SAMUEL GINN
COLLEGE OF ENGINEERING

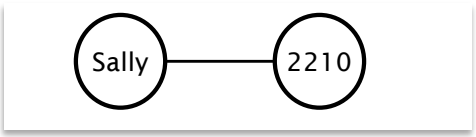
Graphs

Graphs

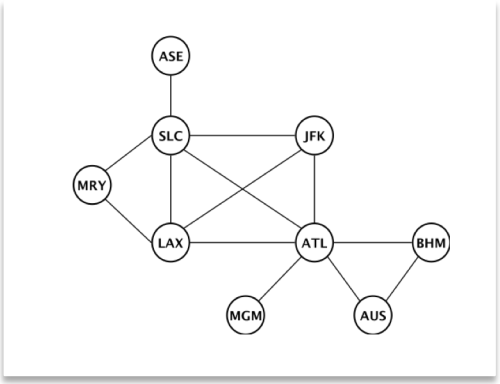
The most general data structure that we've talked about.

Represents pairwise relationships between objects.

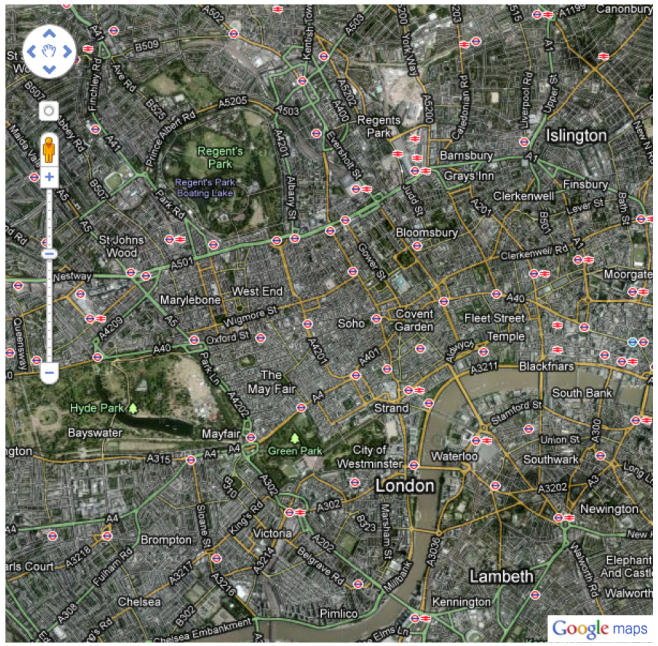
The objects are represented as **vertices** and the relationships are represented as **edges** between the vertices.



A graph is a **model** more than a collection.



Motivating problems

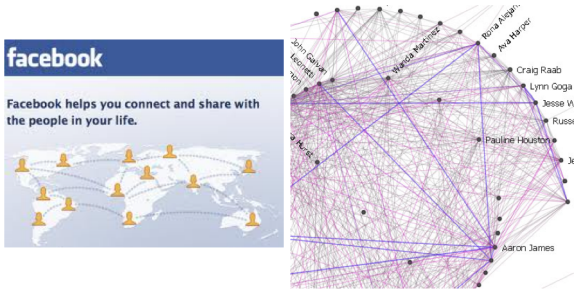


What is the shortest walking route from Westminster Abbey to Hyde Park?

What is the quickest tube route from Trafalgar Square to the Tower of London?

High definition surveillance cameras are to be added at 25 selected locations around the city, to supplement the thousands of cameras that already exist. These new HD cameras are to by physically connected to each other and to a central intelligence command center by fiber optic cables. Engineers have identified all the possible ways of laying the fiber and have cost estimates for each line. What is the cheapest way to connect all the necessary sites?

Motivating problems



Who are the five most active friends of the various people on the terrorist watch list?

Which registered sex offenders have friends younger than 19?

Identify a group that has the strongest connections to Jane Doe.

Curriculum in Software Engineering

Freshman			
Fall	**Hours**	**Spring**	**Hours**
ENGL 1100 English Composition I	3	ENGL 1120 English Composition II	3
World History or Technology & Civilization[1]	3	MATH 1620 Calculus II	4
MATH 1610 Calculus I	4	PHYS 1610 Engineering Physics II	4
PHYS 1600 Engineering Physics I	4	**COMP 1210 Fundamentals of Computing I**	**3**
ENGR 1110 Introduction to Engineering	2		
ENGR 1100 Engineering Orientation	0		
	16		**14**
Sophomore			
Fall	**Hours**	**Spring**	**Hours**
Core Social Science[1]	3	Core Social Science[1]	3
Core Literature[1]	3	MATH 2660 Topics in Linear Algebra	3
Core Fine Arts	3	ELEC 2200 Digital Logic Circuits	3
MATH 2630 Calculus III	4	**COMP 2710 Software Construction**	**3**
COMP 2210 Fundamentals of Computing II	**4**	**COMP 3240 Discrete Structures**	**3**
	17		**15**

What is a legal sequence of courses to take that doesn't violate any prerequisite?

