Graphs

Graphs together with lists, stacks and queues are fundamental structures in computer science. They can be used to solve many real-life complex problems.

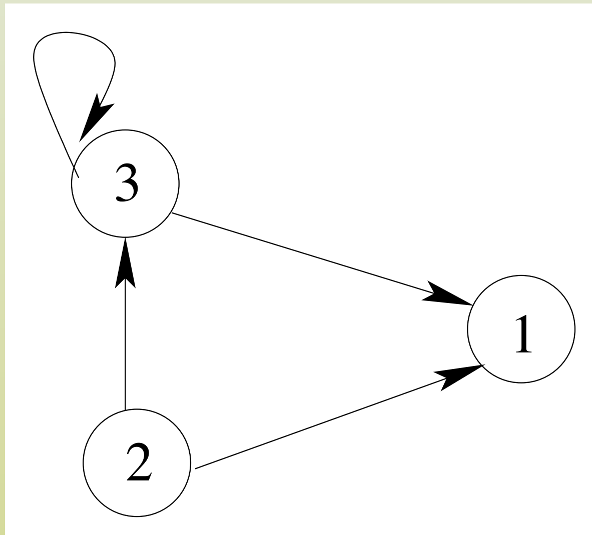
Graphs:

- definitions
- terminology
- representation
- search algorithms: Breadth First Search and Depth First Search
- shortest path algorithms—Dijkstra's algorithm
- minimum spanning tree

DEFINITION:

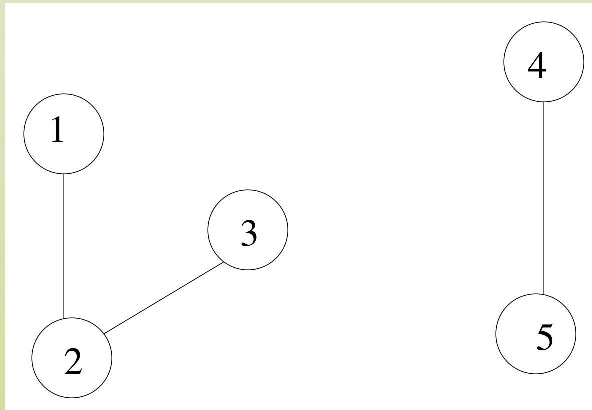
A *graph* G is a pair (V, E), where V is a set of *vertices*, and E is a set of *edges*. Each edge is associated with two vertices, which are endpoints of the edge. Graphs may be either *directed* or *undirected*. A directed graph (called *digraph*), is a set of vertices V and a set of directionally oriented edges.

Examples



Here the set of vertices is $V = \{1,2,3\}$, and the set of edges is $E = \{(3,1),(2,1),(2,3),(3,3)\}$. The edge $(3,3)$ is called *self-loop*.

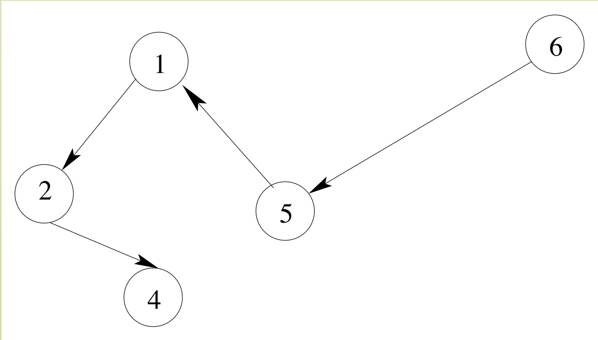
An *undirected graph* is a graph in which an edge between two vertices is not directionally oriented (there is no arrow).



Here the set of vertices is $V = \{1,2,3,4,5\}$, and the set of edges is
$E = \{(1,2),(2,1),(2,3),(3,2),(4,5),(5,4)\}$.

Examples (cont.)

In the directed graph, there exists a *directed path* of length n from the vertex I to J **if and only if** there is a sequence of vertices: $I = I_0, I_1, I_2, ..., I_n = J$ such that I_{k-1} is directly connected to I_k by an edge and $k = 1, 2, ..., n$.



On the graph above there is a directed path from vertex 6 to vertex 4 of length 4.

DEFINITIONS:

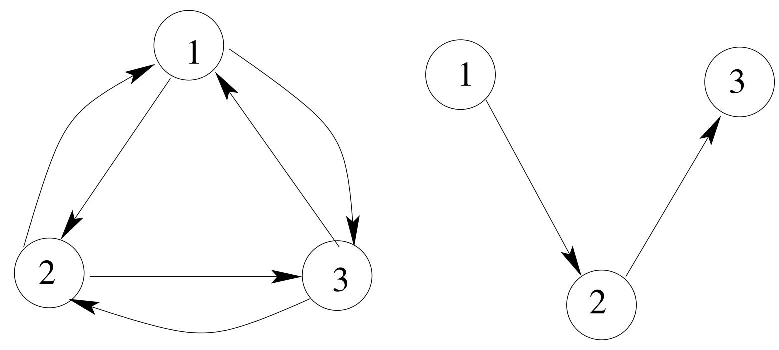
A path is a *simple path* if all vertices in the path are distinct.

