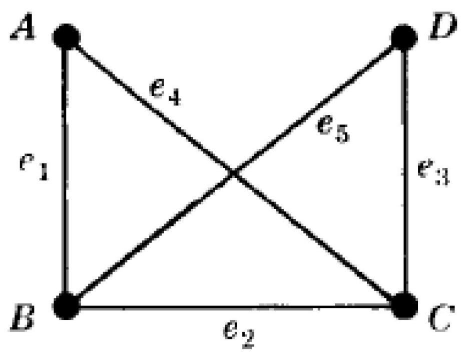
Week Eight

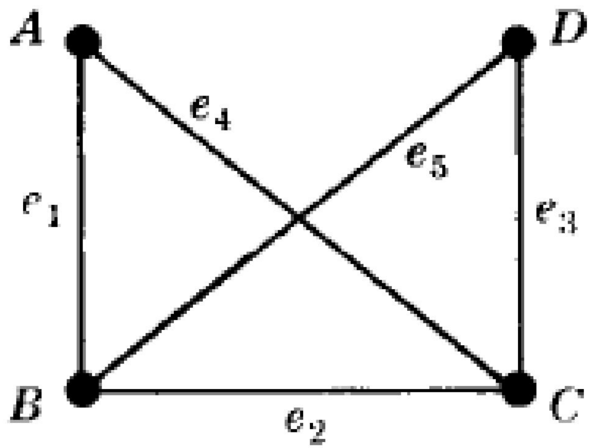
Graphs and Multigraphs

- A graph G consists of two things:
  - (i) A set V = V(G) whose elements are called vertices, points, or nodes of G.
  - (ii) A set E = E(G) of unordered pairs of distinct vertices called edges of G.
- We denote such a graph by G(V,E) when we want to emphasize the two parts of G.

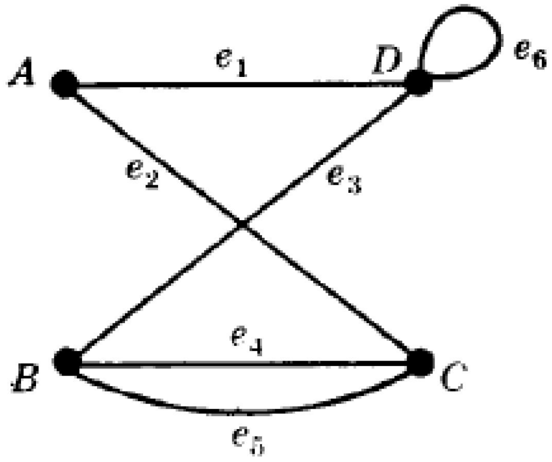


Graphs and Multigraphs

- Vertices u and v are said to be adjacent or neighbors if there is an edge e = {u,v}.
- In such a case, u and v are called the endpoints of e, and e is said to connect u and v.
- Also, the edge e is said to be incident on each of its endpoints u and v.
- Graphs are pictured by diagrams in the plane in a natural way. Specifically, each vertex v in V is represented by a dot (or small circle), and each edge e = {v1, v2} is represented by a curve which connects its endpoints v1 and v2



Multigraphs

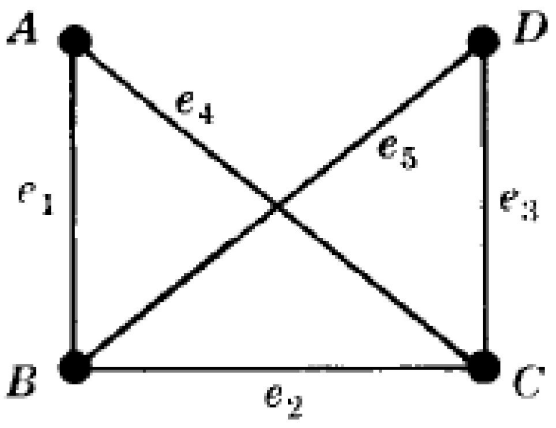


- Consider the diagram on the left.
- The edges e4 and e5 are called multiple edges since they connect the same endpoints, and the edge e6 is called a loop since its endpoints are the same vertex.
- Such a diagram is called a multigraph; the formal definition of a graph permits neither multiple edges nor loops. Thus a graph may be defined to be a multigraph without multiple edges or loops

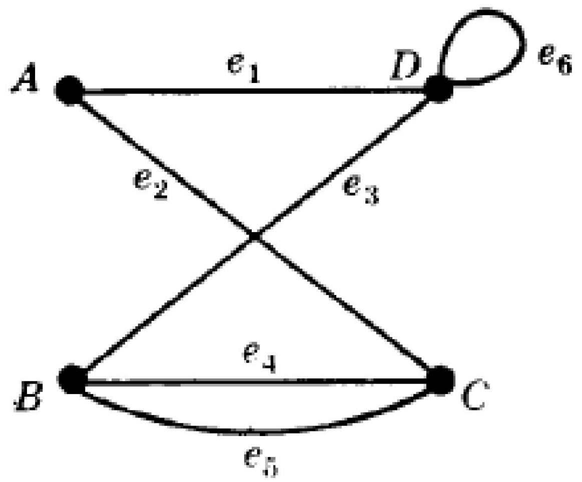
Degree of a Vertex

- The degree of a vertex v in a graph G, written deg (v), is equal to the number of edges in G which contain v, that is, which are incident on v.
- Since each edge is counted twice in counting the degrees of the vertices of G, we have the following simple but important result.
- Theorem 8.1: The sum of the degrees of the vertices of a graph G is equal to twice the number of edges in G.
- Consider, for example, the graph on the right.
- We have

  deg(A) = 2, deg(B) = 3, deg(C) = 3, deg(D) = 2.
- The sum of the degrees equals 10 which, as expected, is twice the number of edges.
- A vertex is said to be even or odd according as its degree is an even or an odd number. Thus A and D are even vertices whereas B and C are odd vertices.



Degree of a Vertex



- Theorem 8.1 also holds for multigraphs where a loop is counted twice toward the degree of its endpoint.
- For example, in the graph on the left we have deg(D) = 4 since the edge e6 is counted twice; hence D is an even vertex.
- A vertex of degree zero is called an isolated vertex.

Finite Graphs, Trivial Graphs

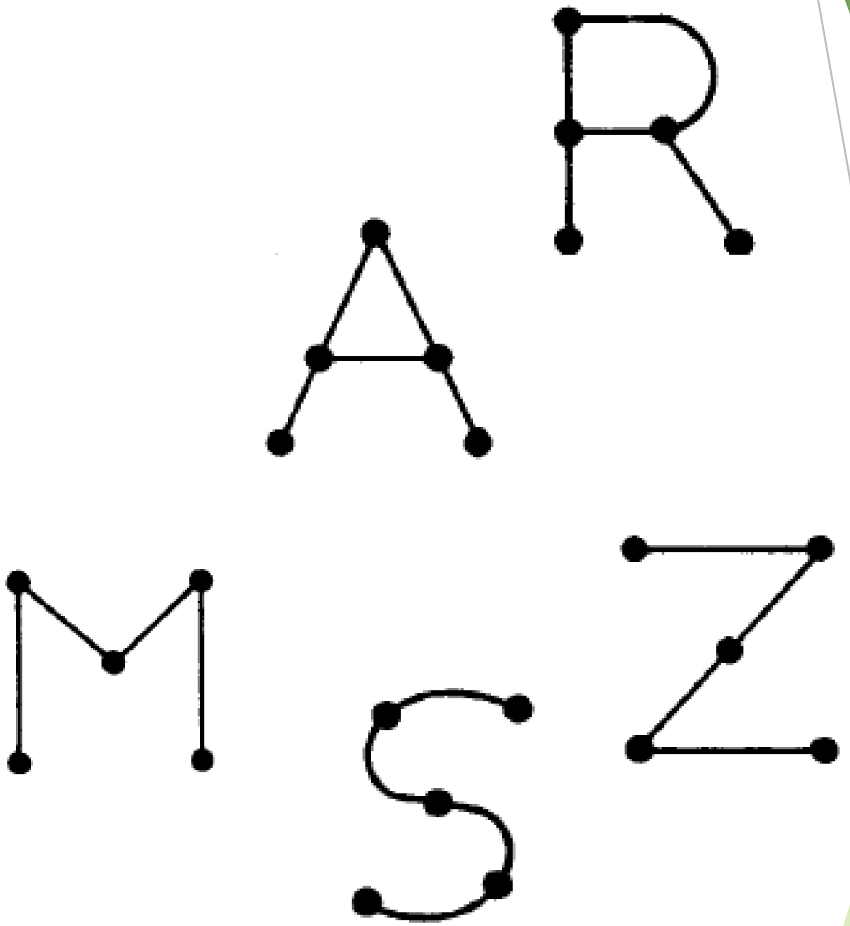
- A multigraph is said to be finite if it has a finite number of vertices and a finite number of edges.
- Observe that a graph with a finite number of vertices must automatically have a finite number of edges and so must be finite.
- The finite graph with one vertex and no edges, i.e., a single point, is called the trivial graph.
- Unless otherwise specified, you may assume that all the multigraphs shall be finite.

