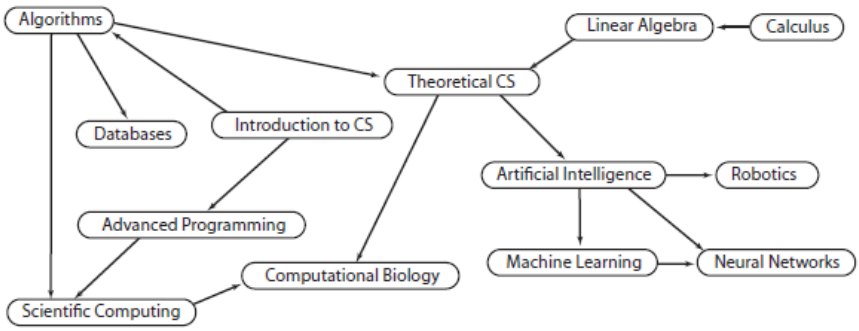
2 Cycles and DAGS

Scheduling problems

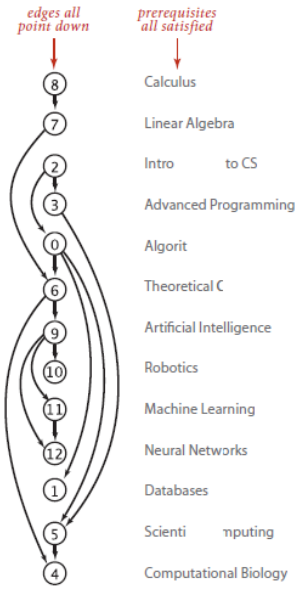
A widely applicable problem-solving model has to do with arranging for the completion of a set of jobs, under a set of constraints, by specifying when and how the jobs are to be performed. The most important type of constraints is precedence constraints, which specify that certain tasks must be performed before certain others.



A precedence-constrained scheduling problem

Precedence-constrained scheduling problem. Given a set of jobs, with precedence constraint, how can schedule jobs such that all are completed? This amount to:

Topological sort. Given a digraph, put the vertices in order such that all its directed edges point from a vertex earlier in the order to a vertex later in the order.



Topological sort

Directed cycle detection problem. Does a given digraph have a directed cycle? If so, find it.

A directed acyclic graph (DAG) is a digraph with no directed cycles.

Algorithm, Directed Cycle Detection

```
:start
global marked[]
global stack // cycle
global onStack[]

dfs(graph, source)
onStack[source] = true
marked[source] = true
for adjV in graph.adj(source)
        if stack!=null
                there is a cycle
        else if !marked[source]
                dfs(graph, adjV)
        else if onStack[w]
                there is a cycle
                create cycle if necessary
        end if
        onStack[source] false
:end
```

Depth-first orders and topological sort

public class	Topological	
	Topological(Digraph G)	*topological-sorting constructor*
boolean	isDAG()	*is G a DAG?*
Iterable<Integer>	order()	*vertices in topological order*

API for topological sorting

Theorem A digraph has a topological order if and only if it is a DAG.

Algorithm Topological Sort

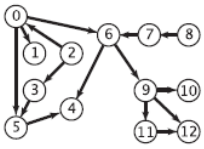
Add to dfs a single line to remember the given vertex in each call.

There are variations to store the visited vertices in dfs:

Preorder : Put the vertex on a queue before the recursive calls.

Postorder : Put the vertex on a queue after the recursive calls.

Reverse postorder : Put the vertex on a stack after the recursive calls.



preorder is order of dfs() calls (queue)

postorder is order in which vertices are done (queue)

(stack)

	pre	post	reversePost
dfs(0)	0		
dfs(5)	0 5		
dfs(4)	0 5 4		
4 done		4	4
5 done		4 5	5 4
dfs(1)	0 5 4 1		
1 done		4 5 1	1 5 4
dfs(6)	0 5 4 1 6		
dfs(9)	0 5 4 1 6 9		
dfs(11)	0 5 4 1 6 9 11		
dfs(12)	0 5 4 1 6 9 11 12		
12 done		4 5 1 12	12 1 5 4
11 done		4 5 1 12 11	11 12 1 5 4
dfs(10)	0 5 4 1 6 9 11 12 10		
10 done		4 5 1 12 11 10	10 11 12 1 5 4
check 12			
9 done		4 5 1 12 11 10 9	9 10 11 12 1 5 4
check 4			
6 done		4 5 1 12 11 10 9 6	6 9 10 11 12 1 5 4
0 done		4 5 1 12 11 10 9 6 0	0 6 9 10 11 12 1 5 4
check 1			
dfs(2)	0 5 4 1 6 9 11 12 10 2		
check 0			
dfs(3)	0 5 4 1 6 9 11 12 10 2 3		
check 5			
3 done		4 5 1 12 11 10 9 6 0 3	3 0 6 9 10 11 12 1 5 4
2 done			
check 3		4 5 1 12 11 10 9 6 0 3 2	2 3 0 6 9 10 11 12 1 5 4
check 4			
check 5			
check 6			
dfs(7)	0 5 4 1 6 9 11 12 10 2 3 7		
check 6			
7 done		4 5 1 12 11 10 9 6 0 3 2 7	7 2 3 0 6 9 10 11 12 1 5 4
dfs(8)	0 5 4 1 6 9 11 12 10 2 3 7 8		
check 7			
8 done		4 5 1 12 11 10 9 6 0 3 2 7 8	8 7 2 3 0 6 9 10 11 12 1 5 4
check 9			
check 10			
check 11			
check 12			

reverse postorder

Computing depth-first orders in a digraph (preorder, postorder, and reverse postorder)