ALGORITHMS AND DATA STRUCTURES LECTURE 8 – GRAPHS (PART I)

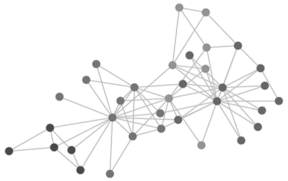
Askar Khaimuldin

askar.khaimuldin@astanait.edu.kz

ASTANA IT UNIVERSITY

CONTENT

1. What is a Graph?
2. Applications
3. Graph API
4. Vertex
5. Edge
6. Glossary



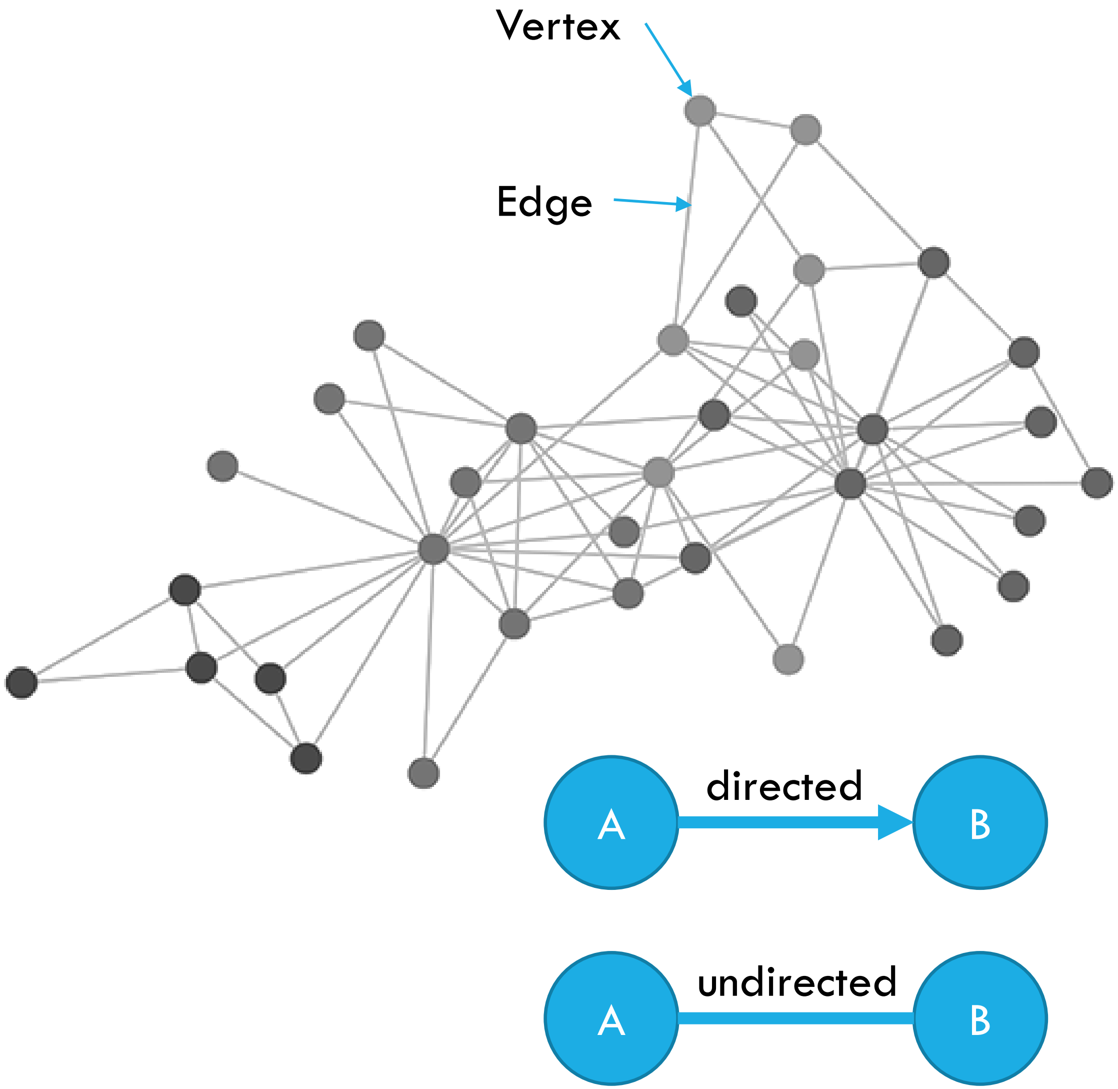
WHAT IS A GRAPH?