SUBGRAPHS, ISOMORPHICAND HOMEOMORPHIC GRAPHS

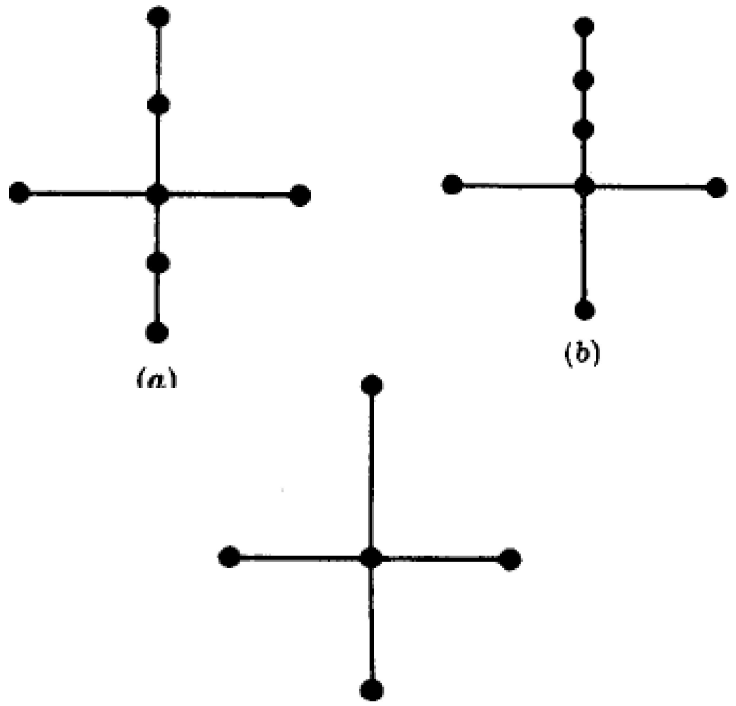
- Subgraphs
- Consider a graph G = G(V,E).Agraph H = H(V’,E’) is called a subgraph of G if the vertices and edges of H are contained in the vertices and edges of G, that is, if V’ ⊆ V and E’⊆ E. In particular:
  - (i) A subgraph H(V’,E’) of G(V,E) is called the subgraph induced by its vertices V’ if its edge set E’ contains all edges in G whose endpoints belong to vertices in H.
  - (ii) If v is a vertex in G, then G − v is the subgraph of G obtained by deleting v from G and deleting all edges in G which contain v.
  - (iii) If e is an edge in G, then G − e is the subgraph of G obtained by simply deleting the edge e from G.

Isomorphic Graphs

- Graphs G(V,E) and G*(V*,E*) are said to be isomorphic if there exists a one-to-one correspondence $f: V \to V*$ such that {u,v} is an edge of G if and only if {f(u),f(v)} is an edge of G*.
- Normally, we do not distinguish between isomorphic graphs (even though their diagrams may "look different")



Homeomorphic Graphs



- Given any graph G, we can obtain a new graph by dividing an edge of G with additional vertices.
- Two graphs G and G* are said to homeomorphic if they can be obtained from the same graph or isomorphic graphs by this method.
- The graphs (a) and (b) on the left are not isomorphic, but they are homeomorphic since they can be obtained from the graph (c) by adding appropriate vertices.

PATHS

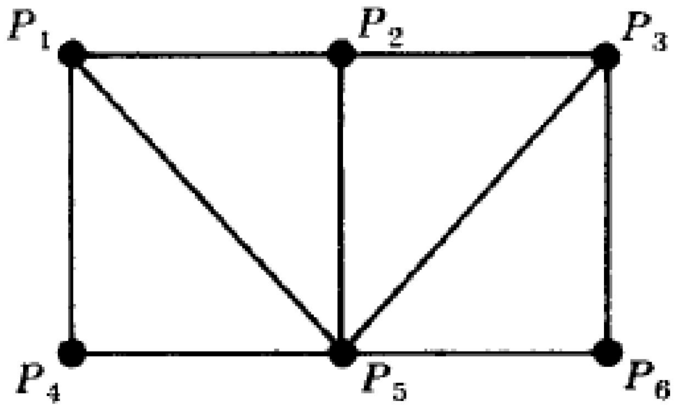
- A path in a multigraph G consists of an alternating sequence of vertices and edges of the form

$$v_0, e_1, v_1, e_2, v_2, \dots, e_{n-1}, v_{n-1}, e_n, v_n$$

- where each edge e_i contains the vertices v_{i-1} and v_i (which appear on the sides of e_i in the sequence).
- The number n of edges is called the length of the path.
- The path is said to be closed if Vo = Vn . Otherwise, we say the path is from Vo to Vn or between Vo and Vn, or connects Vo to Vn.
- A simple path is a path in which all vertices are distinct. A path in which all edges are distinct will be called a trail.
- A cycle is a closed path of length 3 or more in which all vertices are distinct except Vo = Vn. A cycle of length k is called a k-cycle.

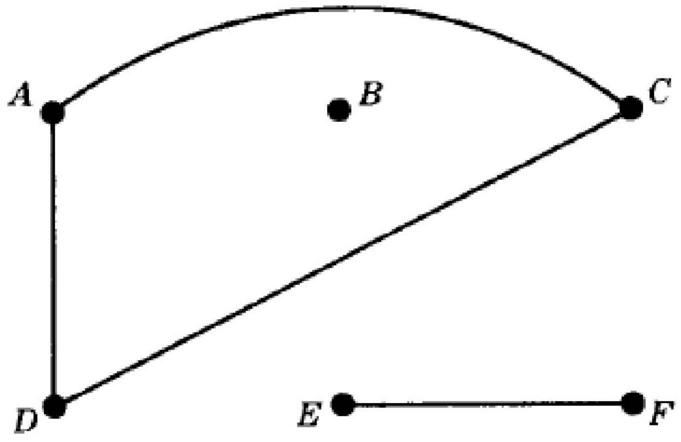
Paths

- Consider the following sequences:
- α = (P4,P1,P2,P5,P1,P2,P3, P6)
- β = (P4,P1,P5,P2,P6)
- γ = (P4,P1,P5,P2,P3,P5,P6)
- The sequence α is a path from P4 to P6; but it is not a trail since the edge {P1,P2} is used twice.
- The sequence β is not a path since there is no edge {P2,P6}
- The sequence γ is a trail since no edge is used twice; but it is not a simple path since the vertex P5 is used twice



