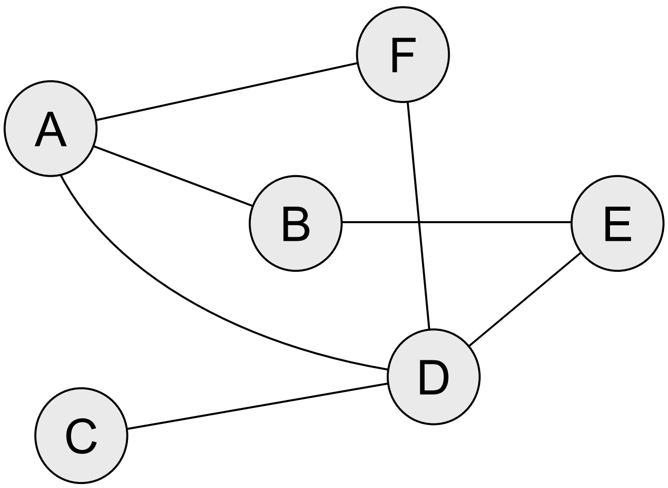
Graphs and Digraphs — Examples

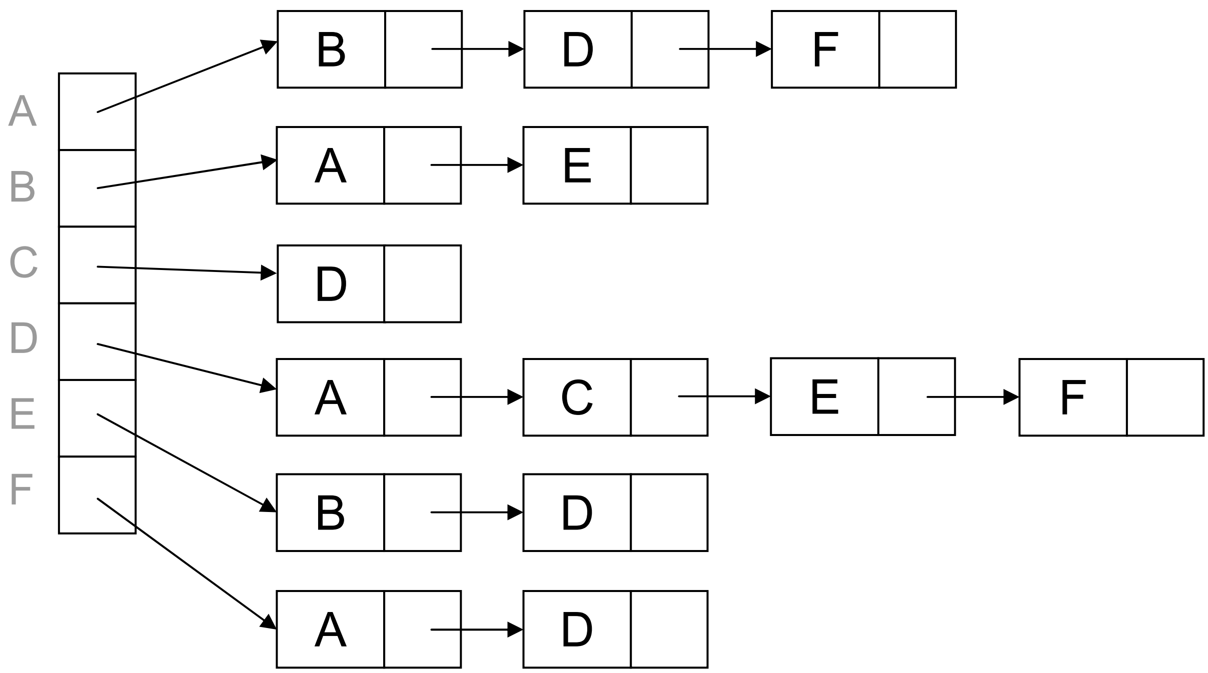
An (undirected) graph *G* = (*V*,*E*)



