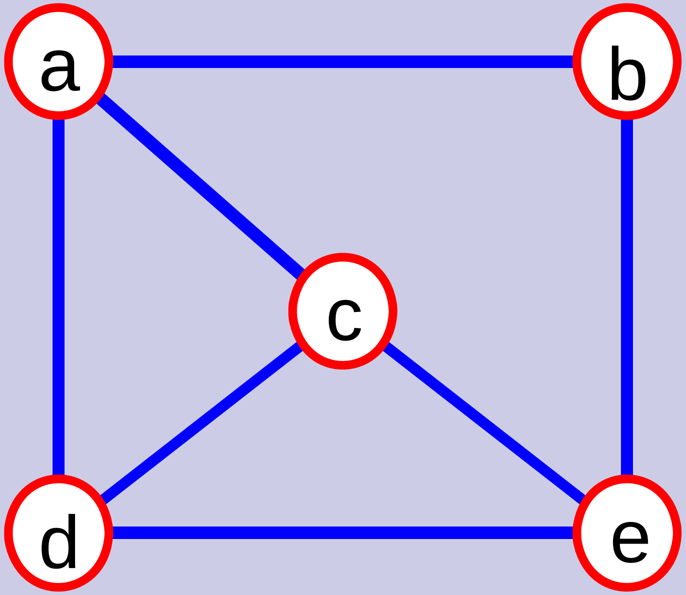
Graphs

- Definitions
- Examples
- The Graph ADT

What is a Graph?

- A graph G = (V,E) is composed of:
  - V: set of vertices
  - E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:

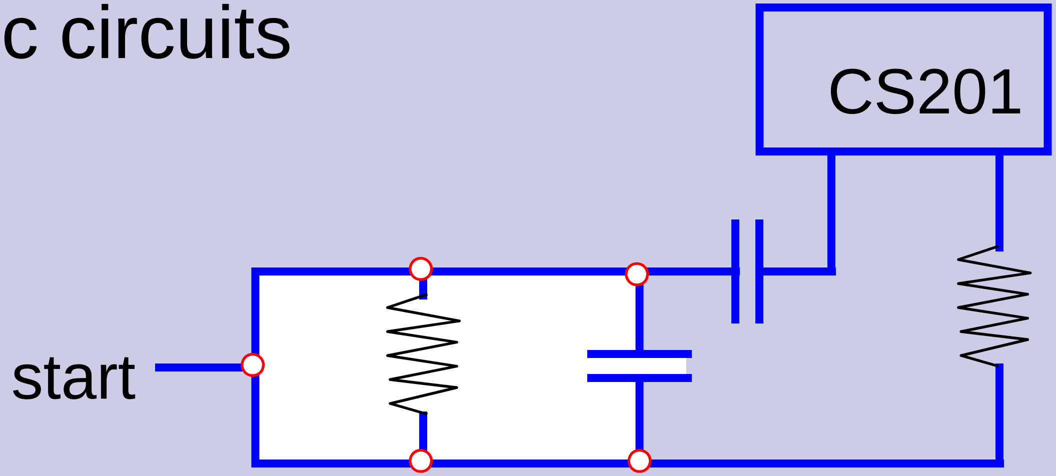


V= {a,b,c,d,e}

E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

Applications

- electronic circuits

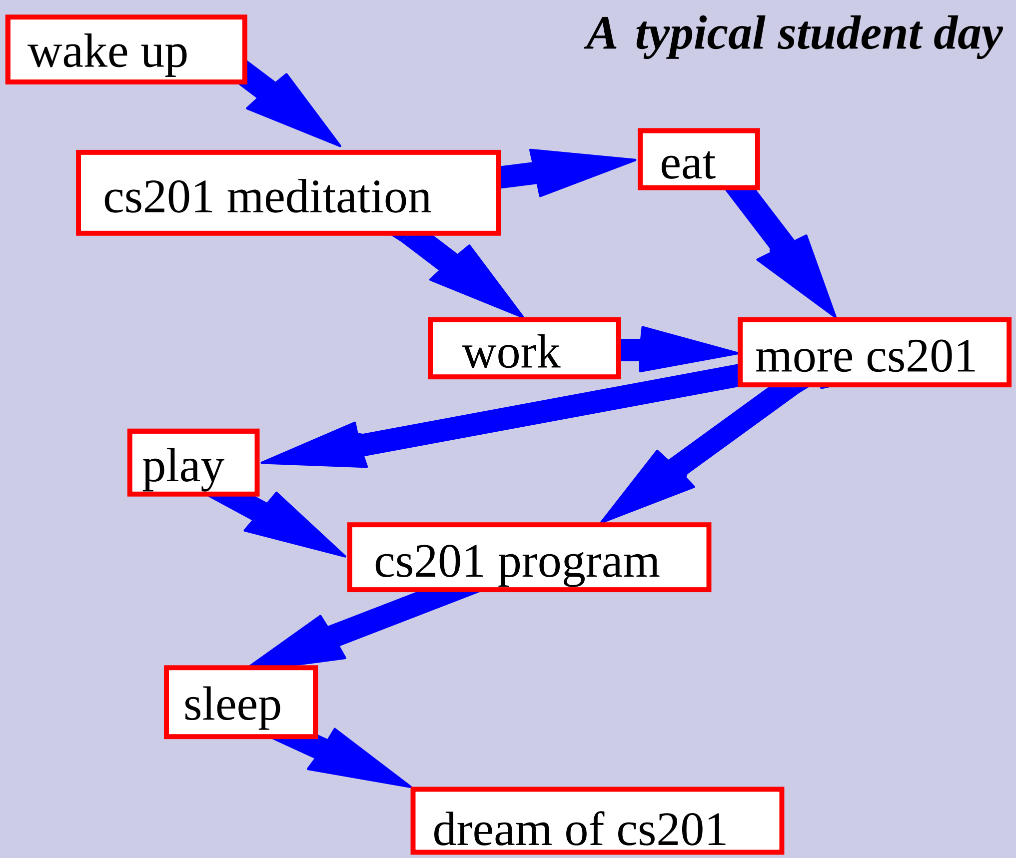


find the path of least resistance to CS201

- networks (roads, flights, communications)