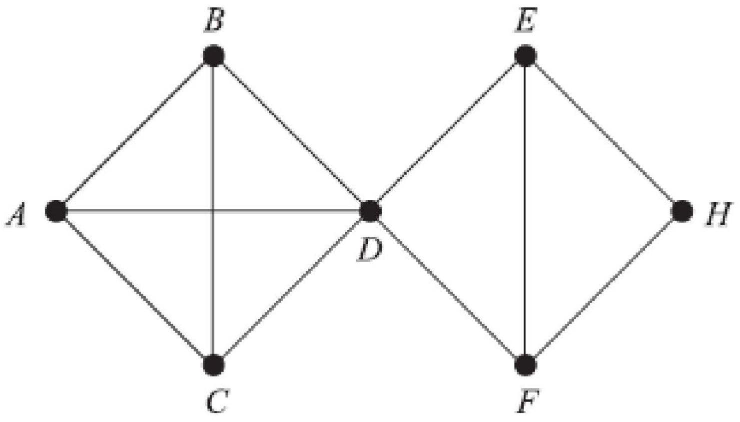
Connectivity/Connected Components

- A graph G is connected if there is a path between any two of its vertices.The graph we just saw is connected, but the graph on the right is not connected since
- Suppose G is a graph. A connected subgraph H of G is called a connected component of G if H is not contained in any larger connected subgraph of G. It is intuitively clear that any graph G can be partitioned into its connected components. The graph on the right has three connected components, the subgraphs induced by the vertex sets {A,C,D}, {E,F}, and {B}.
- The vertex B is called an isolated vertex since B does not belong to any edge or, in other words, deg(B) = 0. B itself forms a connected component of the graph.



(b)

Distance and Diameter

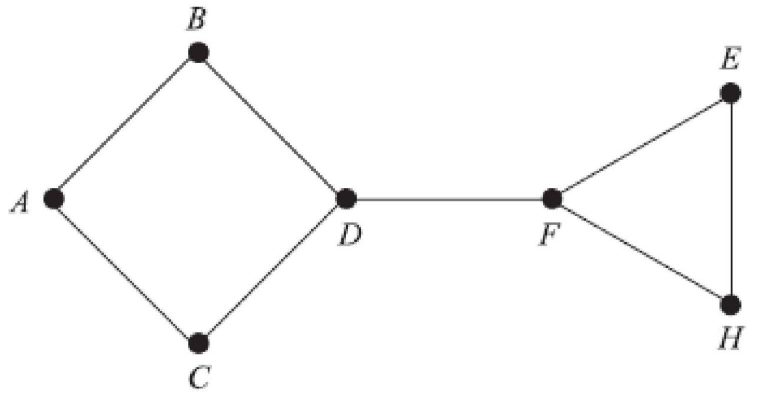


(a)

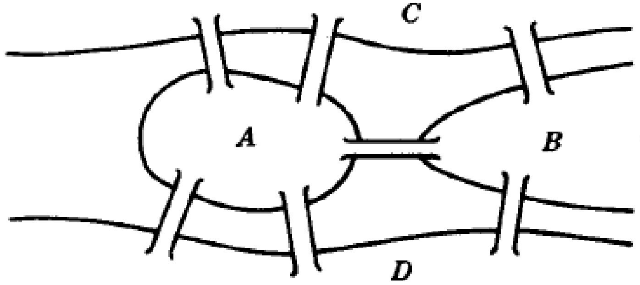
- Consider a connected graph G. The distance between vertices u and v in G, written d(u,v), is the length of the shortest path between u and v.
- The diameter of G, written diam(G), is the maximum distance between any two points in G.
- For example, d(A,F) = 2 and diam(G) = 3

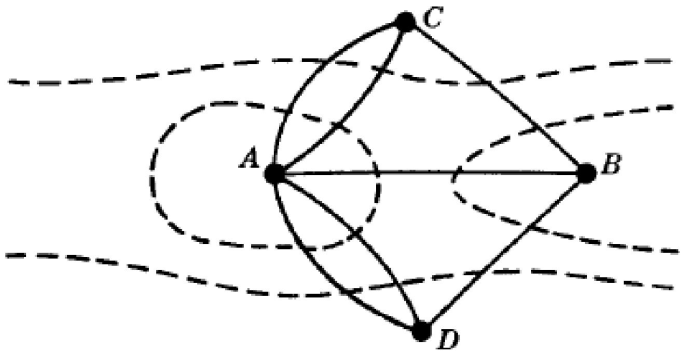
Cutpoints and Bridges

- Let G be a connected graph. A vertex v in G is called a cutpoint if G−v is disconnected.
- An edge e of G is called a bridge if G−e is disconnected. (Recall that G − e is the graph obtained from G by simply deleting the edge e).
- For example, the edge = {D,F} is a bridge.
- Its endpoints D and F are necessarily cutpoints.



TRAVERSABLE AND EULERIAN GRAPHS

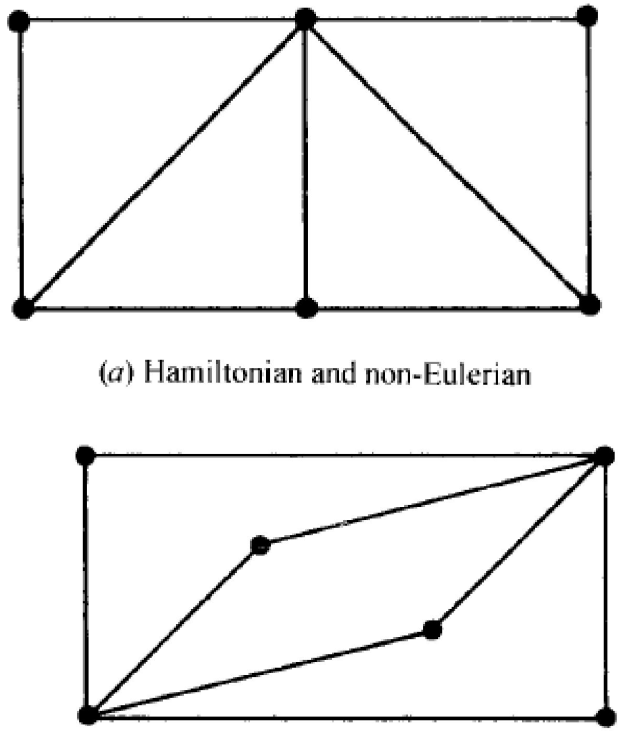




- A multigraph is said to be traversable if it "can be drawn without any breaks in the curve and without repeating any edges," (if there is a path which includes all vertices and uses each edge exactly once)
- Such a path must be a trail (since no edge is used twice) and will be called a traversable trail
- A graph G is called an Eulerian graph if there exists a ***closed traversable trail***, called an Eulerian trail.
- Theorem 8.3 (Euler):
- A finite connected graph is Eulerian if and only if each vertex has even degree

Hamiltonian Graphs

- A Hamiltonian circuit in a graph G, is a closed path that visits every vertex in G exactly once. Such a closed path must be a cycle.
- If G does admit a Hamiltonian circuit, then G is called a Hamiltonian graph.
- Note that an Eulerian circuit traverses every edge exactly once, but may repeat vertices!
- Theorem 8.5: Let G be a connected graph with n vertices. Then G is Hamiltonian if $n \geq 3$ and $n/2 \leq \deg(v)$ for each vertex v in G.