A **graph** is a set of **vertices** and a collection of **edges** that each connect a pair of vertices

- Vertex – node
- Edge – connection between 2 nodes

Types:

- undirected graphs (with simple connections)
- digraphs (where the direction of each connection is significant)
- edge-weighted graphs (where each connection has an associated weight)
- edge-weighted digraphs (where each connection has both a direction and a weight)



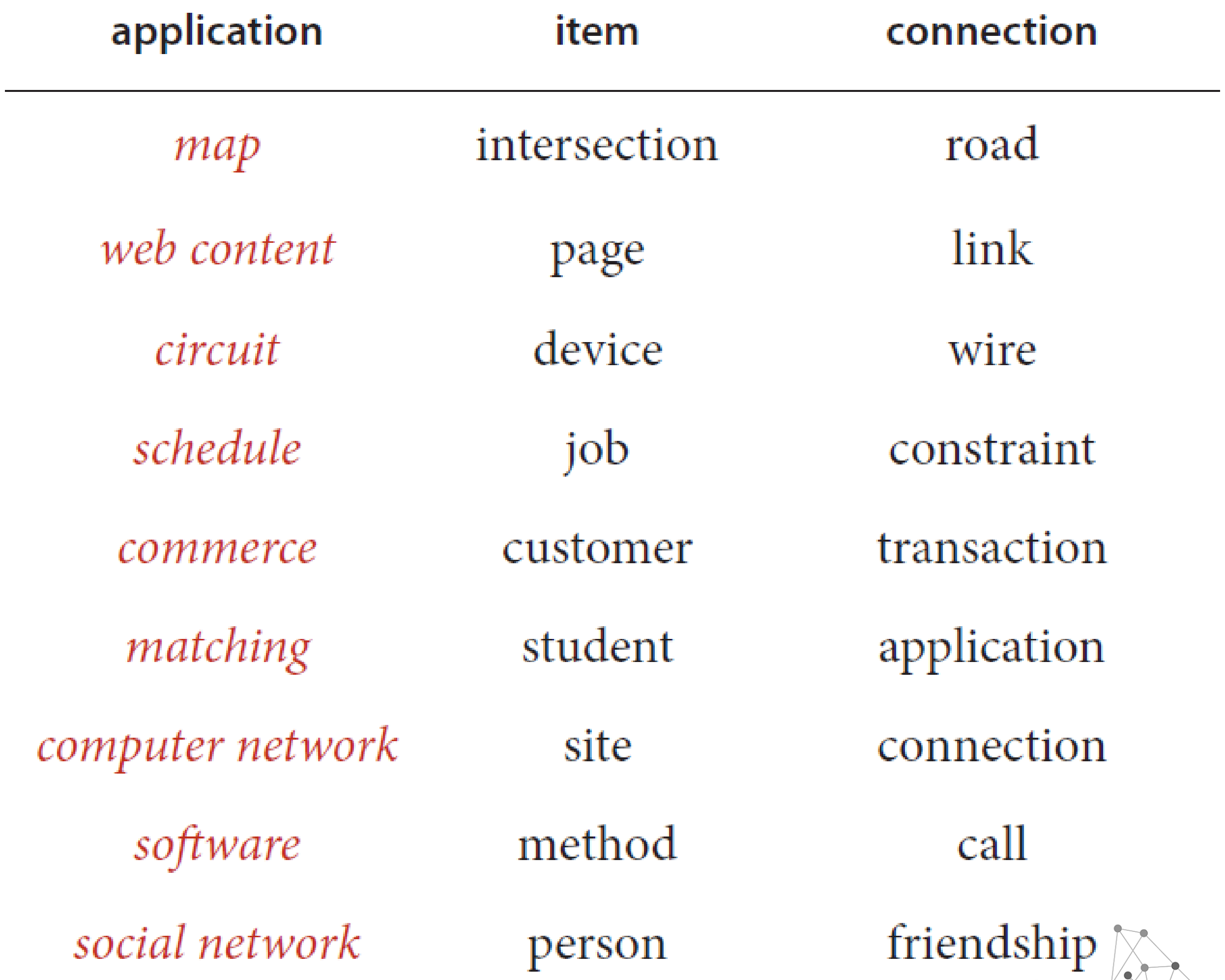
APPLICATIONS

Graph theory, a major branch of mathematics

It has been studied intensively for hundreds of years

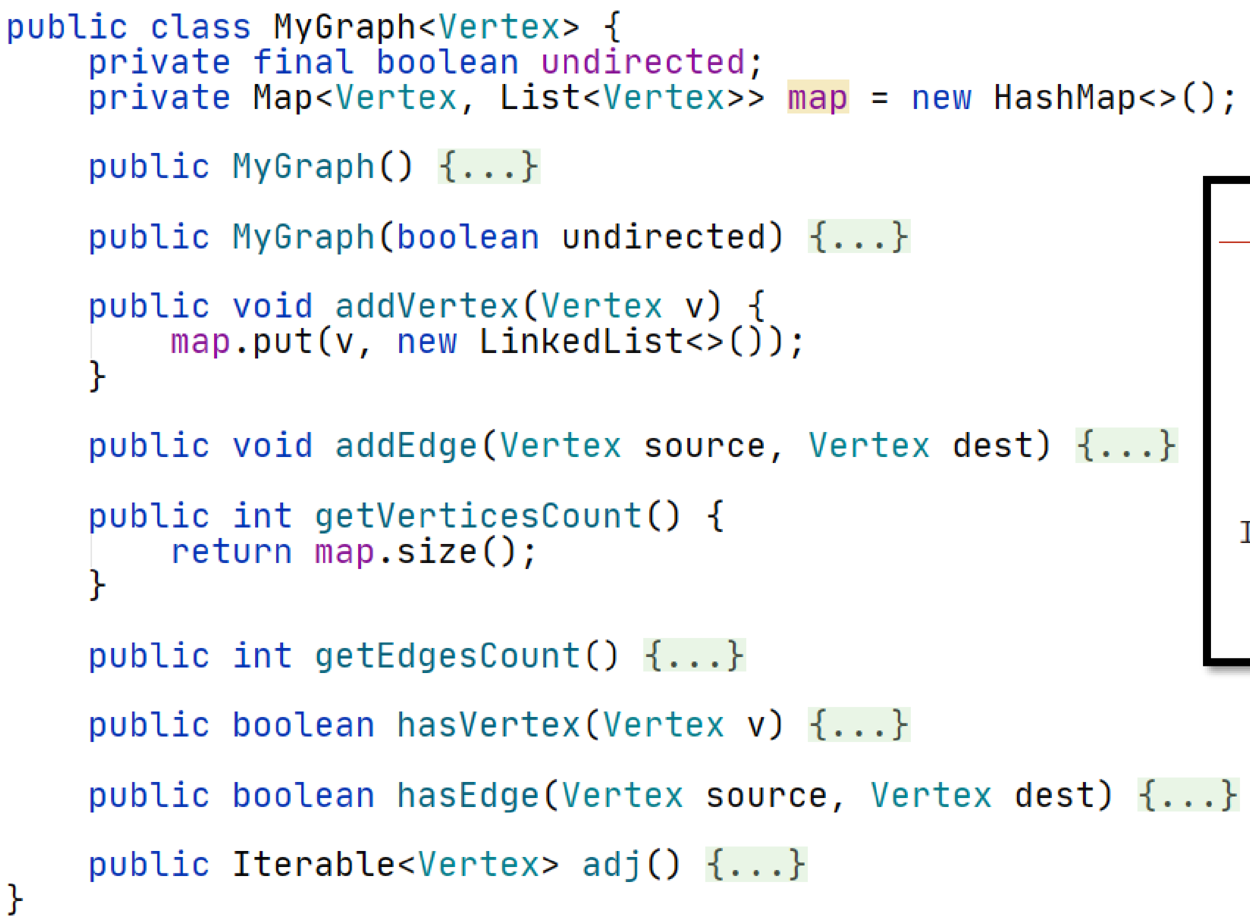
The range of applications for which graphs are the appropriate abstraction (i.e. facebook friends)

