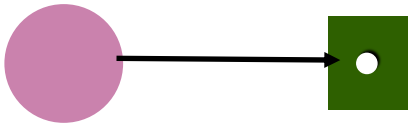
Resource Allocation Graph Algorithm

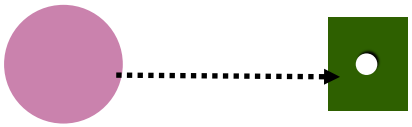
If we have a resource allocation system with only one instance of each resource type, a variant of resource allocation graph can be used for deadlock avoidance. In addition to request and assignment edge, a new edge called a claim edge is also there.

- Request edge



- Assignment edge

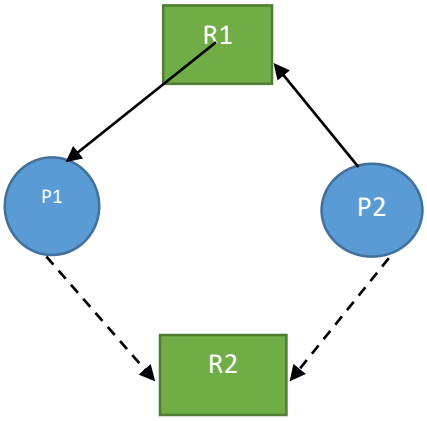
- Claim edge



A claim edge $P_i \rightarrow R_j$ indicates that the process P_i may request resource R_j at some time in the future. This edge resembles a request edge in direction but is represented in the graph by dashed line. When process Pi request resource Rj, the claim edge $P_i \rightarrow R_j$ is converted to a request edge. When the resource Rj is released by Pi, the assignment edge $R_j \rightarrow P_i$ is converted to a claim edge $P_i \rightarrow R_j$. Before process P_i starts executing, all its claim edges must appear in the resource-allocation graph.

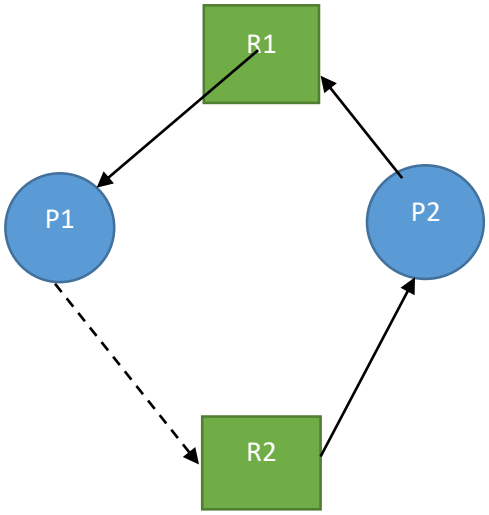
Suppose, process P_i requests resource R_j. The request can be granted only if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle in the resource-allocation graph. Safety is provided by the use of cycle detection algorithm. If no cycle exists, then the allocation of the resources will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state. Therefore, process Pi has to wait for its request to be satisfied.

Consider the following resource allocation graph.



Resource allocation graph for deadlock avoidance

If P2 requests R2, we cannot allocate it to P2 even if R2 is currently free, since this action will create a cycle in the graph. A cycle indicates that the system is in an unsafe state. A cycle indicates that the system is in an unsafe state. If P1 requests R2, and P2 requests R1, then a deadlock will occur.



An unsafe state in a resource allocation graph

Banker's Algorithm

Resource-Allocation Graph Algorithm is not applicable to a resource allocation system with multiple instances of each resource type. Banker's Algorithm deals multiple instances of each resource type. When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

To implement Banker's algorithm, we need the following data structures:

Let ***n*** be the no of processes and ***m*** no of resource type

Available - A vector of length *m* indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type Rj available.

Max - An *n x m* matrix defines the maximum demand of each process. If max[i,j] = k, then process Pi may request at most k instances of resource type Rj

Allocation - An *n x m* matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instances of resource type Rj

Need - An *n x m* matrix indicates the remaining resource need of each process. If Need[i, j] = k, then process Pi may need k more instances of resource type Rj to complete its tasks.

Need[i,j] = Max[i,j] – Allocation[i,j].

We can treat each row in the matrices Allocation and Need as vectors and refer to them as $Allocation_i$ and $Need_i$. The vector $Allocation_i$ specifies the resources currently allocated to process Pi; the vector $Need_i$ specifies the additional resources that process Pi may still request to complete its task.

Safety Algorithm

This algorithm is used to find whether or not a system is in safe state.

1. Let Work and Finish be vectors of length m and n respectively. Initialize

   Work:= Available and Finish[i] := false for i=0,1,2, …, n-1.

2. Find an i such that both
    - Finish[i] = false(Indicates that the corresponding process is notexecuted till now)
    - $Need_i \leq Work$. (If need of the process is <= available, then only the OS can allocate the resource to the process.)

   If no such i exists, go to step 4.
3. Work := Work + $Allocation_i$ (Assume that P0 need is ≤ available resource. So P0 gets resources and completes its execution. Whenever P0 completes its execution, then we can remove the resources from P0 and add them to the available resources)

   Finish[i] := true;

   Go to step 2.
4. If Finish[i] = true for all i, then the system is in a safe state.

Resource-Request Algorithm:

Let $Request_i$ be the request vector for process Pi. If $Request_i[j] = k$, then process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded it maximum claim.
2. If $Request_i \leq$ Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
3. Pretend to have allocated the requested resources to process Pi by modifying the state as follows:
    - Available := Available – $Request_i$;
    - $Allocation_i := Allocation_i + Request_i$;
    - $Need_i := Need_i – Request_i$;

If the resulting resource-allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the new state is unsafe, then Pi must wait for $Request_i$ and the old resource-allocation state is restored.

Example

Consider a system with 5 processes P_0 through P_4 and three resource types *A* (10 instances), *B* (5instances), and *C* (7 instances).

Snapshot at time T_0:

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*ABC*
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

By applying the safety Algorithm, it is proved that the system is in safe state with the sequence <P1,P3,P4,P0,P2>

Suppose now that process P1 request one additional instance of resource type A and two instances of resource type C. So $Request_1$ = (1 0 2) .

Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2))⇒ true.

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we execute our safety algorithm and find that the sequence <P1, P3, P4, P0, P2> satisfies the safety requirement. Hence we can immediately grant the request.