more examples

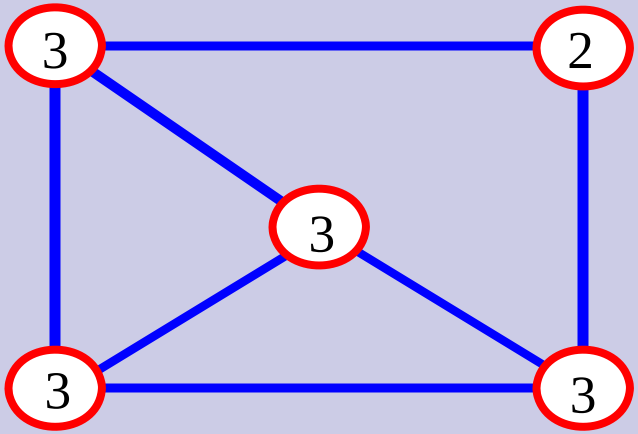
scheduling (project planning)



Graph Terminology

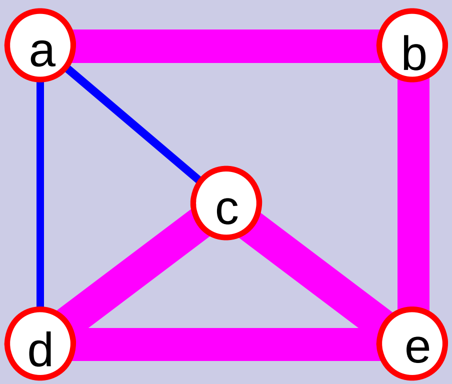
- adjacent vertices: vertices connected by an edge
- degree (of a vertex): # of adjacent vertices

- What is the sum of the degrees of all vertices?
- Twice the number of edges, since adjacent vertices each count the adjoining edge, it will be counted twice

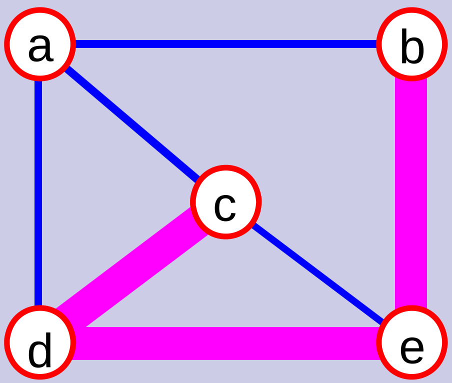


Graph Terminology(2)

- path: sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.



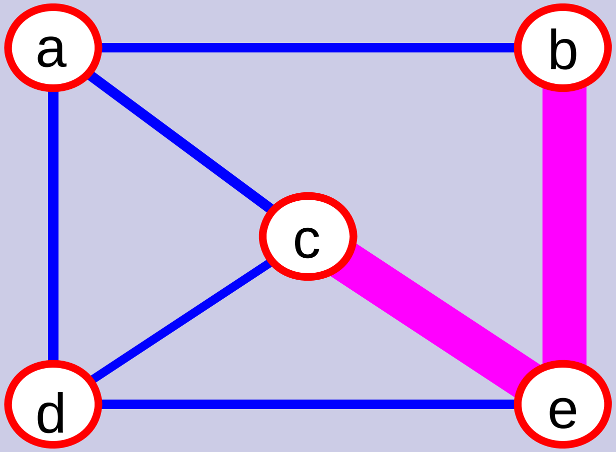
a b e d c e



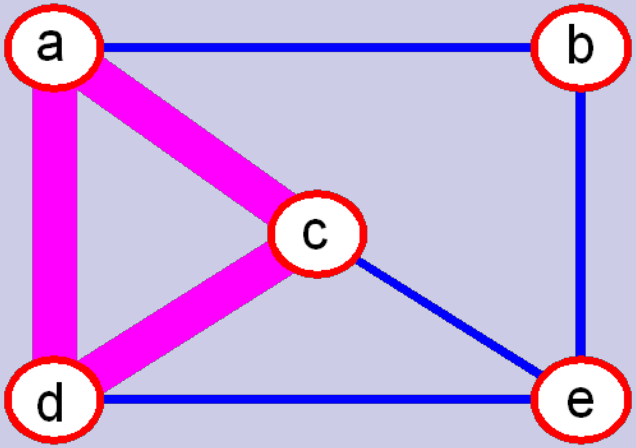
b e d c

Graph Terminology (3)

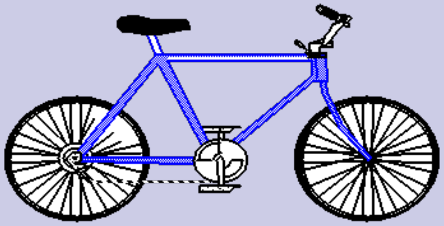
- simple path: no repeated vertices
- cycle: simple path, except that the last vertex is the same as the first vertex