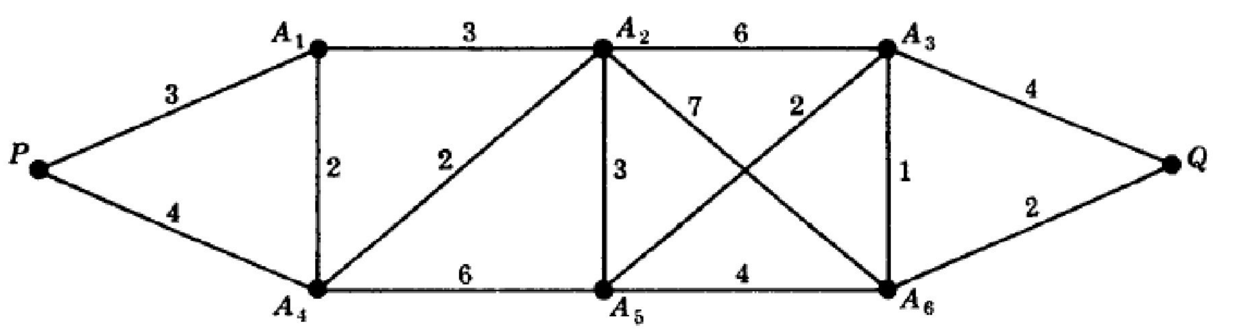


(*a*) Hamiltonian and non-Eulerian

(*b*) Eulerian and non-Hamiltonian

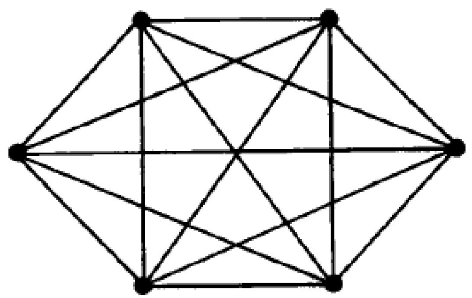
LABELED AND WEIGHTED GRAPHS

- A graph G is called a labeled graph if its edges and/or vertices are assigned data of one kind or another.
- In particular, G is called a weighted graph if each edge e of G is assigned a nonnegative number w(e) called the weight or length of v.
- The weight (or length) of a path in such a weighted graph G is defined to be the sum of the weights of the edges in the path.
- One important problem in graph theory is to find a shortest path, that is, a path of minimum weight (length), between any two given vertices.
- For example, the length of a shortest path between P and Q is 14; one such path is (P,A1,A2,A5,A3,A6,Q)

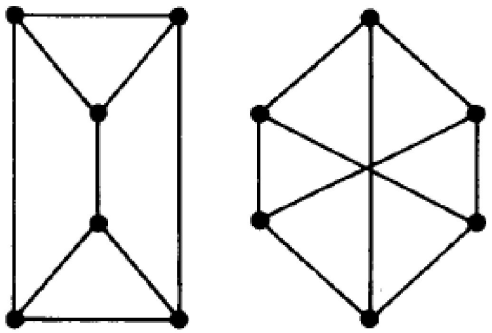


COMPLETE, REGULAR,AND BIPARTITE GRAPHS

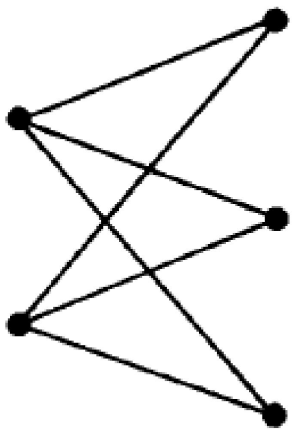
- A graph G is said to be complete if every vertex in G is connected to every other vertex in G. The complete graph with n vertices is denoted by Kn
- A graph G is regular of degree k or k-regular if every vertex has degree k. In other words, a graph is regular if every vertex has the same degree.
- A graph G is said to be bipartite if its vertices V can be partitioned into two subsets M and N such that each edge of G connects a vertex of M to a vertex of N.
- By a complete bipartite graph, we mean that each vertex of M is connected to each vertex of N; this graph is denoted by Km,n where m is the number of vertices in M and n is the number of vertices in N, and, for standardization, we will assume m ≤ n



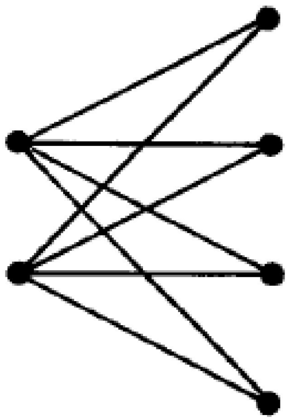
K_6



3-regular



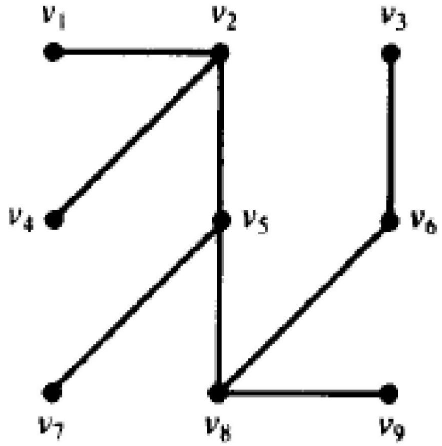
$K_{2,3}$



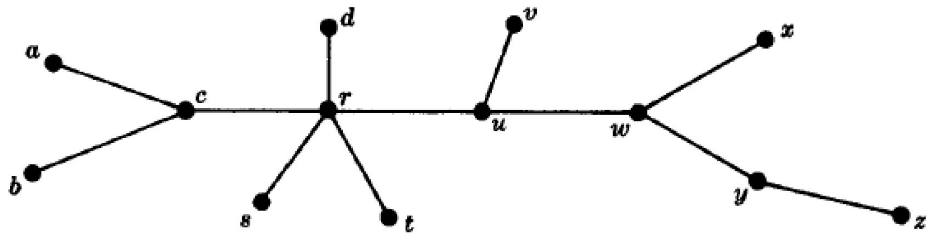
$K_{2,4}$

Trees

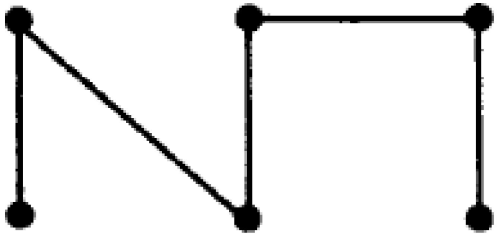
- A graph T is called a tree if T is connected and T has no cycles.
- A forest G is a graph with no cycles; hence the connected components of a forest G are trees. The tree consisting of a single vertex with no edges is called the degenerate tree.
- Consider a tree T. Clearly, there is only one simple path between two vertices of T; otherwise, the two paths would form a cycle. Also:
- (a) Suppose there is no edge {u,v} in T and we add the edge e = {u,v} to T. Then the simple path from u to v in T and e will form a cycle; hence T is no longer a tree.
- (b) On the other hand, suppose there is an edge e = {u,v} in T, and we delete e from T. Then T is no longer connected (since there cannot be a path from u to v); hence T is no longer a tree.



Trees



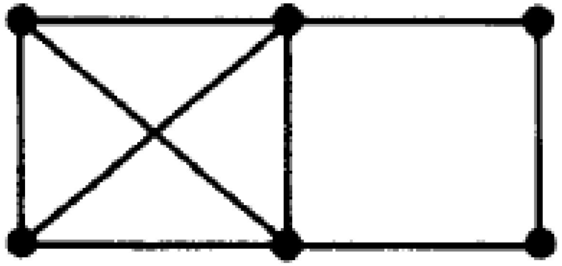
(b)