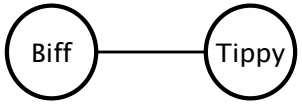
What is the fewest number of semesters required to complete the degree?

How many sets of courses are not related by prerequisites?

Edge characteristics

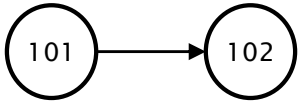
Directed v. Undirected

Undirected edges represent symmetric relationships, and are indicated by a line.



Biff and Tippy are friends.

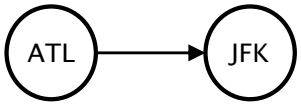
Directed edges represent asymmetric relationships, and are indicated by an arrow.

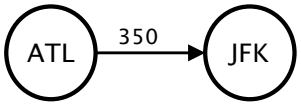


Course 101 is a prerequisite for course 102.

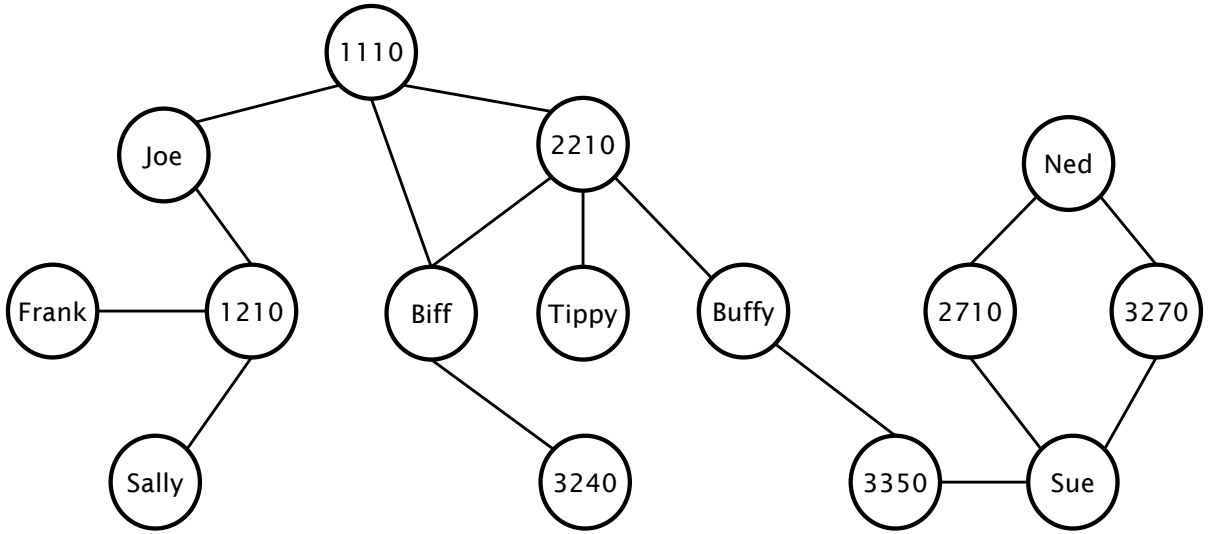
Weighted v. Unweighted

Edges can have numeric values (weights) associated with them or not. These values can represent cost, time, distance, etc. – anything that is relevant to the relationship.