b e c

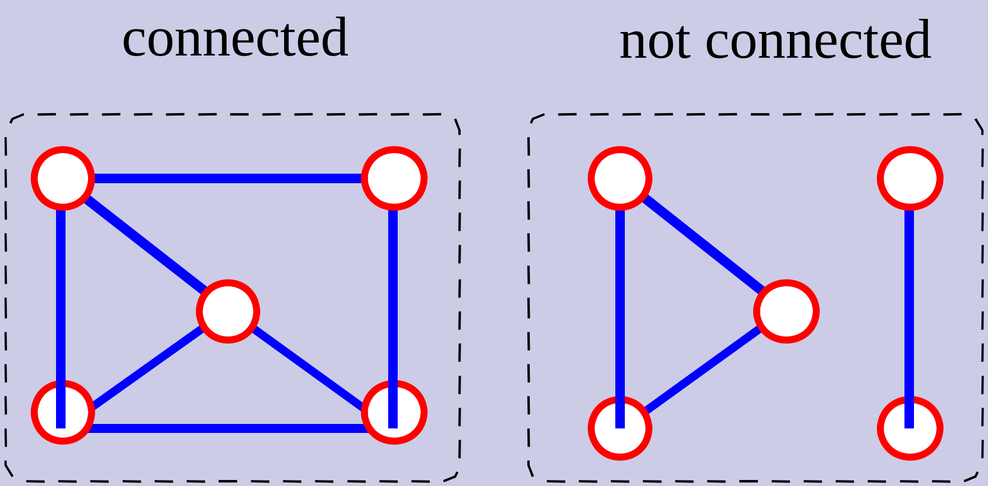


a c d a



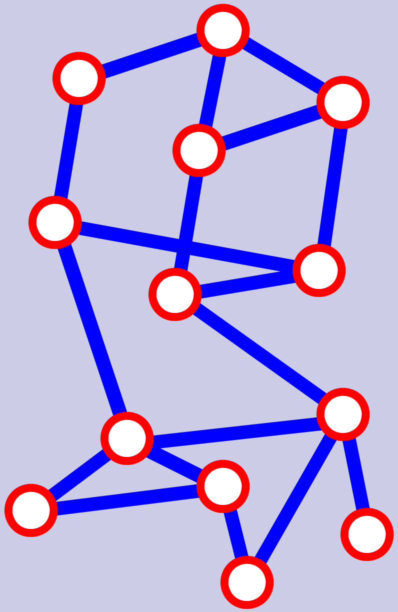
More Terminology

connected graph: any two vertices are connected by some path

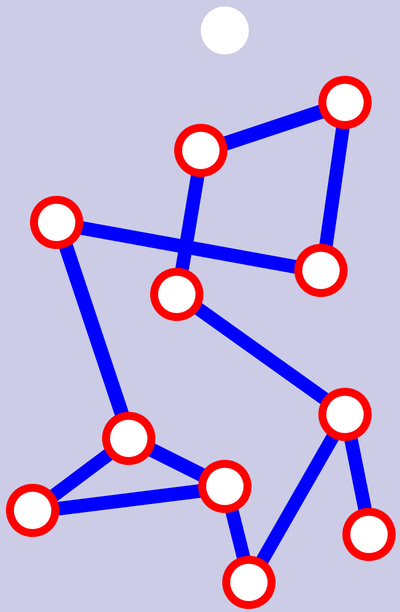


More Terminology(2)

- subgraph: subset of vertices and edges forming a graph



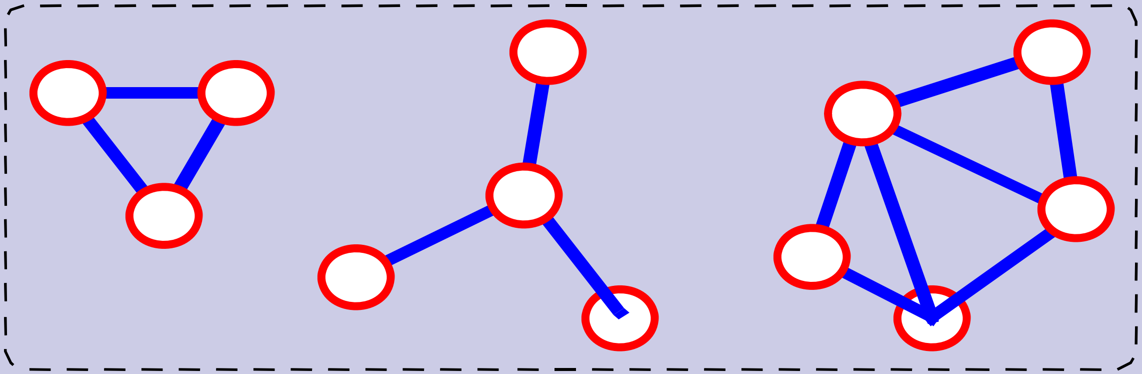
G



subgraph of G

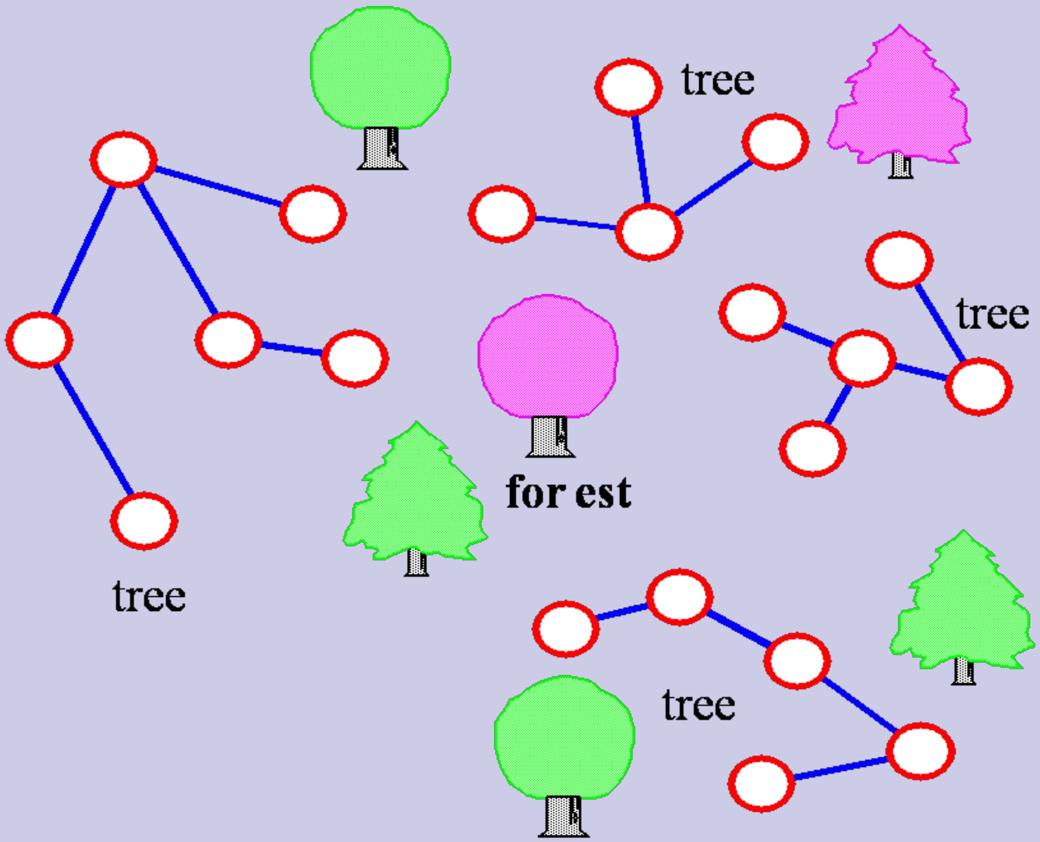
More Terminology(3)

- connected component: maximal connected subgraph. E.g., the graph below has 3 connected components.