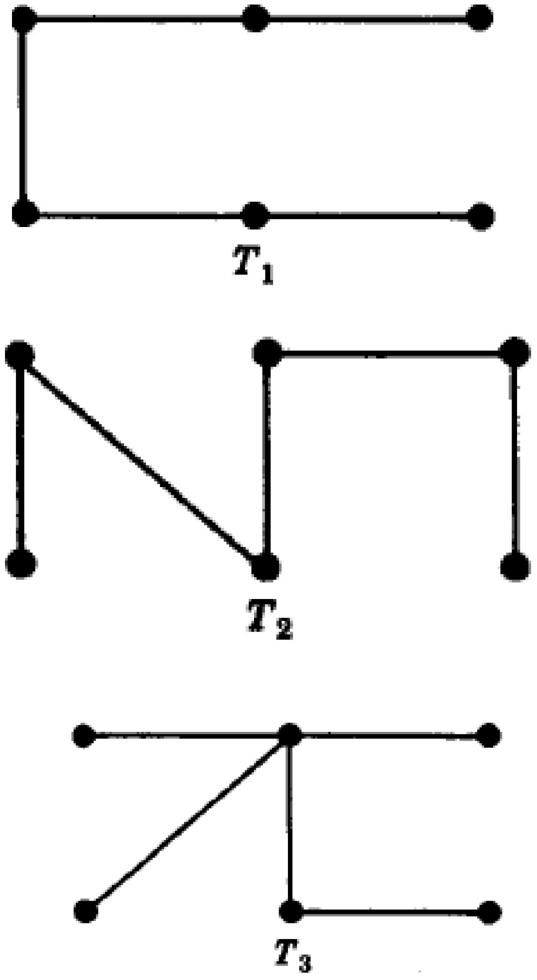
- Theorem 8.6: Let G be a graph with n > 1 vertices. Then the following are equivalent:
- (i) G is a tree.
- (ii) G is a cycle-free and has n − 1 edges.
- (iii) G is connected and has n − 1 edges.

Spanning Trees

- A subgraph T of a connected graph G is called a spanning tree of G if T is a tree and T includes all the vertices of G. Figure 8-18 shows a connected graph G and spanning trees T1, T2, and T3 of G.



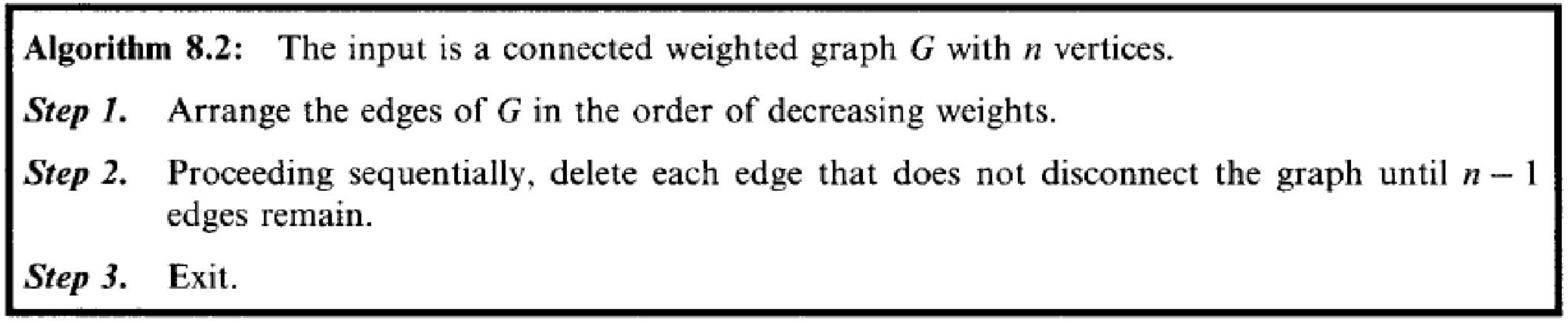
G

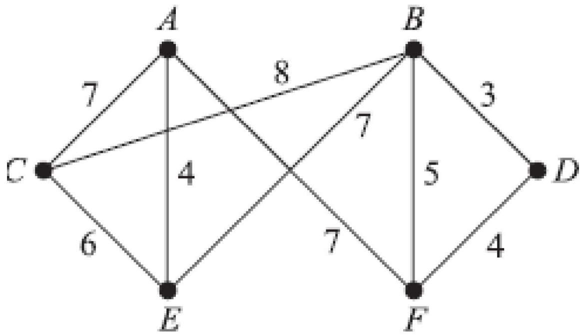


Minimal Spanning Trees

- Suppose G is a connected weighted graph. That is, each edge of G is assigned a nonnegative number called the weight of the edge.
- Then any spanning tree T of G is assigned a total weight obtained by adding the weights of the edges in T. A minimal spanning tree of G is a spanning tree whose total weight is as small as possible.
- The following algorithms enable us to find a minimal spanning tree T of a connected weighted graph G where G has n vertices. (In which case T must have $n - 1$ vertices.)

Minimal Spanning Trees





- Example:
- Find a minimal spanning tree of the weighted graph Q. Note that Q has six vertices, so a minimal spanning tree will have five edges.

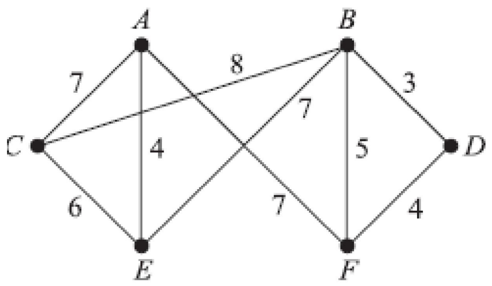
Algorithm 8.2: The input is a connected weighted graph G with n vertices.

Step 1. Arrange the edges of G in the order of decreasing weights.

Step 2. Proceeding sequentially, delete each edge that does not disconnect the graph until $n - 1$ edges remain.

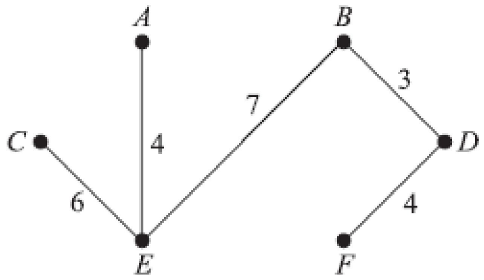
Step 3. Exit.

- First we order the edges by decreasing weights, and then we successively delete edges without disconnecting Q until five edges remain.



- This yields the following data:

Edges	BC	AF	AC	BE	CE	BF	AE	DF	BD
Weight	8	7	7	7	6	5	4	4	3
Delete	Yes	Yes	Yes	No	No	Yes			



- Thus the minimal spanning tree of Q which is obtained contains the edges BE, CE, AE, DF, BD

- The spanning tree has weight 24

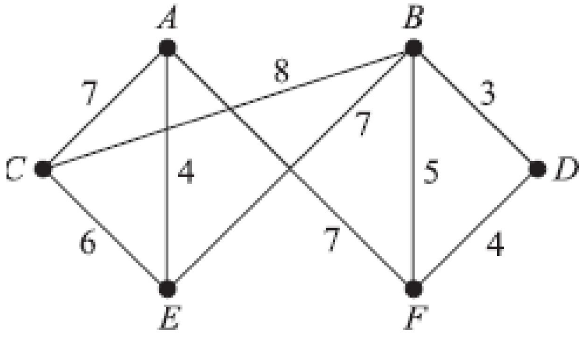
Minimal Spanning Trees

Algorithm 8.3 (Kruskal): The input is a connected weighted graph G with n vertices.

Step 1. Arrange the edges of G in order of increasing weights.

Step 2. Starting only with the vertices of G and proceeding sequentially, add each edge which does not result in a cycle until $n - 1$ edges are added.

Step 3. Exit.



- Example:
- Find a minimal spanning tree of the weighted graph Q. Note that Q has six vertices, so a minimal spanning tree will have five edges.

Algorithm 8.3 (Kruskal): The input is a connected weighted graph G with n vertices.

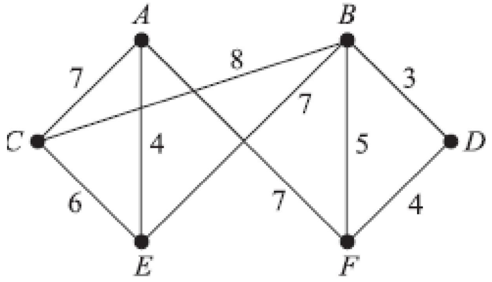
Step 1. Arrange the edges of G in order of increasing weights.