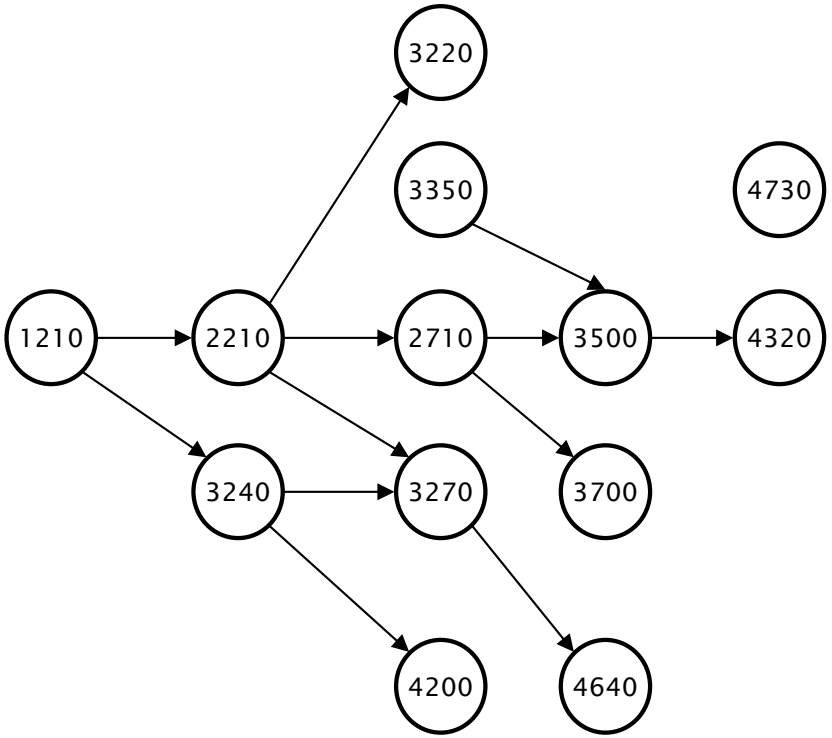




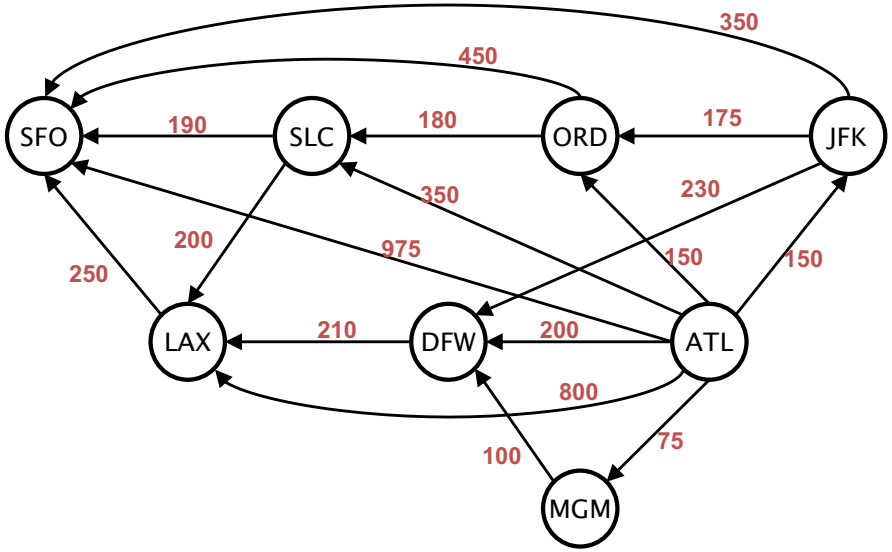
Example graph



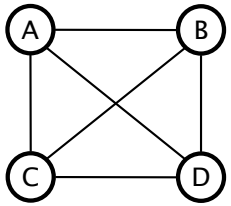
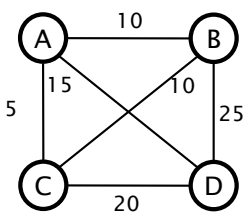
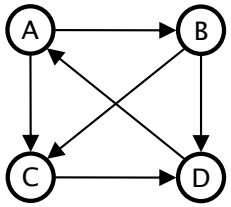
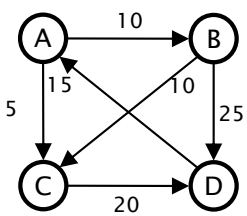
Example graph



Example graph



Four basic graph types

	Unweighted	Weighted
Undirected		
Directed *(digraph)*		

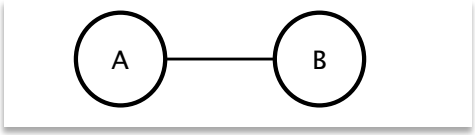
Adjacency and graph representations

Adjacency

Adjacency is the basic connectedness property of vertices.

For undirected graphs

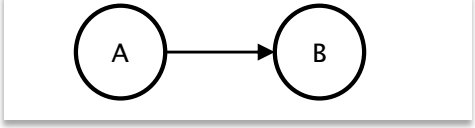
Two vertices are **adjacent** to each other iff there is an edge between them.



A is adjacent to B.
B is adjacent to A.

For directed graphs

Node B is **adjacent** to node A iff there is an edge from A to B.

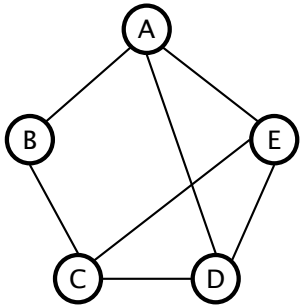


B is adjacent to A.
A is *not* adjacent to B.

Adjacency is the property that is captured in the two primary graph representations.

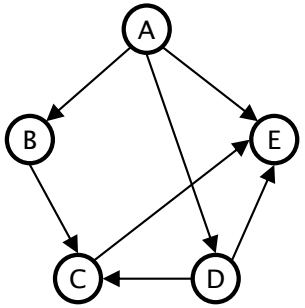
Adjacency matrix

An **adjacency matrix** is a two dimensional table where both the rows and the columns represent the vertices of the graph. Cell *(i, j)* indicates if vertex *j* is adjacent to vertex *i*.