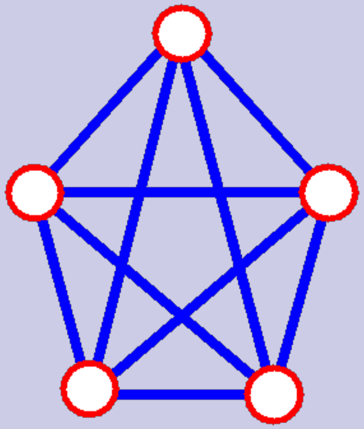
Yet Another Terminology Slide!

- (free) tree - connected graph without cycles
- forest - collection of trees



Connectivity

- Let n = #vertices, and m = #edges
- Complete graph: one in which all pairs of vertices are adjacent
- *How many edges does a complete graph have?*
  - There are n(n-1)/2 pairs of vertices and so m = n(n -1)/2.
- Therefore, if a graph is not complete, m < n(n -1)/2



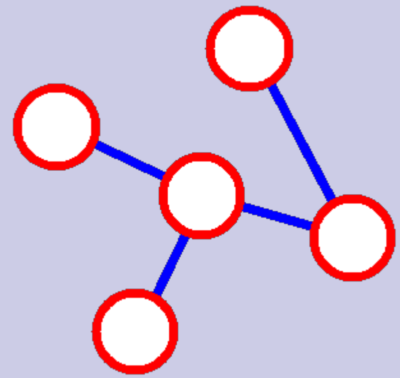
n = 5
m = (5 * 4)/2 = 10

More Connectivity

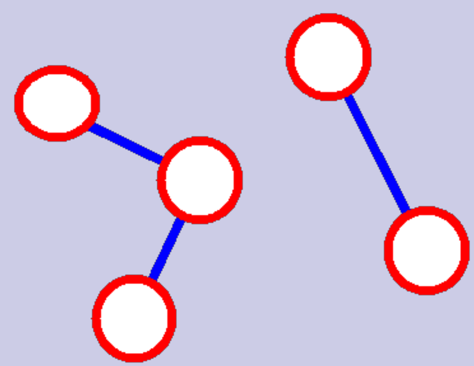
n = #vertices

m = #edges

- For a tree $m = n - 1$
- If $m < n - 1$, G is not connected



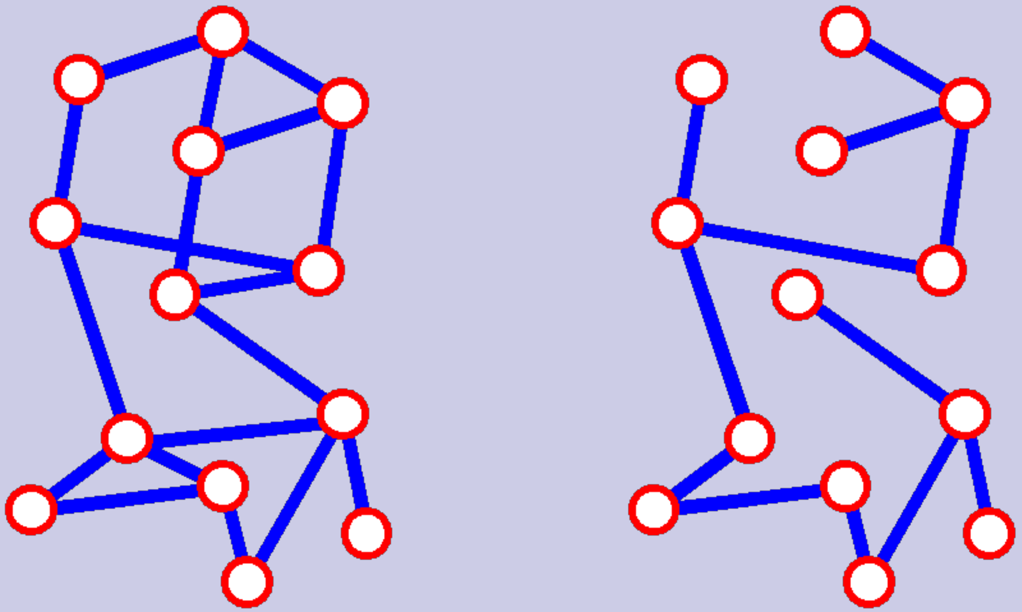
$n = 5$
$m = 4$



$n = 5$
$m = 3$

Spanning Tree

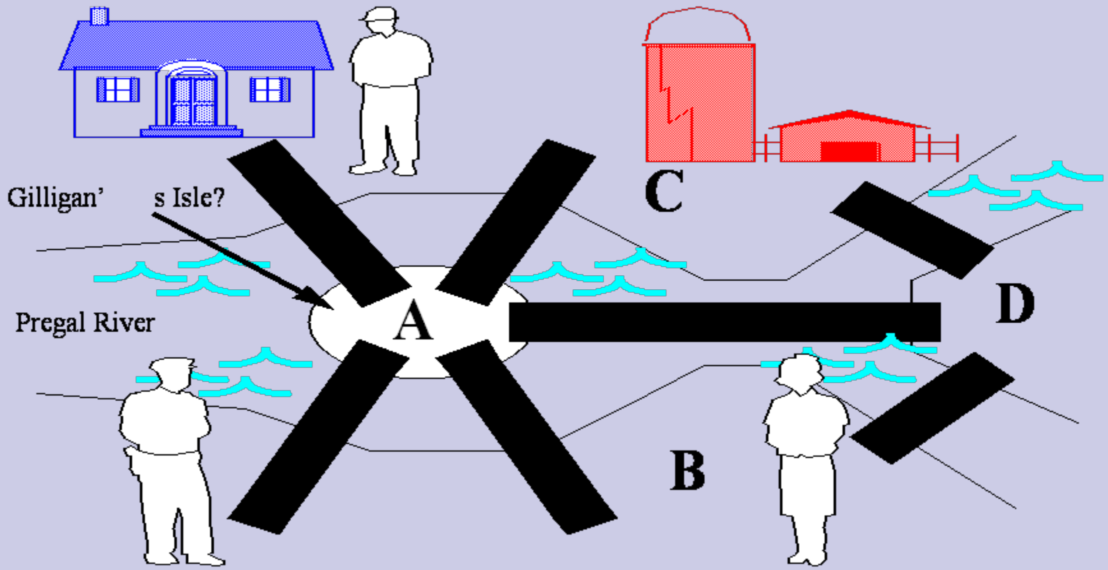
- A spanning tree of G is a subgraph which is a tree and which contains all vertices of G