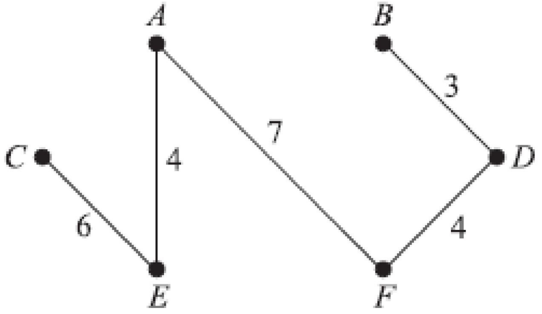
Step 2. Starting only with the vertices of G and proceeding sequentially, add each edge which does not result in a cycle until $n - 1$ edges are added.

Step 3. Exit.

- First we order the edges by increasing weights, and then we successively add edges without forming any cycles until five edges are included. This yields the following data:
- This yields the following data:

Edges	BD	AE	DF	BF	CE	AC	AF	BE	BC
Weight	3	4	4	5	6	7	7	7	8
Add?	Yes	Yes	Yes	No	Yes	No	Yes		

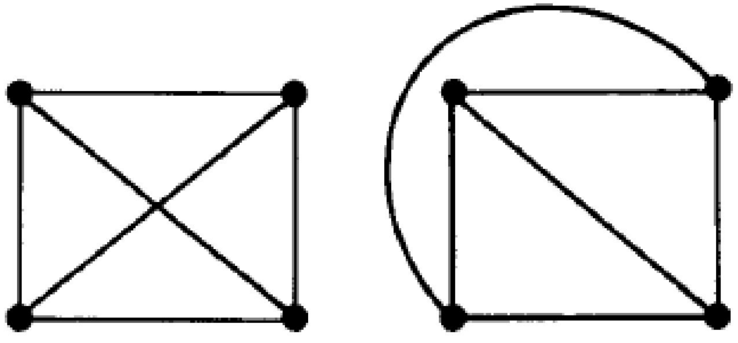




- Thus the minimal spanning tree of Q which is obtained contains the edges BD, AE, DF, CE, AF
- The spanning tree has weight 24

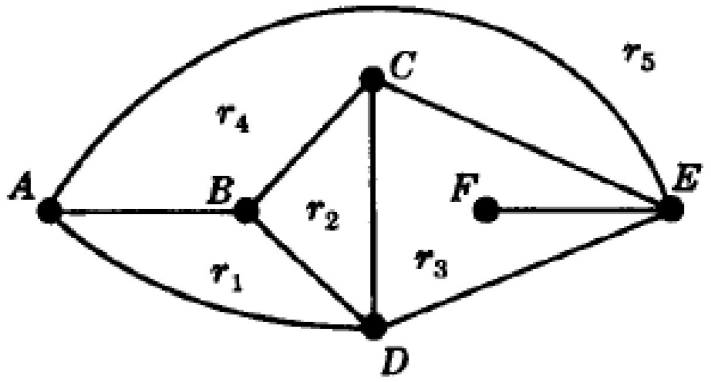
Planar Graphs

- A graph or multigraph which can be drawn in the plane so that its edges do not cross is said to be planar.
- Although the complete graph with four vertices K 4 is usually pictured with crossing edges as in (a), it can also be drawn with noncrossing edges as in (b)
- Hence K 4 is planar.
- Tree graphs form an important class of planar graphs. This section introduces our reader to these important graphs.



(a) (b)

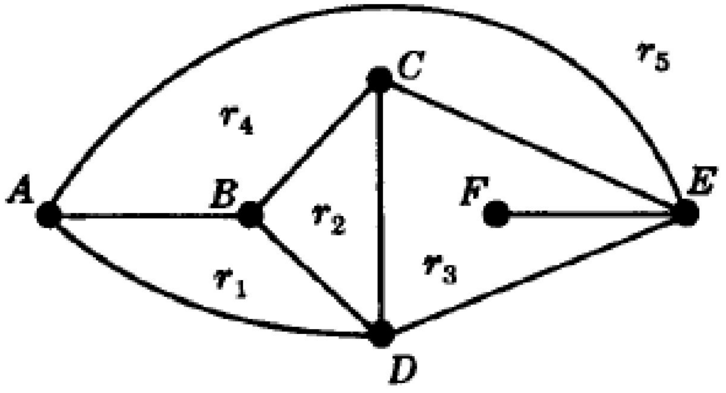
Maps, Regions



- A particular planar representation of a finite planar multigraph is called a map. We say that the map is connected if the underlying multigraph is connected. A given map divides the plane into various regions.
- Observe that four of the regions are bounded, but the fifth region, outside the diagram, is unbounded. Observe that the border of each region of a map consists of edges. Sometimes the edges will form a cycle, but sometimes not. For example, the borders of all the regions are cycles except for r3.
- However, if we do move counterclockwise around r 3 starting, say, at the vertex C, then we obtain the closed path (C,D,E,F,E,C) where the edge {E,F} occurs twice.
- By the degree of a region r, written deg(r), we mean the length of the cycle or closed walk which borders r. We note that each edge either borders two regions or is contained in a region and will occur twice in any walk along the border of the region.

Maps, Regions

- Theorem 8.7: The sum of the degrees of the regions of a map is equal to twice the number of edges.
- Euler's Formula
- Euler gave a formula which connects the number V of vertices, the number E of edges, and the number R of regions of any connected map. Specifically:
- Theorem 8.8 (Euler): $V - E + R = 2$

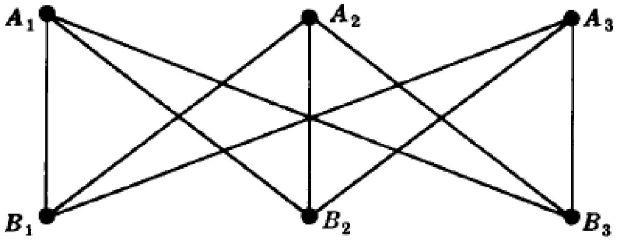


- The degrees of the regions of Fig. 8-22 are: deg(r 1) = 3, deg(r 2) = 3, deg(r 3) = 5, deg(r 4) = 4, deg(r 5) = 3. The sum of the degrees is 18, which, as expected, is twice the number of edges

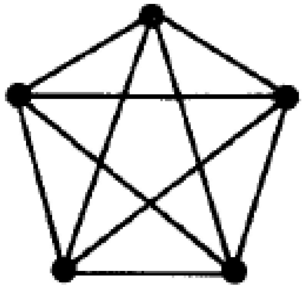
Non-planar Graps

- Theorem 8.10: (Kuratowski)

- A graph is nonplanar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or K_5



(a) $K_{3,3}$

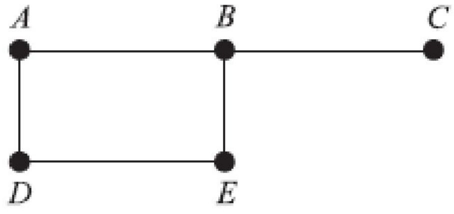


(b) K_5

REPRESENTING GRAPHS IN COMPUTER MEMORY

- Adjacency Matrix
- Suppose G is a graph with m vertices, and suppose the vertices have been ordered, say, V1,V2 ,...,Vm .Then the adjacency matrix A = $[a_{ij}]$ of the graph G is the m × m matrix defined by

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$



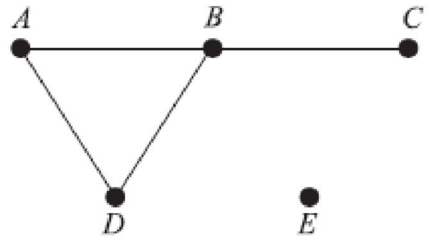
(a)

$$\begin{array}{c c} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

(b)