A graph is *connected* if for any two vertices, there is a path from one vertex to the other. A graph is *disconnected* if it is not connected.

Definitions

A digraph is *strongly connected* if for any two vertices in the graph there is a directed path from I to J and a directed path from J to I.

A digraph is *weakly connected* if for any two vertices I and J there is a directed path from I to J or from J to I.



Strongly connected graph

Weakly connected graph

The *outdegree* of a vertex in a digraph is the number of vertices adjacent to it or edges leaving from the vertex.

The *indegree* of a vertex in a digraph is the number of edges entering the vertex.

A *sink* vertex is a vertex with outdegree equal to zero.

A *source* is a vertex with indegree equal to zero.

A *cycle* in a directed graph is a directed path of the length at least 1 which starts and terminates at the same vertex. A self-loop is a cycle of length 1.

An *acyclic* graph is a graph without cycles.

A graph is *connected* if for any two vertices, there is a path from one vertex to the other. A graph is *disconnected* if it is not connected.

Operations on a Graph

Six operations are defined for a graph:

- add a vertex - inserts a new vertex into graph without connecting it to any other vertex.
- delete a vertex - removes a vertex from a graph.
- add an edge - inserts an edge between a source and destination vertices in a graph.
- delete an edge - removes an edge connecting a source and destination vertices in a graph.
- find a vertex - traverses a graph looking for a specified vertex.
- traverse a graph - visits all vertices in the graph and processes them one by one. There are two standard graph traversals: *depth-first traversal* and *breadth-first traversal*. In the depth-first traversal, all of a vertex's descendents are processed before moving to an adjacent vertex. In the breadth-first traversal, all of the adjacent vertices are processed before processing the descendents of a vertex.

Representation of a Graph

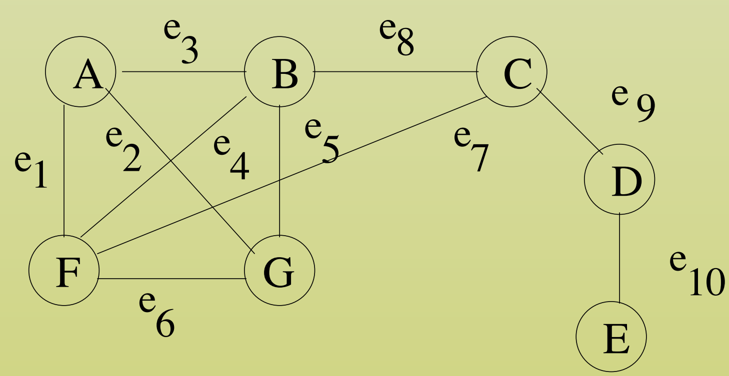
To represent a graph in a computer we need to store a set of vertices and a set of edges.

There are two common representations of graphs:

- adjacency matrix
- adjacency list

A vertex *v* is called *adjacent to u* if there is an edge from *u* to *v*.

EXAMPLE:



The adjacency matrix for this graph is a two-dimensional array of size 7 by 7, whose indices correspond to vertices according to the mapping function.

Adjacency Matrix

The mapping function is defined: the first row is a set of vertices and the second row is a set of indices of the array.

A	B	C	D	E	F	G	H
↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7

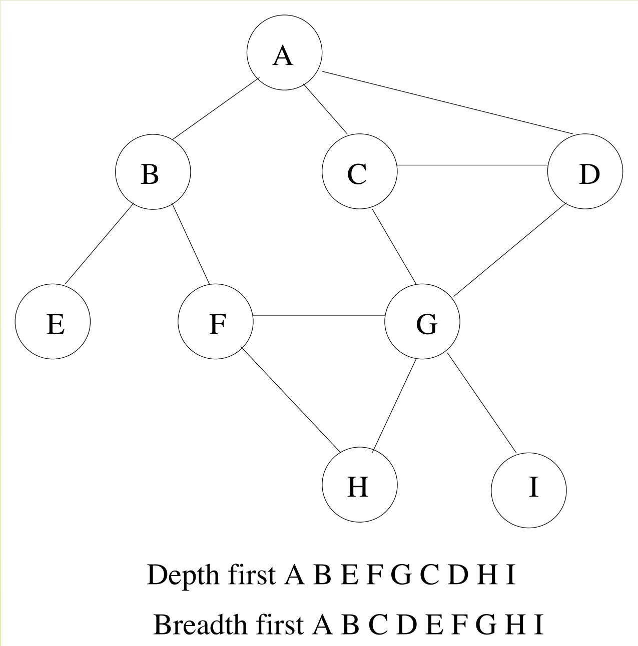
	A	B	C	D	E	F	G
A	0	1	0	0	0	1	1
B	1	0	1	0	0	1	1
C	0	1	0	1	0	1	0
D	0	0	1	0	1	0	0
E	0	0	0	1	0	0	0
F	1	1	1	0	0	0	1
G	1	1	0	0	0	1	0