G　　spanning tree of G

- Failure on any edge disconnects system (least fault tolerant)

The Bridges of Koenigsberg

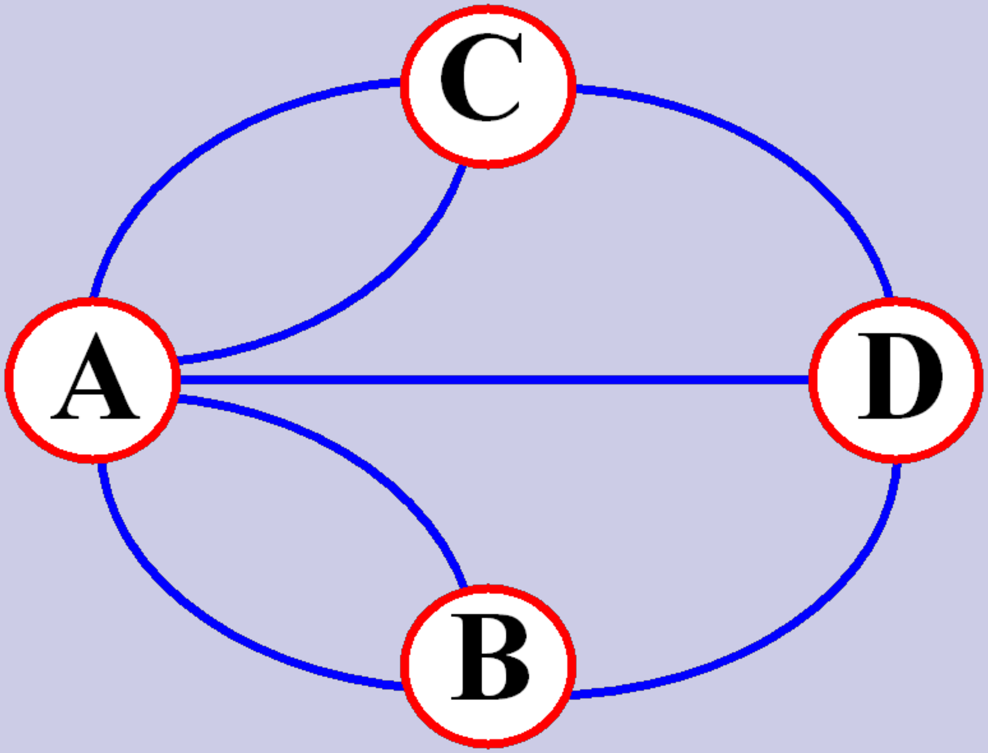


Can one walk across each bridge exactly once and return at the starting point?

- Suppose you are a postman, and you didn't want to retrace your steps.
- In 1736, Euler proved that this is not possible

Graph Model (with parallel edges)

- Eulerian Tour: path that traverses every edge exactly once and returns to the first vertex
- Euler's Theorem: A graph has a Eulerian Tour if and only if all vertices have even degree



The Graph ADT

- The **Graph ADT** is a positional container whose positions are the vertices and the edges of the graph.
  - size() Return the number of vertices + number of edges of G.
  - isEmpty()
  - elements()
  - positions()
  - swap()
  - replaceElement()
- Notation: Graph G; Vertices v, w; Edge e; Object o
  - numVertices() Return the number of vertices of G.
  - numEdges() Return the number of edges of G.
  - vertices() Return an enumeration of the vertices of G.
  - edges() Return an enumeration of the edges of G.

The Graph ADT (contd.)

- directedEdges() enumeration of all directed edges in G.
- undirectedEdges() enumeration of all undirected edges in G.
- incidentEdges(v) enumeration of all edges incident on v.
- inIncidentEdges(v) enumeration of all edges entering v.
- outIncidentEdges(v) enumeration of all edges leaving v.
- opposite(v, e) an endpoint of e distinct from v
- degree(v) the degree of v.
- inDegree(v) the in-degree of v.
- outDegree(v) the out-degree of v.

More Methods ...