adjacency matrix for G

	A	B	C	D	E	F
A	0	1	0	1	0	1
B	1	0	0	0	1	0
C	0	0	0	1	0	0
D	1	0	1	0	1	1
E	0	1	0	1	0	0
F	1	0	0	1	0	0

adjacency list for G



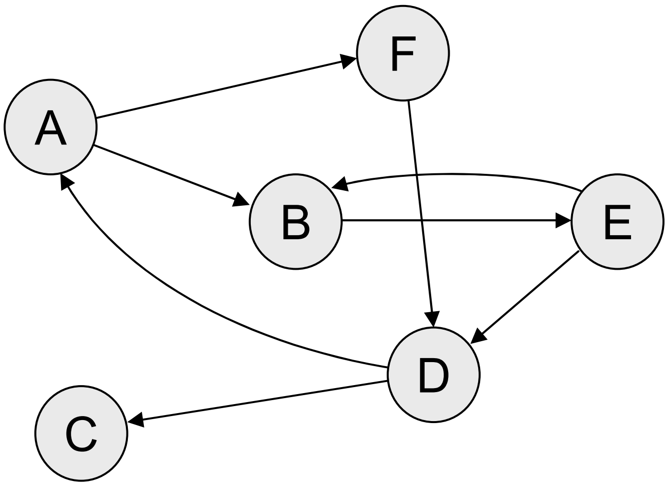
n vertices, e edges $(0 \le e \le n(n-1)/2 \approx n^2/2)$.

Adjacency matrix: $\Theta(n^2)$ space. An algorithm that examines the entire graph structure will require $\Omega(n^2)$ time.

Adjacency list: $\Theta(n+e)$ space. An algorithm that examines the entire graph structure will require $\Omega(n+e)$ time.

Often, $e \ll n^2$. In this case, the adjacency list may be preferable.

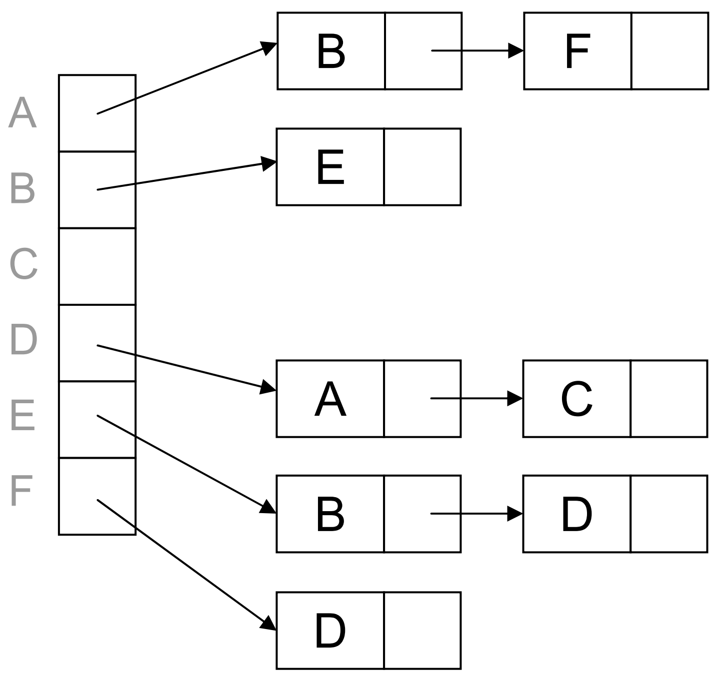
A digraph *G* = (*V*,*E*)