REPRESENTING GRAPHS IN COMPUTER MEMORY

- Linked Representation of a Graph G
- G may be represented in memory by some type of linked representation, also called an adjacency structure, which is described below by means of an example.
- Observe that G may be defined by the table:



(a)

Vertex	Adjacency list
A	*B*, *D*
B	*A*, *C*, *D*
C	*B*
D	*A*, *B*
E	∅

(b)

- This table may also be presented in the compact form

G = [A:B,D; B:A,C,D; C:B; D:A,B; E:Ø]

REPRESENTING GRAPHS IN COMPUTER MEMORY

- The linked representation of a graph G, which maintains G in memory by using its adjacency lists, will normally contain two files (or sets of records), one called the Vertex File and the other called the Edge File, as follows.
- (a) Vertex File: The Vertex File will contain the list of vertices of the graph G usually maintained by an array or by a linked list. Each record of the Vertex File will have the form

VERTEX	NEXT-V	PTR	

- Here VERTEX will be the name of the vertex, NEXT-V points to the next vertex in the list of vertices in the Vertex File when the vertices are maintained by a linked list, and PTR will point to the first element in the adjacency list of the vertex appearing in the Edge File.
- The shaded area indicates that there may be other information in the record corresponding to the vertex.

REPRESENTING GRAPHS IN COMPUTER MEMORY

- Edge File: The Edge File contains the edges of the graph G. Specifically, the Edge File will contain all the adjacency lists of G where each list is maintained in memory by a linked list. Each record of the Edge File will correspond to a vertex in an adjacency list and hence, indirectly, to an edge of G. The record will usually have the form

EDGE	ADJ	NEXT	

- Here:
  - (1) EDGE will be the name of the edge (if it has one).
  - (2) ADJ points to the location of the vertex in the Vertex File.
  - (3) NEXT points to the location of the next vertex in the adjacency list.
- We emphasize that each edge is represented twice in the Edge File, but each record of the file corresponds to a unique edge. The shaded area indicates that there may be other information in the record corresponding to the edge.

REPRESENTING GRAPHS IN COMPUTER MEMORY

START: 4

Vertex file

	1	2	3	4	5	6	7	8
VERTEX	*B*		*F*	*D*	*A*		*C*	*E*
NEXT-V	3		5	1	8		0	7
PTR	9		4	7	6		5	12

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADJ	4	4	1	8	8	1	5	3	5	8	4	7		
NEXT	8	0	10	0	0	2	3	0	11	0	0	1		

REPRESENTING GRAPHS IN COMPUTER MEMORY

- List the vertices in the order they appear in memory:

- Since START = 4, the list begins with the vertex D. The NEXT-V tells us to go to 1(B), then 3(F), then 5(A), then 8(E), and then 7(C); that is, D, B, F, A, E, C

Vertex file

START 4

	1	2	3	4	5	6	7	8
VERTEX	*B*		*F*	*D*	*A*		*C*	*E*
NEXT-V	3		5	1	8		0	7
PTR	9		4	7	6		5	12

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADJ	4	4	1	8	8	1	5	3	5	8	4	7		
NEXT	8	0	10	0	0	2	3	0	11	0	0	1		

REPRESENTING GRAPHS IN COMPUTER MEMORY

- Find the adjacency list adj(v) of each vertex v of G
- Here adj(D) = [5(A),1(B),8(E)].
- Specifically, PTR[4(D)] = 7 and ADJ[7] = 5(A) tells us that adj(D) begins with A.
- Then NEXT[7] = 3 and ADJ[3] = 1(B) tells us that B is the next vertex in adj(D).
- Then NEXT[3] = 10and ADJ[10] = 8(E) tells us that E in the next vertex in adj(D).
- However, NEXT[10] = 0 tells us that there are no more neighbors of D.

Vertex file

START 4

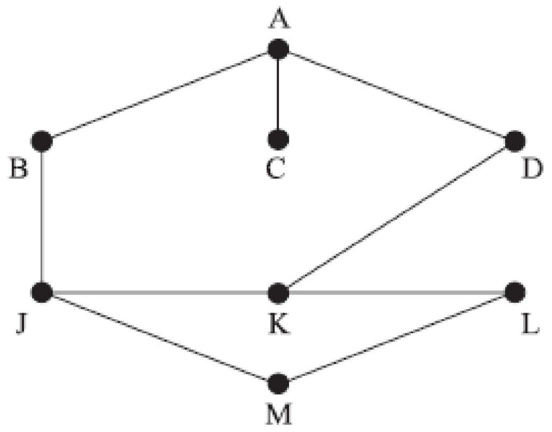
	1	2	3	4	5	6	7	8
VERTEX	*B*		*F*	*D*	*A*		*C*	*E*
NEXT-V	3		5	1	8		0	7
PTR	9		4	7	6		5	12

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADJ	4	4	1	8	8	1	5	3	5	8	4	7		
NEXT	8	0	10	0	0	2	3	0	11	0	0	1		

GRAPH ALGORITHMS

- This section discusses two important graph algorithms which systematically examine the vertices and edges of a graph G.
- One is called a depth-first search (DFS) and the other is called a breadth-first search (BFS).
- Any particular graph algorithm may depend on the way G is maintained in memory. Here we assume G is maintained in memory by its adjacency structure. Here is our test graph G (we assume the vertices are ordered alphabetically)



GRAPH ALGORITHMS

- During the execution of our algorithms, each vertex (node) N of G will be in one of three states, called the status of N, as follows:
  - STATUS = 1: (Ready state) The initial state of the vertex N.
  - STATUS = 2: (Waiting state) The vertex N is on a (waiting) list, waiting to be processed.
  - STATUS = 3: (Processed state) The vertex N has been processed.
- The waiting list for the depth-first seach (DFS) will be a (modified) STACK (which we write horizontally with the top of STACK on the left), whereas the waiting list for the breadth-first search (BFS) will be a QUEUE.

DFS

- The general idea behind a depth-first search beginning at a starting vertex A is as follows.
- First we process the starting vertex A. Then we process each vertex N along a path P which begins at A; that is, we process a neighbor of A, then a neighbor of a neighbor, and so on.
- After coming to a “dead end,” that is to a vertex with no unprocessed neighbor, we backtrack on the path P until we can continue along another path P’. And so on.
- The backtracking is accomplished by using a STACK to hold the initial vertices of future possible paths. We also need a field STATUS which tells us the current status of any vertex so that no vertex is processed more than once.

DFS

Algorithm 8.5 (Depth-first Search): This algorithm executes a depth-first search on a graph G beginning with a starting vertex A.

Step 1. Initialize all vertices to the ready state (STATUS = 1).

Step 2. Push the starting vertex A onto STACK and change the status of A to the waiting state (STATUS = 2).

Step 3. Repeat Steps 4 and 5 until STACK is empty.

Step 4. Pop the top vertex N of STACK. Process N, and set STATUS (N) = 3, the processed state.

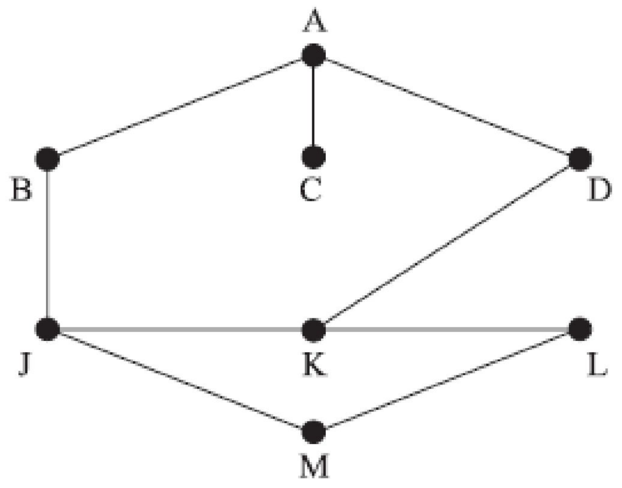
Step 5. Examine each neighbor J of N.