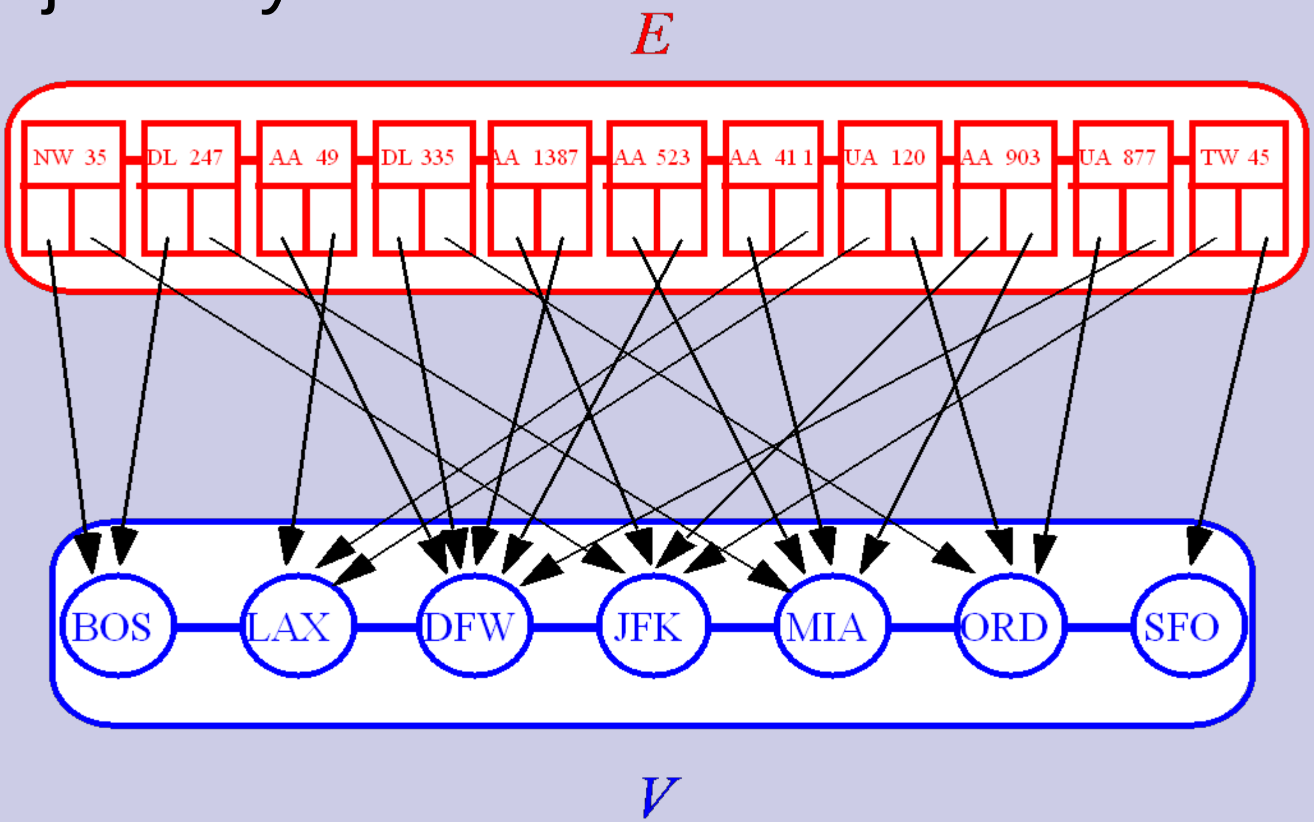
- adjacentVertices(v) enumeration of vertices adjacent to v.
- inAdjacentVertices(v) enumeration of vertices adjacent to v along incoming edges.
- outAdjacentVertices(v) enumeration of vertices adjacent to v along outgoing edges.
- areAdjacent(v,w) whether vertices v and w are adjacent.
- endVertices(e) the end vertices of e.
- origin(e) the end vertex from which e leaves.
- destination(e) the end vertex at which e arrives.
- isDirected(e) true iff e is directed.

Update Methods

- makeUndirected(e) Set e to be an undirected edge.
- reverseDirection(e) Switch the origin and destination vertices of e.
- setDirectionFrom(e, v) Sets the direction of e away from v, one of its end vertices.
- setDirectionTo(e, v) Sets the direction of e toward v, one of its end vertices.
- insertEdge(v, w, o) Insert and return an undirected edge between v and w, storing o at this position.
- insertDirectedEdge(v, w, o) Insert and return a directed edge between v and w, storing o at this position.
- insertVertex(o) Insert and return a new (isolated) vertex storing o at this position.
- removeEdge(e) Remove edge e.