adjacency matrix for G

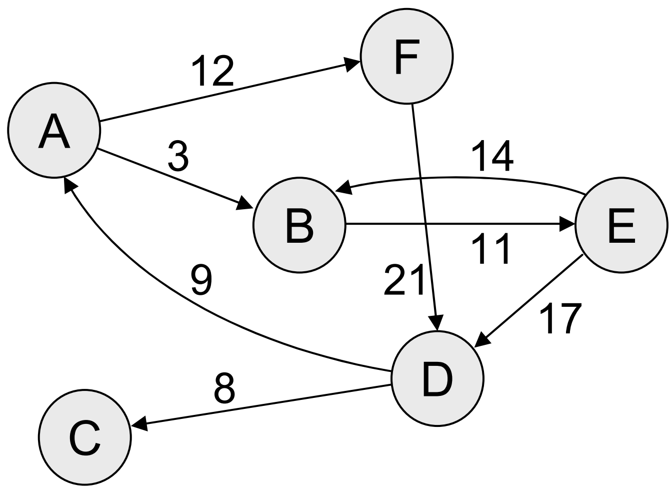
	A	B	C	D	E	F
A	0	1	0	0	0	1
B	0	0	0	0	1	0
C	0	0	0	0	0	0
D	1	0	1	0	0	0
E	0	1	0	1	0	0
F	0	0	0	1	0	0

adjacency list for G



In a digraph, e may be as high as $n(n-1) \approx n^2$, but otherwise the remarks on the previous page hold.

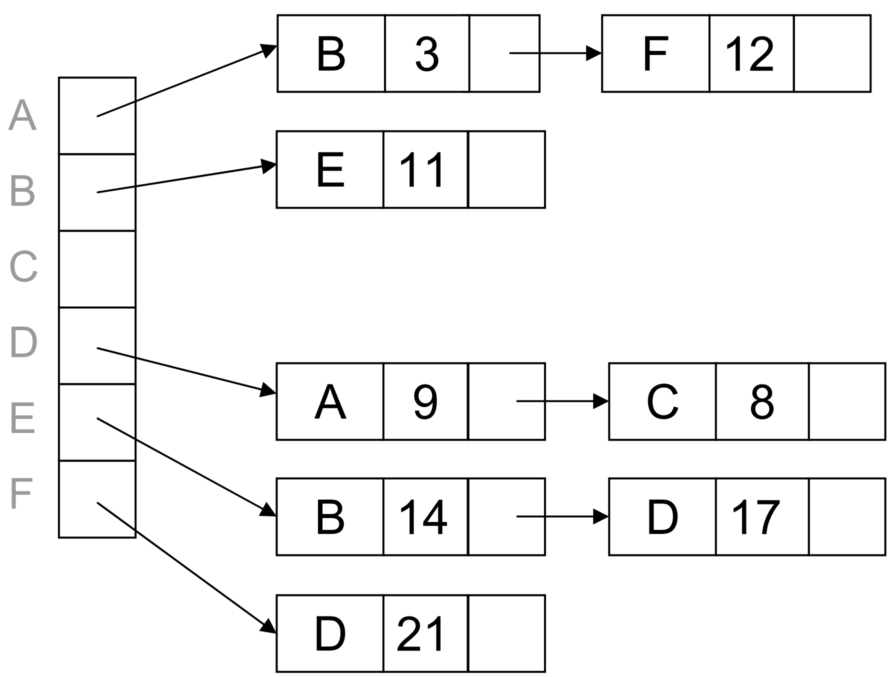
A weighted digraph *G* = (*V*,*E*,*W*)



adjacency matrix for G

	A	B	C	D	E	F
A	∞	3	∞	∞	∞	12
B	∞	∞	∞	∞	11	∞
C	∞	∞	∞	∞	∞	∞
D	9	∞	8	∞	∞	∞
E	∞	14	∞	17	∞	∞
F	∞	∞	∞	21	∞	∞

adjacency list for G



In the adjacency matrix, a non-existent edge might be denoted by 0 or ∞. For example, a non-existent edge could represent

i) a capacity of 0, or

ii) a cost of ∞.