	A	B	C	D	E
A		1		1	1
B	1		1		
C		1		1	1
D	1		1		1
E	1		1	1	

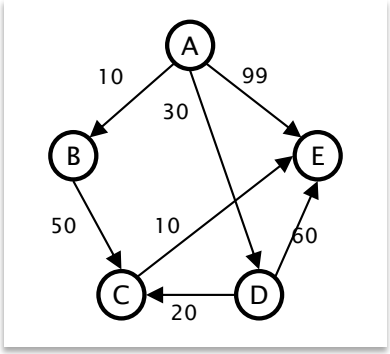
Undirected graphs will always have a symmetric matrix.



	A	B	C	D	E
A		1		1	1
B			1		
C					1
D			1		1
E					

Adjacency matrix

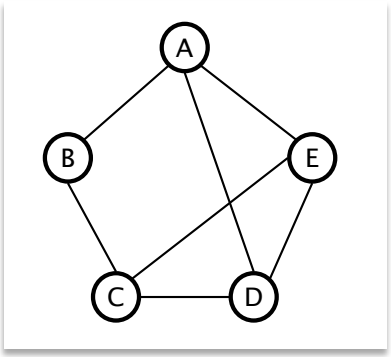
For a weighed graph, cell *(i, j)* indicates if vertex *j* is adjacent to vertex *i* by storing the weight of the edge from vertex *i* to vertex *j*.

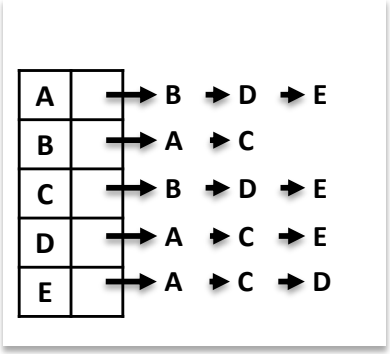


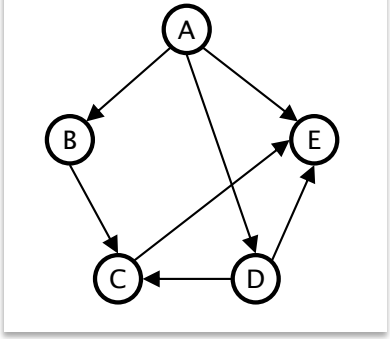
	A	B	C	D	E
A		10		30	99
B			50		
C					10
D			20		60
E					

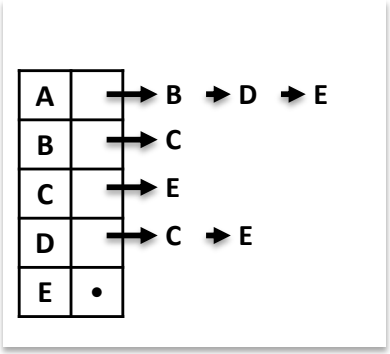
Adjacency list

An **adjacency list** is a one dimensional table where each entry represents the vertices of the graph. Entry k stores a linked list of all the vertices that are adjacent to vertex k.



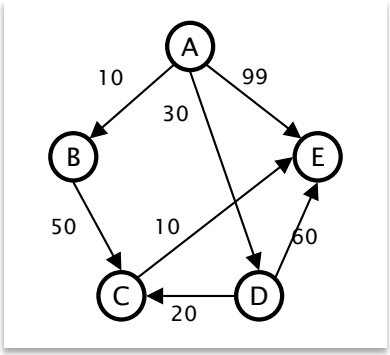


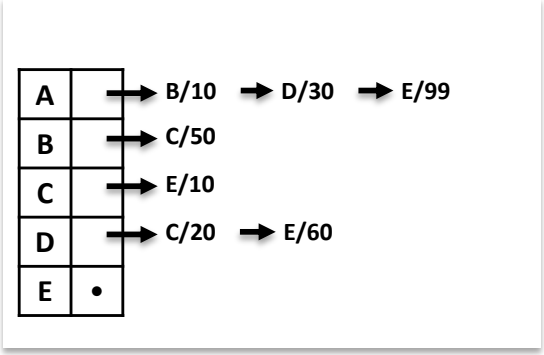




Adjacency list

For a weighted graph, entry k stores a linked list of all the vertices that are adjacent to vertex k. Each node in the linked list stores not only the label of an adjacent vertex m but also the weight on the edge from k to m.



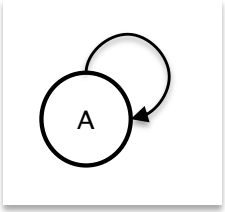


Terminology

Glossary

Vertex *Edge* *Directed edge* *Undirected edge* *Weighted edge*

Unweighted edge *Adjacency*



Self loop – an edge that links a vertex to itself.

Simple graph – a graph with no self loops.

Path – A sequence of vertices/edges from a start vertex to an end vertex.