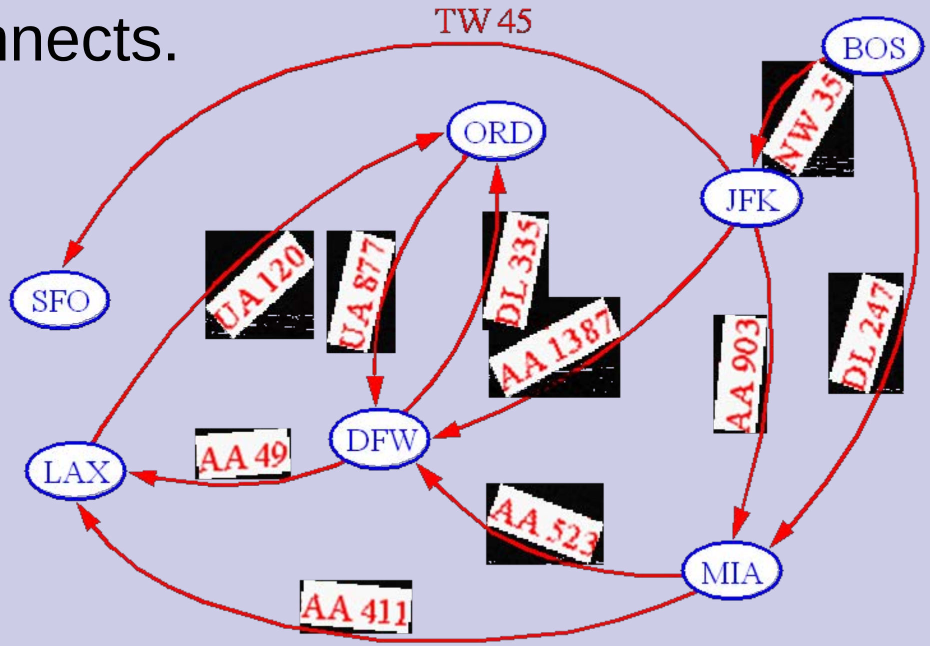
Data Structures for Graphs

- Edge list
- Adjacency lists
- Adjacency matrix



Data Structures for Graphs

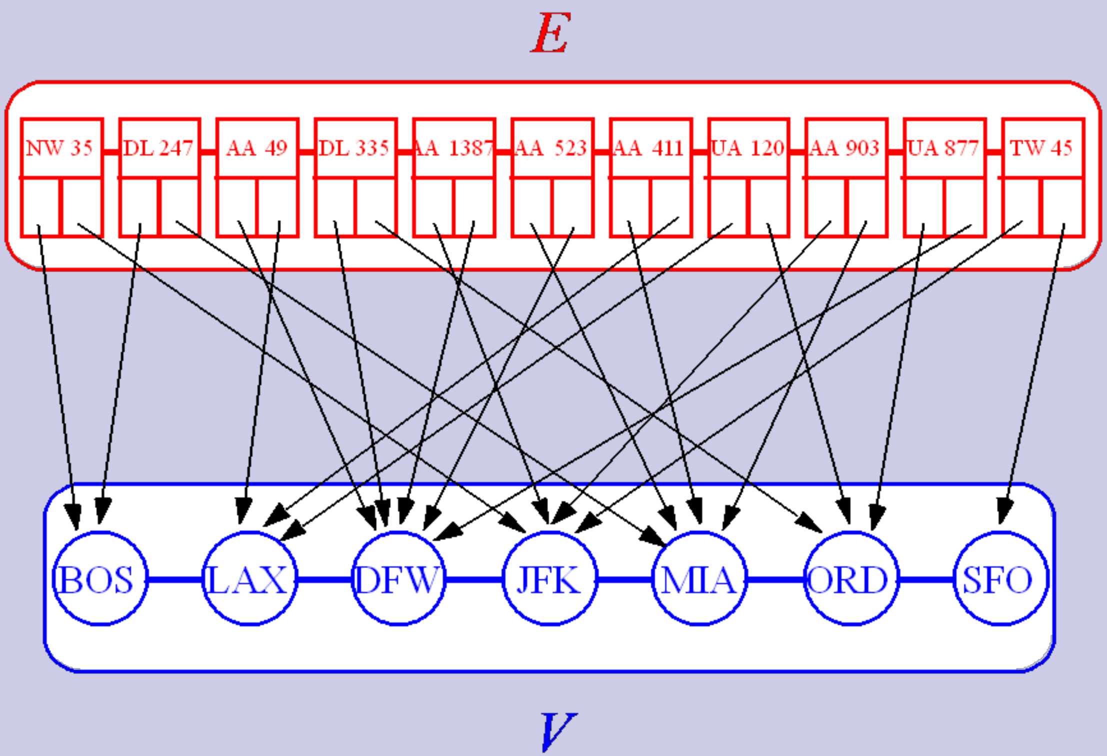
- A Graph! How can we represent it?
- To start with, we store the vertices and the edges into two containers, and each edge object has references to the vertices it connects.



Additional structures can be used to perform efficiently the methods of the Graph ADT

Edge List

- The edge list structure simply stores the vertices and the edges into unsorted sequences.
- Easy to implement.
- Finding the edges incident on a given vertex is inefficient since it requires examining the entire edge sequence



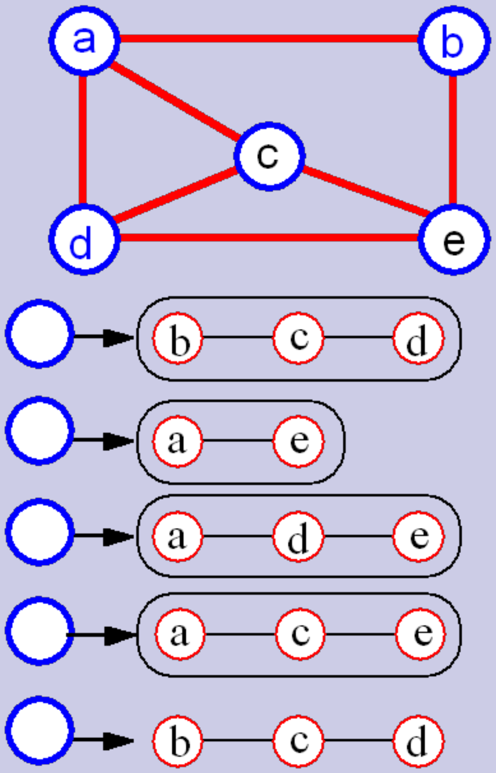
Performance of the Edge List Structure

Operation	Time
size, isEmpty, replaceElement, swap	O(1)
numVertices, numEdges	O(1)
vertices	O(n)
edges, directedEdges, undirectedEdges	O(m)
elements, positions	O(n+m)
endVertices, opposite, origin, destination, isDirected	O(1)
incidentEdges, inIncidentEdges, outIncidentEdges, adjacent Vertices , inAdjacentVertices, outAdjacentVertices,areAdjacent, degree, inDegree, outDegree	O(m)
insertVertex, insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection, setDirectionFrom, setDirectionTo	O(1)
removeVertex	O(m)

Adjacency List (traditional)

- adjacency list of a vertex v:

  sequence of vertices adjacent to v
- represent the graph by the adjacency lists of all the vertices