Directed Acyclic Graphs (DAGs)

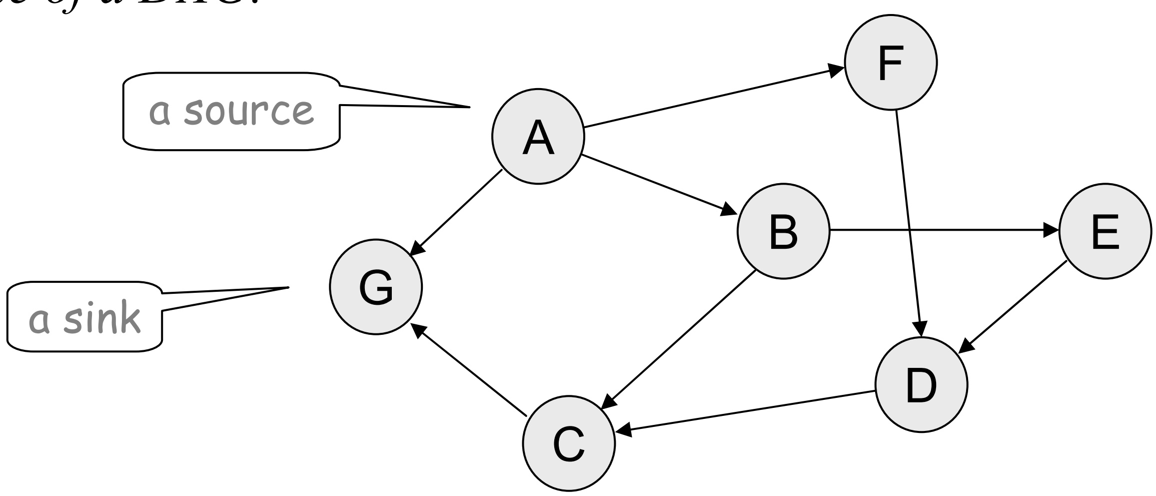
In any *digraph*, we define a vertex v to be

a source, if there are no edges leading into v, and

a sink if there are no edges leading out of v.

A directed acyclic graph (or DAG) is a digraph that has no cycles.

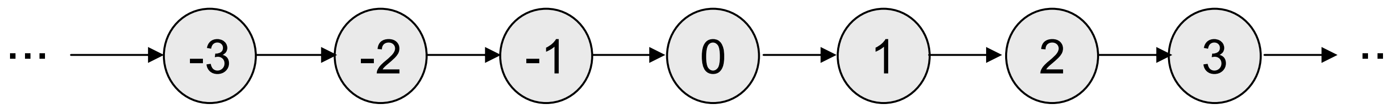
Example of a DAG:



Theorem Every *finite DAG* has at least one source, and at least one sink.

In fact, given any vertex v, there is a path from some source to v, and a path from v to some sink.

Note: This theorem need not hold in an infinite DAG. For example, this DAG has neither a source nor a sink.



Note: In any digraph, the vertices could represent tasks, and the edges could represent constraints on the order in which the tasks be performed.

For example, A must be performed before B, F, or G.
B must be performed before C or E.
C must be performed before G.
D must be performed before C.
E must be performed before D.
F must be performed before D.

We will see that the constraints are consistent if any only if the digraph has no cycles, i.e., is a DAG.

A <u>topological sort</u> of a digraph G = (V,E) is labeling of the vertices by 1, 2, ..., |V| (or by elements of some other ordered set) such that

$$(u,v) \text{ is a edge } \Rightarrow \text{ label}(u) < \text{label}(v).$$

We will see that a digraph has a topological sort if and only if it is a DAG.

For a tasks / constraints graph, a topological sort provides an order in which the tasks can be performed serially, and conversely any valid order for performing the tasks serially gives a topological sort.