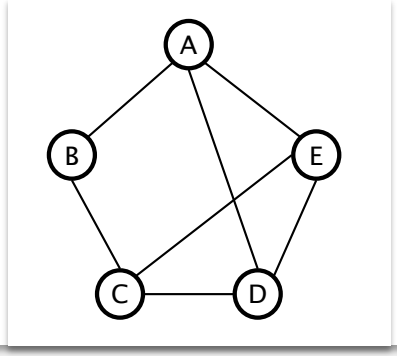
Simple path – A path that does not cross the same edge twice.

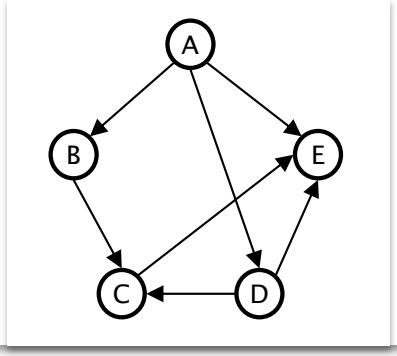
Cycle – A simple path that starts and ends at the same vertex.

Acyclic graph – a graph with no cycles.

Paths
- A-B-C
- D-C-B-A-E
- *A-B-C-B*
- *A-B-C-B-A*

Cycle
- A-E-D-C-B-A





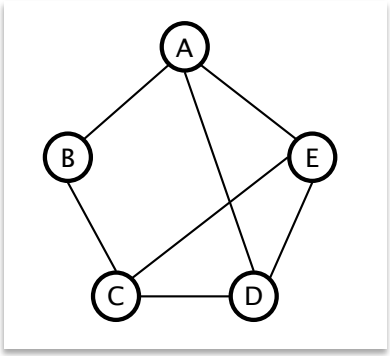
Paths
- A-B-C
- A-B-C-E
- D-C-E

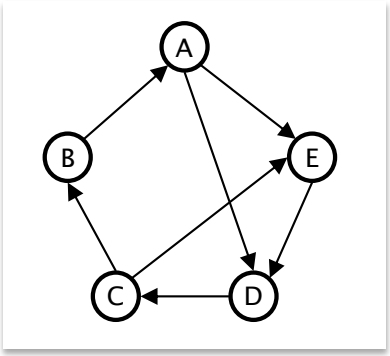
Acyclic

Glossary

Connected Graph – a graph in which there is a simple path between any two pair of vertices.

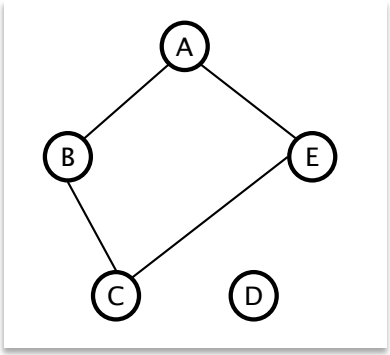
Connected:

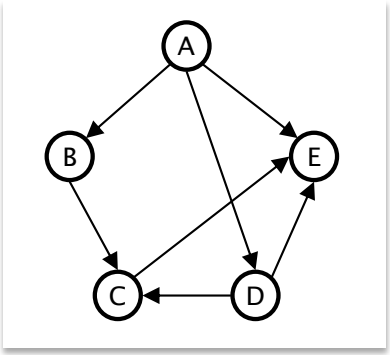




Strongly connected

Not connected:

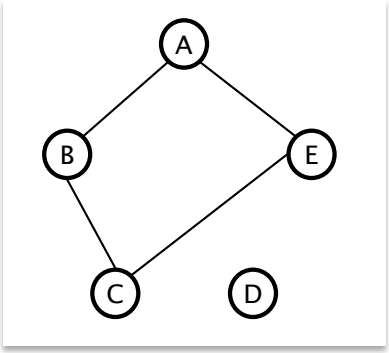




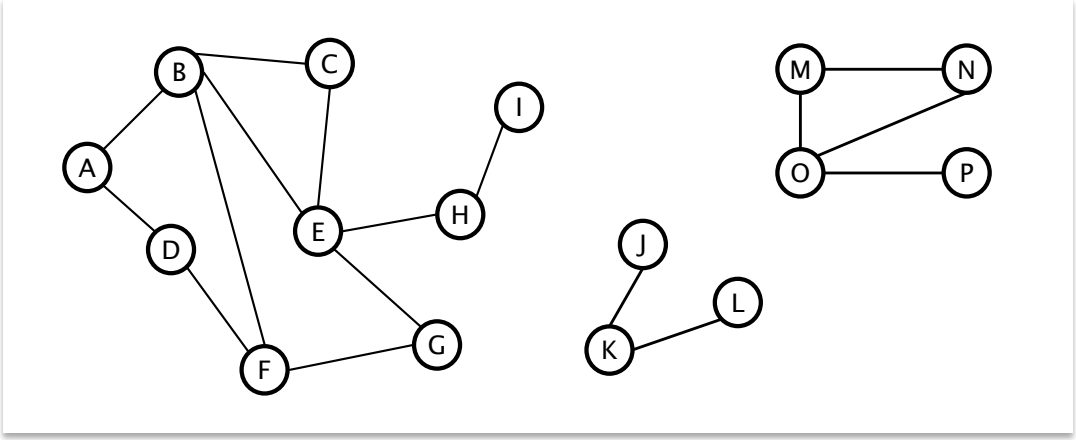
Not strongly connected

Glossary

Connected Component – maximal subgraph that is connected.