(*a*) If STATUS (J) = 1 (ready state), push J onto STACK and reset STATUS (J) = 2 (waiting state).
(*b*) If STATUS (J) = 2 (waiting state), delete the previous J from the STACK and push the current J onto STACK.
(*c*) If STATUS (J) = 3 (processed state), ignore the vertex J.
[End of Step 3 loop.]

Step 6. Exit.

DFS

STACK	A	DCB	KCB	$LJCB$	$MJCB$	JCB	CB	B	$\emptyset$
Vertex	A	D	K	L	M	J	C	B	



- During the DFS algorithm, the first vertex N in STACK is processed and the neighbors of N (which have not been previously processed) are then pushed onto STACK
- Initially, the beginning vertex A is pushed onto STACK. The following shows the sequence of waiting lists in STACK and the vertices being processed

BFS

- The general idea behind a breadth-first search beginning at a starting vertex A is as follows.
- First we process the starting vertex A. Then we process all the neighbors of A. Then we process all the neighbors of neighbors of A. And so on.
- Naturally we need to keep track of the neighbors of a vertex, and we need to guarantee that no vertex is processed twice. This is accomplished by using a QUEUE to hold vertices that are waiting to be processed, and by a field STATUS which tells us the current status of a vertex.

BFS

Algorithm 8.6 (Breadth-first Search): This algorithm executes a breadth-first search on a graph G beginning with a starting vertex A.

Step 1. Initialize all vertices to the ready state (STATUS = 1).

Step 2. Put the starting vertex A in QUEUE and change the status of A to the waiting state (STATUS = 2).

Step 3. Repeat Steps 4 and 5 until QUEUE is empty.

Step 4. Remove the front vertex N of QUEUE. Process N, and set STATUS $(N) = 3$, the processed state.

Step 5. Examine each neighbor J of N.

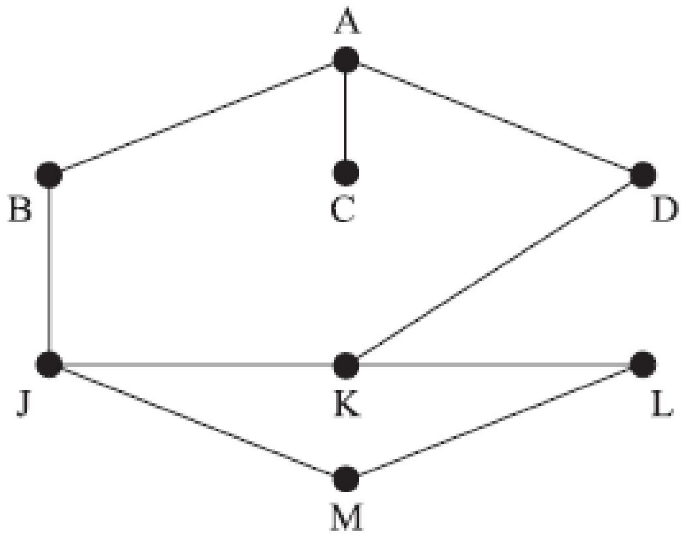
(*a*) If STATUS $(J) = 1$ (ready state), add J to the rear of QUEUE and reset STATUS $(J) = 2$ (waiting state).

(*b*) If STATUS $(J) = 2$ (waiting state) or STATUS $(J) = 3$ (processed state), ignore the vertex J.

[End of Step 3 loop.]

Step 6. Exit.

BFS



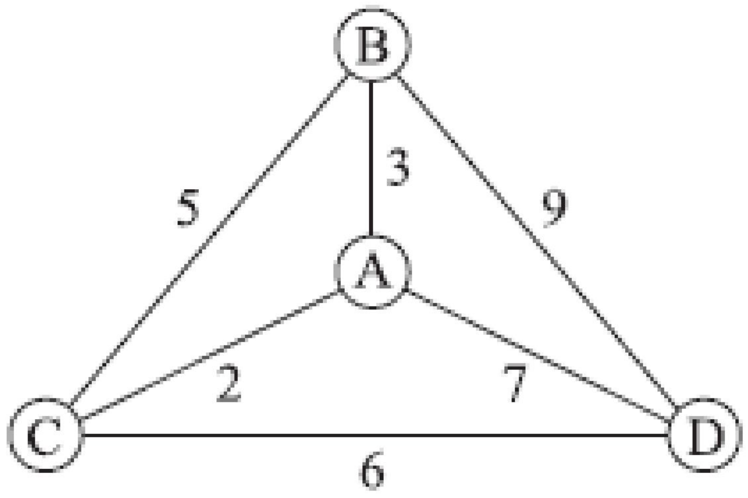
QUEUE	A	DCB	DC	LKD	LK	MJL	MJ	M	$\varnothing$
Vertex	A	B	C	D	K	L	J	M	

Traveling Salesman Problem

- Let G be a complete weighted graph. (We view the vertices of G as cities, and the weighted edges of G as
- the distances between the cities.) The “traveling-salesman” problem refers to finding a Hamiltonian circuit for G of minimum weight.
- First we note the following theorem:
- Theorem 8.13: The complete graph K n with $n \geq 3$ vertices has $H = (n - 1)!/2$ Hamiltonian circuits (where we do not distinguish between a circuit and its reverse).

Traveling Salesman Problem

- Consider the complete weighted graph G in Fig. 8-35(a). It has four vertices, A, B, C, D.
- By the previous theorem it has H = 3!/2 = 3 Hamiltonian circuits. Assuming the circuits begin at the vertex A, the following are the three circuits and their weights:
- |ABCDA| = 3 + 5 + 6 + 7 = 21
- |ACDBA| = 2 + 6 + 9 + 3 = 20
- |ACBDA| = 2 + 5 + 9 + 7 = 23



Traveling Salesman Problem

- We solved the “traveling-salesman problem” for the weighted complete graph by listing and finding the weights of its three possible Hamiltonian circuits. However, for a graph with many vertices, this may be impractical or even impossible.
- For example, a complete graph with 15 vertices has over 40 million Hamiltonian circuits. Accordingly, for circuits with many vertices, a strategy of some kind is needed to solve or give an approximate solution to the traveling-salesman problem.
- We discuss one of the simplest algorithms here.
- Nearest-NeighborAlgorithm
- The nearest-neighbor algorithm, starting at a given vertex, chooses the edge with the least weight to the next possible vertex, that is, to the “closest” vertex. This strategy is continued at each successive vertex until a Hamiltonian circuit is completed.

Traveling Salesman Problem

- Starting at P, the first row of the table shows us that the closest vertex to P is S with distance 15. The fourth row shows that the closest vertex to S is Q with distance 12. The closest vertex to Q is R with distance 11. From R, there is no choice but to go to T with distance 10. Finally, from T, there is no choice but to go back to P with distance 20. Accordingly, the nearest-neighbor algorithm beginning at P yields the following weighted Hamiltonian circuit:
- |PSQRTP| = 15 + 12 + 11 + 10 + 20 = 68

	P	Q	R	S	T
P		18	22	15	20
Q	18		11	12	22
R	22	11		16	10
S	15	12	16		13
T	20	22	10	13	

Traveling Salesman Problem

- Starting at Q, the closest vertex is R with distance 11; from R the closest is T with distance 10; and from T the closest is S with distance 13. From S we must go to P with distance 15; and finally from P we must go back to Q with distance 18.
- Accordingly, the nearest-neighbor algorithm beginning at Q yields the following weighted Hamiltonian circuit:
- |QRTSPQ| = 11 + 10 + 13 + 15 + 18 = 67

	P	Q	R	S	T
P		18	22	15	20
Q	18		11	12	22
R	22	11		16	10
S	15	12	16		13
T	20	22	10	13	

Questions?