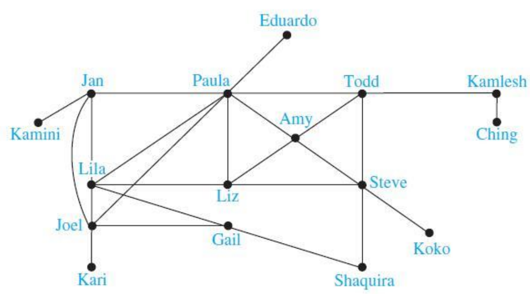
A Quick Review of Graphs

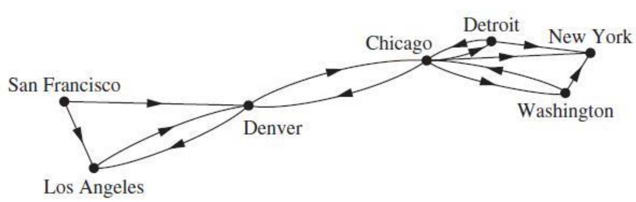
- A **graph** is a tuple of two: $G = (V, E)$, where V is the set of vertices and E is the set of edges.
  - In other words, as an object, a graph has two attributes: a set of vertices and a set of edges.
  - The vertices are usually the objects that we want to study, and edges are the relations between two objects.

1. For example, when we want to study the relationship between people in a group, we can mode this group of people into a graph like this:



This is an **undirected graph** (in which edges do not have directions) since the acquaintanceship between two people is mutual.

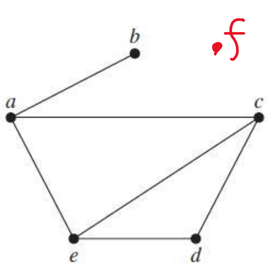
When we want to study the flight trips between cities, we can model the cities into a graph like this:



This is a **directed graph** (in which edges have directions) since the existence of a flight from city A to B doesn't guarantee the existence of a flight from city B to A.

- There are two common ways to represent a graph: One way is to use a set of LinkedList to list each vertex's neighbors; these LinkedList are called **adjacency lists**. Another way to represent a graph is to use an **adjacency matrix**.

2. Given the following graph, we can represent it using adjacency lists or an adjacency matrix as follows.

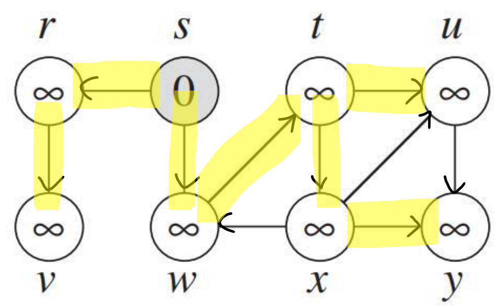


a → b → e → c
b → a
c → a → e → d
e → c → d → a
d → c → e
f → ⊠

	a	b	c	d	e	f
a	0	1	1	0	1	
b	1	0	0	0	0	
c	1	0	0	1	1	
d	0	0	1	0	1	
e	1	0	1	1	0	
f						

- **Breadth First Search** can find all the shortest paths from a single vertex to all other vertices in an (unweighted) graph.

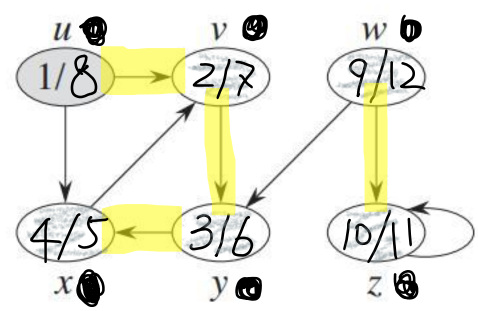
```
BFS (G, s)
for each vertex u ∈ G.V − {s}
        u.color = WHITE // white means unvisited
        u.d = ∞ // d means the distance from s to u
        u.π = NIL  // u.\pi is the predecessor of u
s.color = GRAY // gray means being visited or in
s.d = 0                    //queue
s.π = NIL
Q = ∅             // Q is a queue of vertices
Enqueue (Q, s)
while Q ≠ ∅
        u = Dequeue (Q)            // u is being visited. u = s in the first iteration.
        for each v ∈ G.Adj[u]      // the adjacency list of u
                if v.color = WHITE        //find an unvisited neighbor
                        v.color = GRAY
                        v.d = u.d + 1
                        v.π = u
                        Enqueue (Q, v)
  u.color = BLACK      // finished visiting
```



More about Graphs

- **Depth First Search** (DFS) is another graph searching algorithm. Unlike BFS, DFS search "deeper" in the graph whenever possible.

```
DFS (G) //no starting vertex
for each vertex u ∈ G.V
        u.color = WHITE //white means unvisited
        u.π = NIL //\pi is predecessor
time = 0
for each vertex u ∈ G.V
        if u.color = WHITE
                DFS-VISIT (G, u)
```



(((( )))) (( ))
u v y x x y v u w z z w
1 2 3 4 ....
12

```
DFS-VISIT (G, u)
time = time + 1
u.d = time //d is the discover time
u.color = GRAY //Gray means being visited
for each v ∈ G.Adj[u]
        if v.color = WHITE //if a neighbor unvisited
                v.π = u
                DFS-VISIT (G, v) //recursive call DFS-VISIT from the unvisited neighbor
u.color = BLACK // black means finished visiting
time = time + 1
```

$u.f = time$ // f is the finishing time