Adjacency List

The adjacency list representation for the graph above

A → B → F → G
B → A → C → F → G
C → B → D → F
D → C → E
E → D
F → A → B → C → G
G → A → B → F

Graph Traversal



BFS(G, s)

```
initialize G by marking each
vertex
   as unvisited
enqueue(Q,s) and mark s as
visited
while (notEmpty(Q))
   u = dequeue(Q)
   for each v adjacent to u
      if (v is unvisited)
enqueue(Q,v)
          and mark v as visited
   end for

end while
```

Efficiency of BFS:

- $O(V^2)$ if graph is represented using adjacency matrix
- $O(V+E)$ if graph is represented using adjacency list

```
DFS(G,s)
initialize G by marking each vertex as unvisited
for each vertex u in V, call DFS-Visit(G,u)
DFS-Visit(G,u)
if(u is unvisited){
   mark vertex u as visited
   for each v adjacent to u
      if (v is unvisited)
         DFS-Visit(G,u) /*recursive call*/
   end for
}
```

Efficiency of DFS:

- $O(V^2)$ if graph is represented using adjacency matrix
- $O(V+E)$ if graph is represented using adjacency list

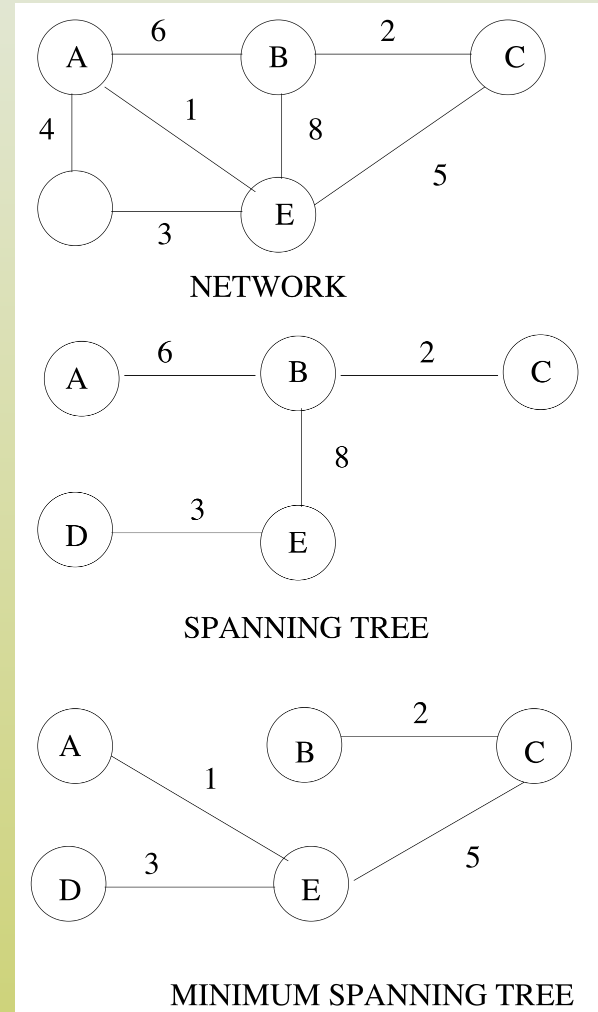
Network (Weighted Graph)

A *network* or *weighted graph* is a graph whose edges are weighted. The meaning of the weights depends on the application. Weights, for instance, can be the distance between two vertices (cities).

The weight of a *path* of length n from the vertex I to J is the sum of weights of all consecutive weights of edges on this path. Consider a sequence of vertices: $I = I_0, I_1, I_2, ..., I_n = J$ such that $I_{k-1}$is directly connected to I_k by an edge with a certain weight $w(I_{k-1}, I_k)$ and $k = 1, 2, ..., n$. Then the path weight can be expressed by the formula

$$w(I_0, I_n) = \sum_{k=1}^{n} w(I_{k-1}, I_k)$$

Minimum Spanning Tree



DEFINITION:

A *minimum spanning* tree is a tree that contains all the vertices in the graph and the total weight of edges is the minimum.

Network Problems

- Find the shortest path between two vertices (the Dijkstra's algorithm)
- Find the shortest path between all pairs (every pair) of vertices (the Floyd's algorithm)
- Find the shortest path from a given vertex to every other vertex in the graph

EXAMPLE: Find the shortest path from the vertex 1 to all other vertices in the graph.