph Process
P1;
P2;
P3;
P4;
end
subgraph Resource
R1;
R2;
R3;
R4;
end
R1 --> P1
P1 --> R2
```

```
R2 --> P2

R3 --> P3

R4 --> P4

P4 --> R1

R2 --> P4

P2 --> R3

P3 --> R4
```

Next we can complete process 4, as it is requesting resource 1, which we have 2 free instance of according to our analysis in Part 1 of this question.

```
flowchart TB

subgraph Process

P1;

P2;

P3;

end

subgraph Resource

R1;

R2;

R3;

R4;

end

R1 --> P1

P1 --> R2

R2 --> P2

R3 --> P3

P2 --> R3

P3 --> R4
```

Now we have removed all cycles. We may now execute process 3

```
flowchart TB
subgraph Process
P1;
P2;
end
subgraph Resource
R1;
R2;
R3;
R4;
end
R1 --> P1
P1 --> R2
R2 --> P2
P2 --> R3
```

Next we can complete process 2

```
flowchart TB
subgraph Process
P1;
end
subgraph Resource
R1;
R2;
R3;
R4;
end
```

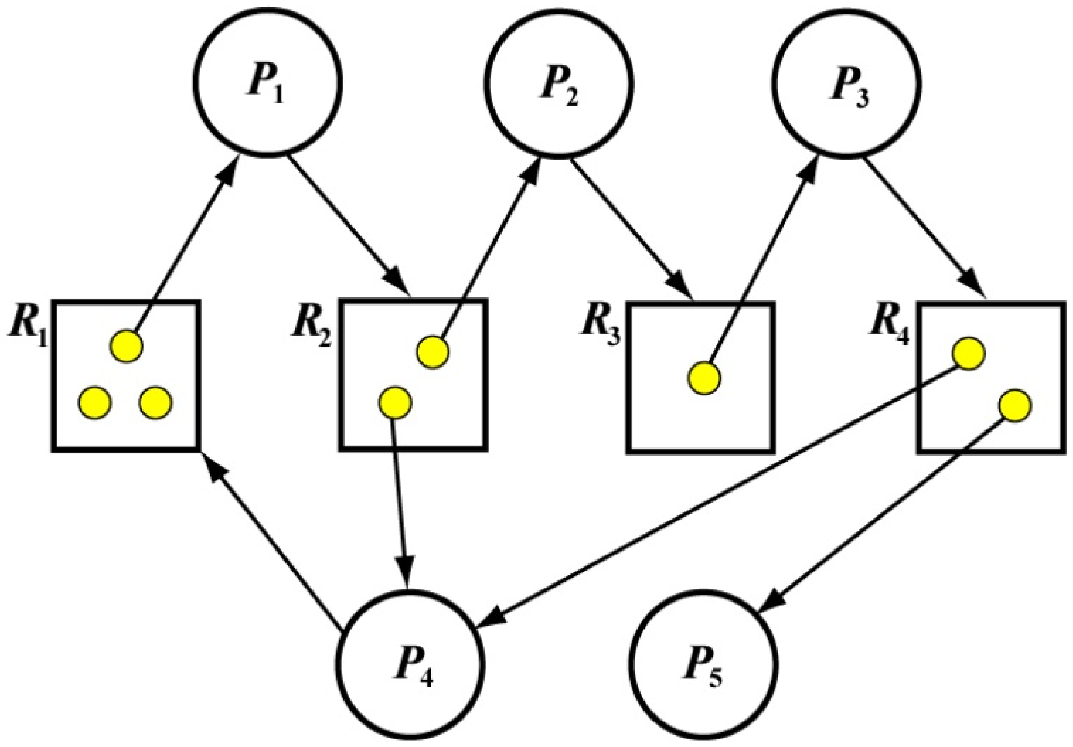
```
R1 --> P1

P1 --> R2
```

Finally we can complete process 1

```
flowchart TB

subgraph Resource

R1;

R2;

R3;

R4;

end
```

All processes have now finished executing and all resources are free

1. Is there a deadlock? If there is a deadlock, which processes are involved?
2. No deadlock occurs. This can be seen by the definition of deadlock. Deadlock is when a circular wait causes processes to not be able to run. In this case however, our modified bankers algorithm shows that the processes can be run to completion because of the multiple instances of resources.


P_1
P_2
P_3
R_1
R_2
R_3
R_4
P_4
P_5