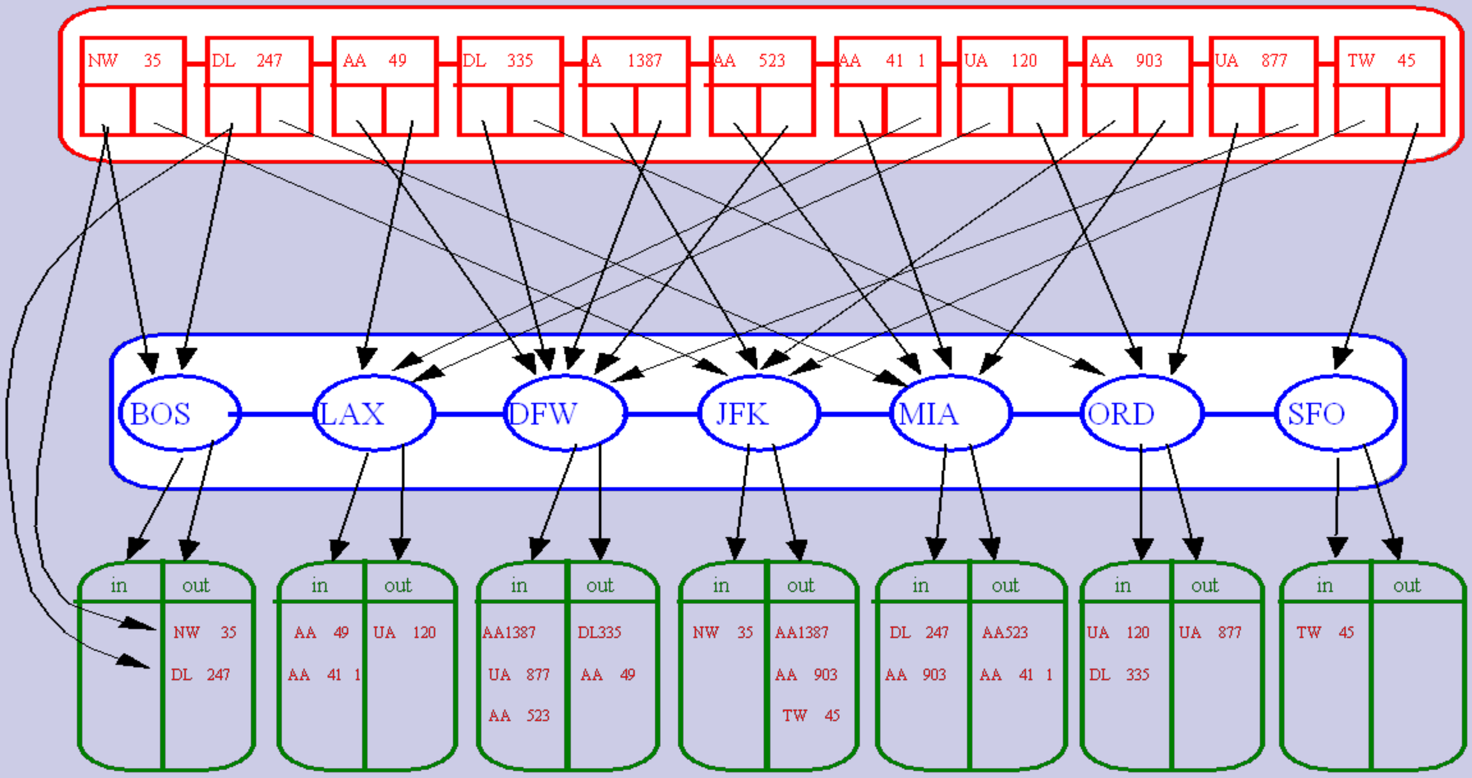


Space =

$$\Theta\,(\mathbf{N} + \Sigma\ \mathbf{deg(v)}) = \Theta(\mathbf{N} + \mathbf{M})$$

Adjacency List (modern)

- The adjacency list structure extends the edge list structure by adding to each vertex.

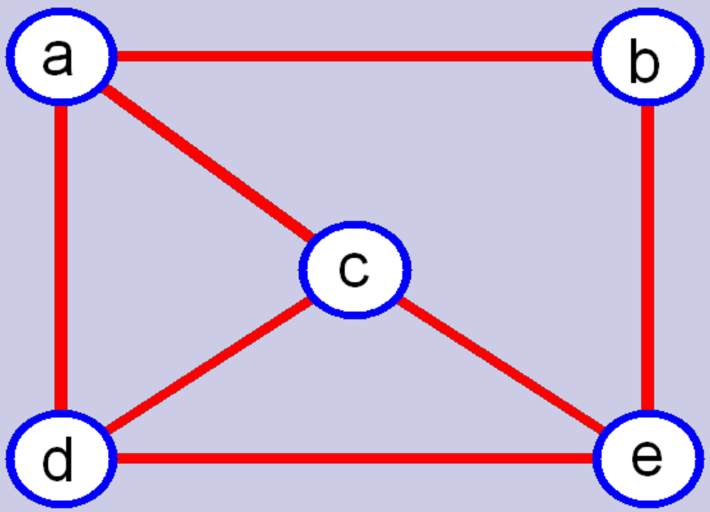


The space requirement is O(n + m).

Performance of the Adjacency List Structure

size, isEmpty, replaceElement, swap	O(1)
numVertices, numEdges	O(1)
vertices	O(n)
edges, directedEdges, undirectedEdges	O(m)
elements, positions	O(n+m)
endVertices, opposite, origin, destination, isDirected, degree, inDegree, outDegree	O(1)
incidentEdges(v), inIncidentEdges(v), outIncidentEdges(v), adjacentVertices(v), inAdjacentVertices(v), outAdjacentVertices(v)	O(deg(v))
areAdjacent(u, v)	O(min(deg(u),deg(v)))
insertVertex, insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection, insertVertex, insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection	O(1)

Adjacency Matrix (traditional)



	a	b	c	d	e
a	F	T	T	T	F
b	T	F	F	F	T
c	T	F	F	T	T
d	T	F	T	F	T
e	F	T	T	T	F

- matrix M with entries for all pairs of vertices
- M[i,j] = true means that there is an edge (i,j) in the graph.
- M[i,j] = false means that there is no edge (i,j) in the graph.
- There is an entry for every possible edge, therefore:

$$\text{Space} = \Theta(N^2)$$

Adjacency Matrix (modern)

- The adjacency matrix structures augments the edge list structure with a matrix where each row and column correspo

	0	1	2	3	4	5	6
0	Ø	Ø	NW 35	Ø	DL 247	Ø	Ø
1	Ø	Ø	Ø	AA 49	Ø	DL 335	Ø
2	Ø	AA 1387	Ø	Ø	AA 903	Ø	TW 45
3	Ø	Ø	Ø	Ø	Ø	UA 120	Ø
4	Ø	AA 523	Ø	AA 411	Ø	Ø	Ø
5	Ø	UA 877	Ø	Ø	Ø	Ø	Ø
6	Ø	Ø	Ø	Ø	Ø	Ø	Ø

BOS	DFW	JFK	LAX	MIA	ORD	SFO
0	1	2	3	4	5	6

Performance of the Adjacency Matrix Structure

Operation	Time
size, isEmpty, replaceElement, swap	$O(1)$
numVertices, numEdges	$O(1)$
vertices	$O(n)$
edges, directedEdges, undirectedEdges	$O(m)$
elements, positions	$O(n+m)$
endVertices, opposite, origin, destination, isDirected, degree, inDegree, outDegree	$O(1)$
incidentEdges, inIncidentEdges, outIncidentEdges, adjacentVertices, inAdjacentVertices, outAdjacentVertices,	$O(n)$
areAdjacent	$O(1)$
insertEdge, insertDirectedEdge, removeEdge, makeUndirected, reverseDirection, setDirectionFrom, setDirectionTo	$O(1)$
insertVertex, removeVertex	$O(n^2)$