Strongly Connected Components of a Digraph

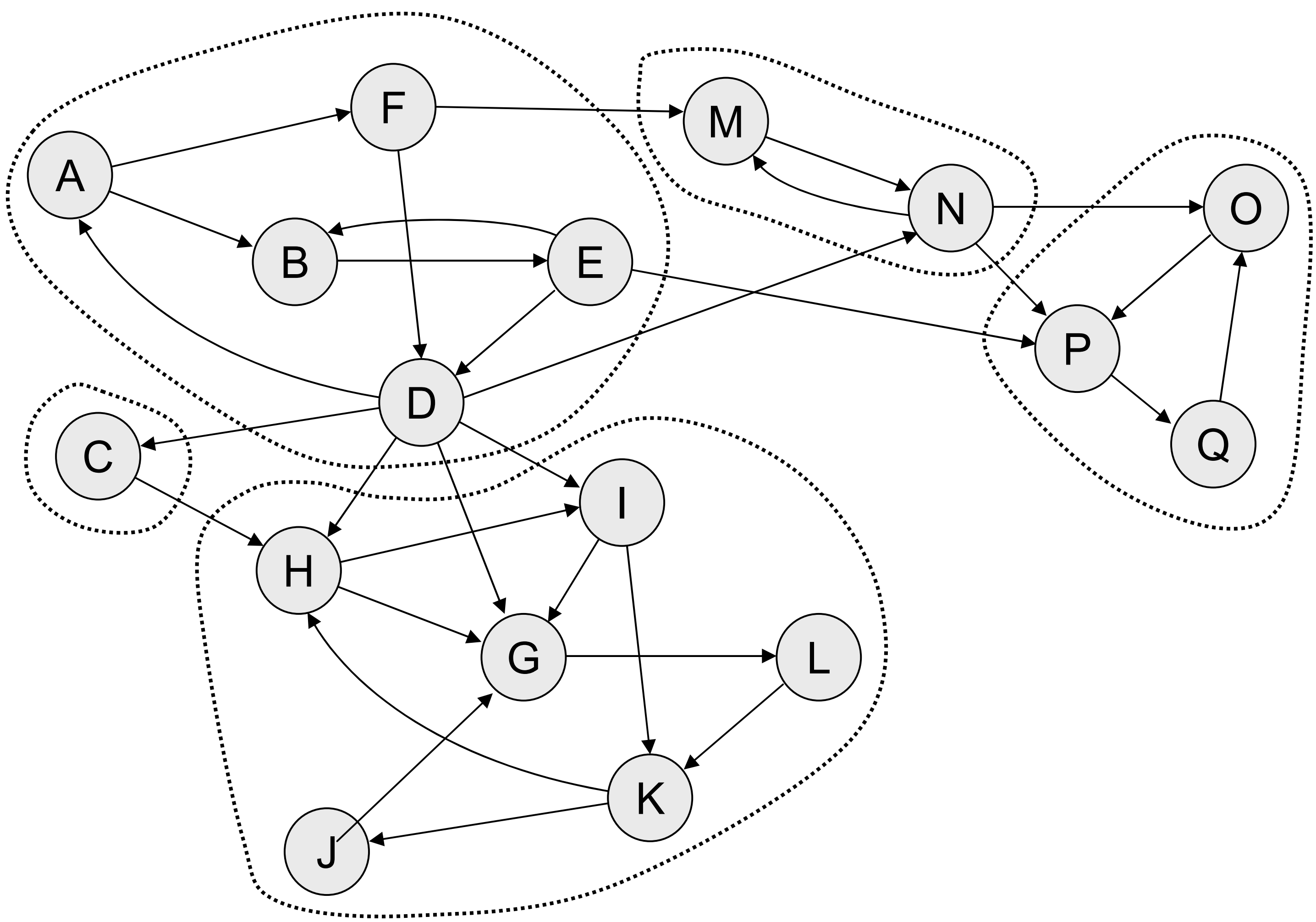
If G is a digraph, define a relation ~ on the vertices by:

$a \sim b$ is there is both a path from a to b, and a path from b to a.

This is an equivalence relation. The equivalence classes are called the strong components of G.

G is strongly connected if it has just one strong component.

This digraph has five strong components.



Given a strongly connected digraph G, we may form the component digraph G^{SCC} as follows:

i) The vertices of G^{SCC} are the strongly connect components of G.

ii) There is an edge from *v* to *w* in G^{SCC} if there is an edge from some vertex of component *v* to some vertex of component *w* in G.

Theorem: The component graph of a digraph is a DAG.

Here is the component digraph for the digraph on the preceding page.