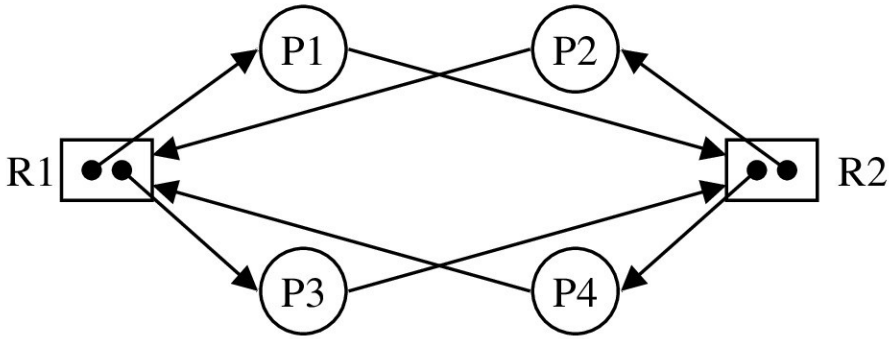
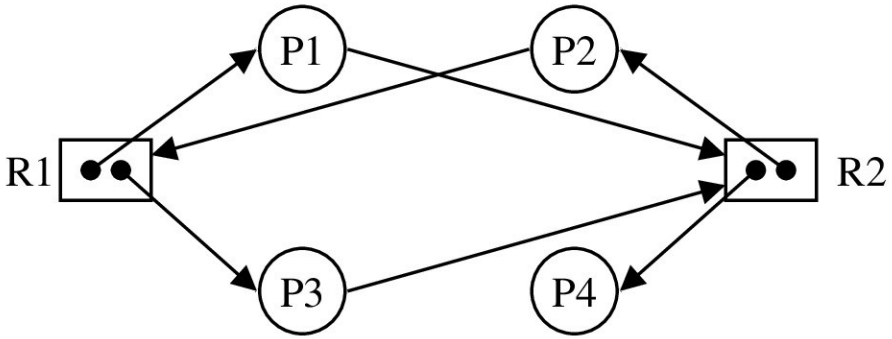
1. Deadlock or not? Justify.



(a)



(b)

(a) Deadlock: The two cycles P1 → R2 → P2 → R1 → P1 and P3 → R2 → P4 → R1 → P3 include **all** the processes, so no process can proceed, and there are no other processes to release any resource.

(b) No deadlock: P4 is not involved in a cycle, and is given the resource R2 that it needs, so P4 can finish releasing one instance of R2. If this instance is given to P1, P1 can finish too, releasing one instance of R1 and one instance of R2. Then, P2 and P3 can finish.

2. [Unsafe does not necessarily mean deadlock]

One possibility: MaxNeed$_{ij}$ are upper bounds. Threads may or may not ask for so many resources.

Assume that each T_i will ask for MaxNeed$_{ij}$ for each j (not necessarily together). Does this mean: Unsafe ⇒ Deadlock?

Suppose that there are two resources A and B with 12 and 1 instances, respectively. At some point of time, we have:

	MAX			ALLOC			NEED			AVAILABLE	
	A	B		A	B		A	B		A	B
P1	10	1	P1	5	0	P1	5	1		3	0
P2	4	1	P2	2	0	P2	2	1			
P3	9	1	P3	2	1	P3	7	0			

This is an unsafe state. However, the processes can still finish without encountering deadlock as follows.

P3 releases B. AVAILABLE = [3 1]
P2 requests [2 1], releases all the resources, and terminates. AVAILABLE = [5 1]
P1 requests [5 1], releases all the resources, and terminates. AVAILABLE = [10 1]
P3 requests [7 0], releases all the resources, and terminates. AVAILABLE = [12 1]

Here, P3 does not require [9 1] together. P1 and P2 hold their respective max needs at certain points of time.

3. There are four resource types A, B, C, D with 3, 14, 12, 12 instances, respectively. There are five processess with the following snapshot at some point of time.

	MaxNeed				Allocation			
	A	B	C	D	A	B	C	D
P1	0	0	1	2	0	0	1	2
P2	1	7	5	0	1	0	0	0
P3	2	3	5	6	1	3	5	4
P4	0	6	5	2	0	6	3	2
P5	0	6	5	6	0	0	1	4

(a) Is the system in a safe state?

We have

Available = [1 5 2 0].

Need =

0	0	0	0
0	7	5	0
1	0	0	2
0	0	2	0
0	6	4	2

A safe sequence at this point is:

P1 Available = [1 5 3 2] P3 Available = [2 8 8 6] P2 Available = [3 8 8 6]

P4 Available = [3 14 11 8] P5 Available = [3 14 12 12]

(b) P2 makes a request (0, 4, 2, 0). Can the request be served immediately? That is, will immediately serving the request leave the system in a safe state?

Granting the request will give the new state:

Available = 1 1 0 0

Need =					Allocation =				
	0	0	0	0		0	0	1	2
	0	3	3	0		1	4	2	0
	1	0	0	2		1	3	5	4
	0	0	2	0		0	6	3	2
	0	6	4	2		0	0	1	4

P1 can finish. Available = [1 1 1 2]
P3 can finish. Available = [2 4 6 6]
P2 can finish. Available = [3 8 8 6]
P4 can finish. Available = [3 14 11 8]
P5 can finish. Available = [3 14 12 12]

Since there is a safe sequence, the request of P2 can be granted.

4. Suppose that an OS uses deadlock avoidance using banker's algorithm. However, the OS restricts the threads to request for a resource of only one type at any point time. Multiple instances of that resource type are allowed in the request. That is, each Request vector has exactly one non-zero entry. In order to reduce deadlock-avoidance overhead, the OS runs banker's algorithm only on the requested resource type. That is, it grants the request if and only if all the threads can finish assuming that the requested resource type is the only resource type available. This reduces the running time from $O(mn^2)$ to $O(n^2)$. Can this shortcut throw a system from a safe state to an unsafe state? Deadlock?

Suppose that there are two resources A and B, each with 12 instances. At some point of time, we have:

	MAX			ALLOC			NEED			AVAILABLE	
	A	B		A	B		A	B		A	B
P1	10	9	P1	5	2	P1	5	7		3	8
P2	4	4	P2	2	2	P2	2	2			
P3	9	10	P3	2	0	P3	7	10			

This is a safe state. A safe sequence is <P2, P1, P3>. Then, the following happens:

P3 asks for [0 5]. By the example of the book (after typo correction), this request does not violate safety with respect to Resource B, and is therefore granted. Now, the state changes to:

	MAX			ALLOC			NEED			AVAILABLE	
	A	B		A	B		A	B		A	B
P1	10	9	P1	5	2	P1	5	7		3	3
P2	4	4	P2	2	2	P2	2	2			
P3	9	10	P3	2	5	P3	7	5			

This is unsafe. Only P2 can finish.