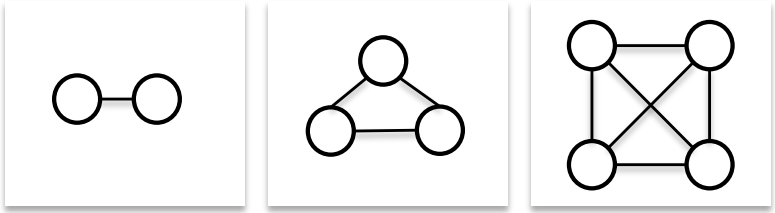
2 connected components

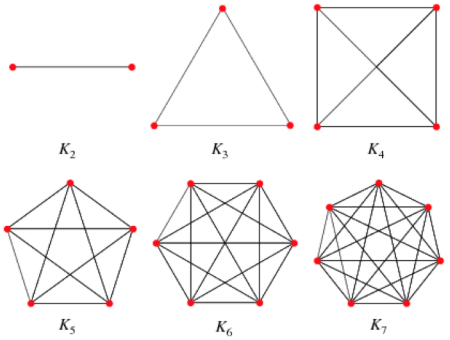


3 connected components

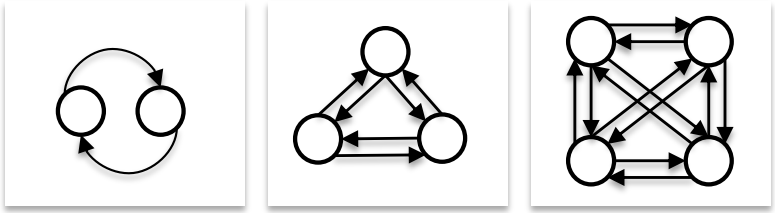
Glossary

Complete Graph – a simple graph in which every pair of vertices is adjacent to each other.





A complete undirected graph with N vertices will have N(N-1)/2 edges.

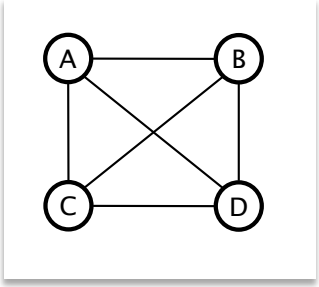


A complete directed graph with N vertices will have N(N-1) edges.

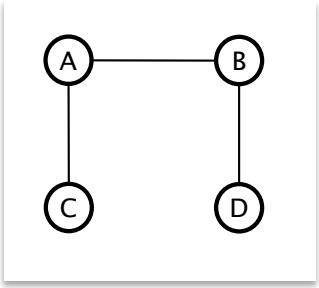
Glossary

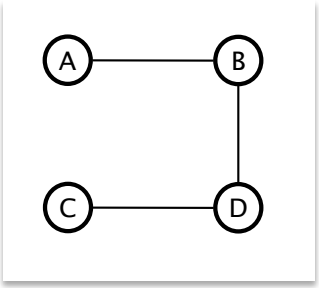
Spanning Tree – A spanning tree of a *connected, undirected* graph is a connected, acyclic subgraph that contains all the vertices of the graph.

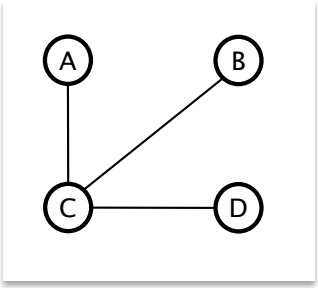
Graph:

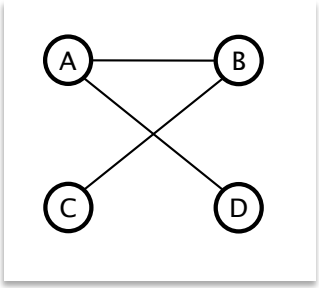


Spanning Trees:

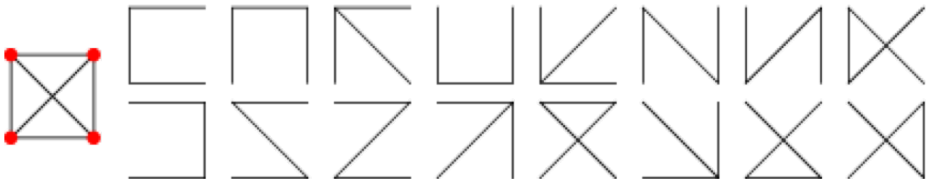








and many more …



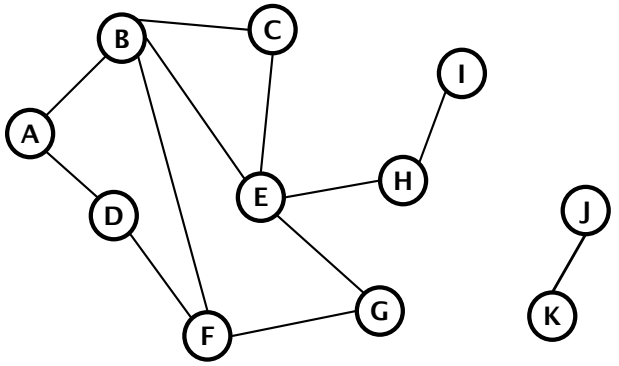
DFS and BFS

Graph traversals: depth-first

Explore the graph by looking for new vertices far away from the start vertex, and examining nearer vertices only when dead ends are encountered.

Has a very simple recursive formulation.

```
dfs(Vertex v)
visit(v)
mark v as visited
for each w adjacent to v {
   if notVisited(w) {
      dfs(w)
   }
}
```



Will visit each vertex that is *reachable* from the start vertex.

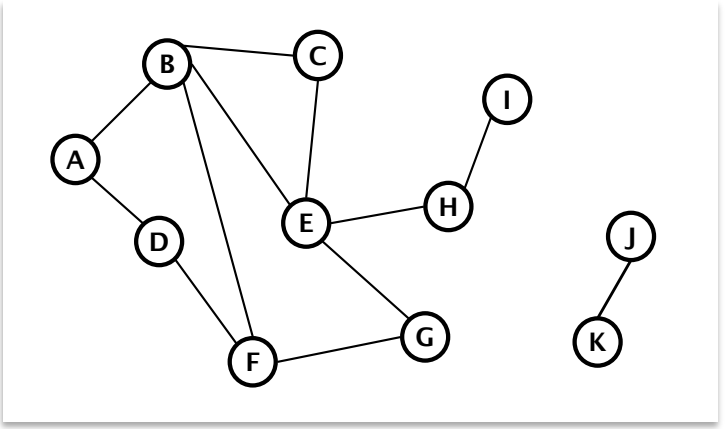
For a graph with V vertices and E edges, DFS can be implemented with O(V + E) time complexity. Note that in a complete graph this would be $O(V^2)$.

Graph traversals: breadth-first

Explore the graph by looking all the vertices closest to the start vertex, and move farther away only when everything nearby has been examined.

Typically implemented iteratively with a FIFO queue.

```
bfs(Vertex v)
visit(v)
mark v as visited
queue.add(v)
while (!queue.isEmpty()) {
   w = queue.remove()
   for each p adjacent to w {
      if notVisited(p) {
         visit(p)
         mark p as visited
         queue.add(p)
      }
    }
}
```



Will visit each vertex that is *reachable* from the start vertex.

For a graph with V vertices and E edges, BFS can be implemented with O(V + E) time complexity. Note that in a complete graph this would be $O(V^2)$.

DFS/BFS: core solutions

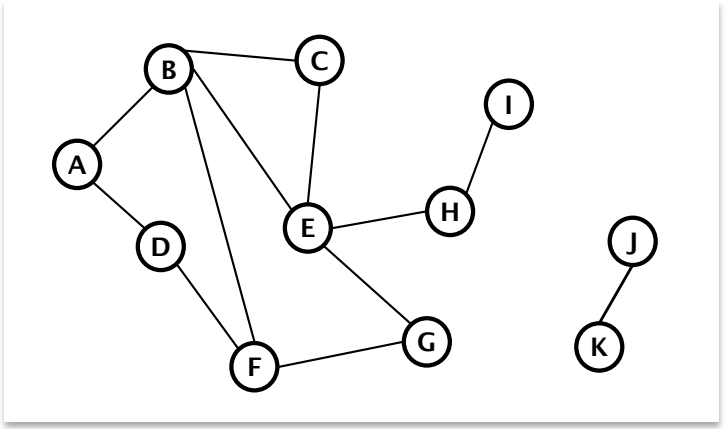
Is there a path from A to H?

What is the shortest path from A to H?

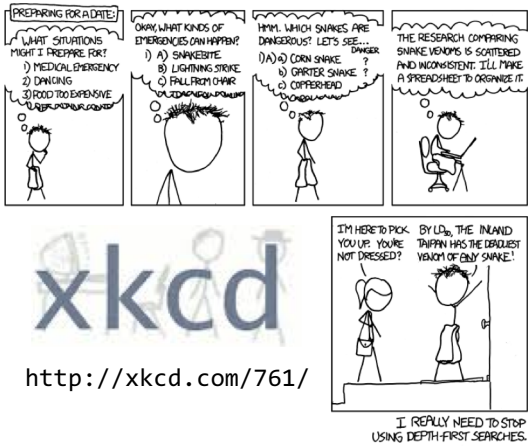
Does the graph have any cycles?

Is the graph connected?

Identify the connected components in the graph.



But sometimes one is more applicable than the other…

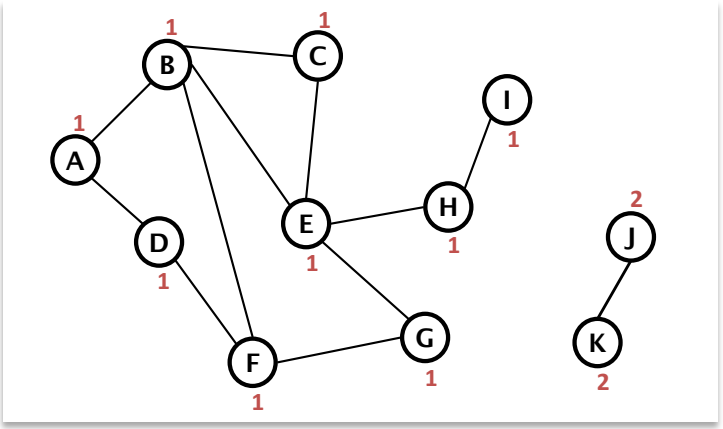


http://xkcd.com/761/

DFS: connected components

```
dfs(Vertex v)
visit(v)
mark v as visited
for each w adjacent to v {
   if notVisited(w) {
      dfs(w)
   }
}
```

Modify DFS to label all vertices in same component as v.



Time complexity: O(V + E)

```
dfsComp(Vertex v, int c)
visit(v)
mark v with c
for each w adjacent to v {
   if notVisited(w) {
      dfsComp(w, c)
   }
}
```

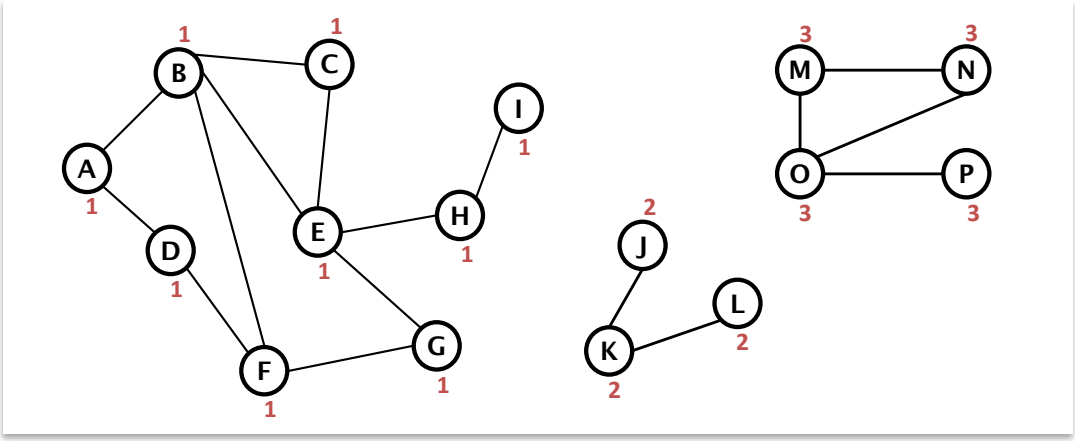
Do this for each vertex.

```
dfsCompDriver()
c = 0
for each vertex v {
   if notVisited(v) {
      c = c + 1
      dfsComp(v, c)
   }
}
```

DFS: connected components

```
dfsCompDriver()
c = 0
for each vertex v {
   if notVisited(v) {
      c = c + 1
      dfsComp(v, c)
   }
}
```

```
dfsComp(Vertex v, int c)
visit(v)
mark v as visited
mark v with c
for each w adjacent to v {
   if notVisited(w) {
      dfsComp(w, c)
   }
}
```



DFS: topological sorting

List the nodes in an order such that for every edge (u, v) u appears before v in the order.

```
dfsTSDriver()
topnum = N // #vertices
for each vertex v {
   if notVisited(v) {
      dfsTopSort(v, topnum)
   }
}
```

```
dfsTopSort(Vertex v, int topnum)
visit(v)
mark v as visited
for each w adjacent to v {
   if notVisited(w) {
      dfsTopSort(w, topnum)
   }
}
mark v with topnum
topnum = topnum - 1
```