A "binary tree" is a special case of a directed graph

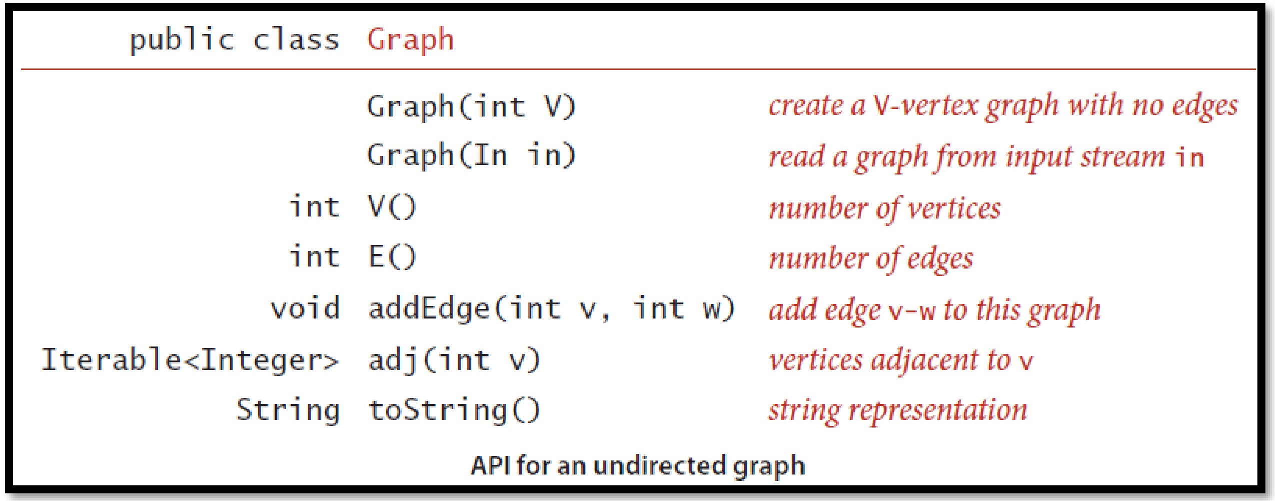


application	item	connection
map	intersection	road
web content	page	link
circuit	device	wire
schedule	job	constraint
commerce	customer	transaction
matching	student	application
computer network	site	connection
software	method	call
social network	person	friendship

GRAPH API



```java
public class MyGraph<Vertex> {
    private final boolean undirected;
    private Map<Vertex, List<Vertex>> map = new HashMap<>();

    public MyGraph() {...}

    public MyGraph(boolean undirected) {...}

    public void addVertex(Vertex v) {
        map.put(v, new LinkedList<>());
    }

    public void addEdge(Vertex source, Vertex dest) {...}

    public int getVerticesCount() {
        return map.size();
    }

    public int getEdgesCount() {...}

    public boolean hasVertex(Vertex v) {...}

    public boolean hasEdge(Vertex source, Vertex dest) {...}

    public Iterable<Vertex> adj() {...}
}
```



public class	Graph	
	Graph(int V)	*create a V-vertex graph with no edges*
	Graph(In in)	*read a graph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v-w *to this graph*
Iterable<Integer>	adj(int v)	*vertices adjacent to* v
String	toString()	*string representation*

API for an undirected graph

VERTEX

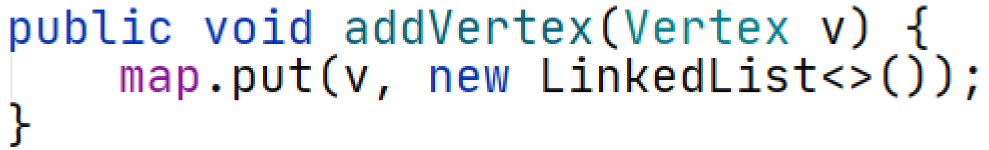
Vertex is a node that holds a single item from the collection

One vertex could contain a set of vertices **adjacent** to it

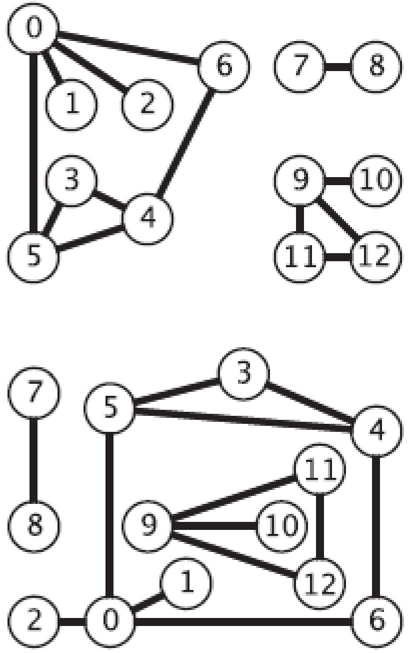
The vertex is created to the graph by adding it to **the array of adjacency lists (map)**

Each vertex has a list of adjacent vertices

*When there is an edge connecting two vertices, we say that the vertices are **adjacent** to one another and that the edge is **incident** to both vertices*

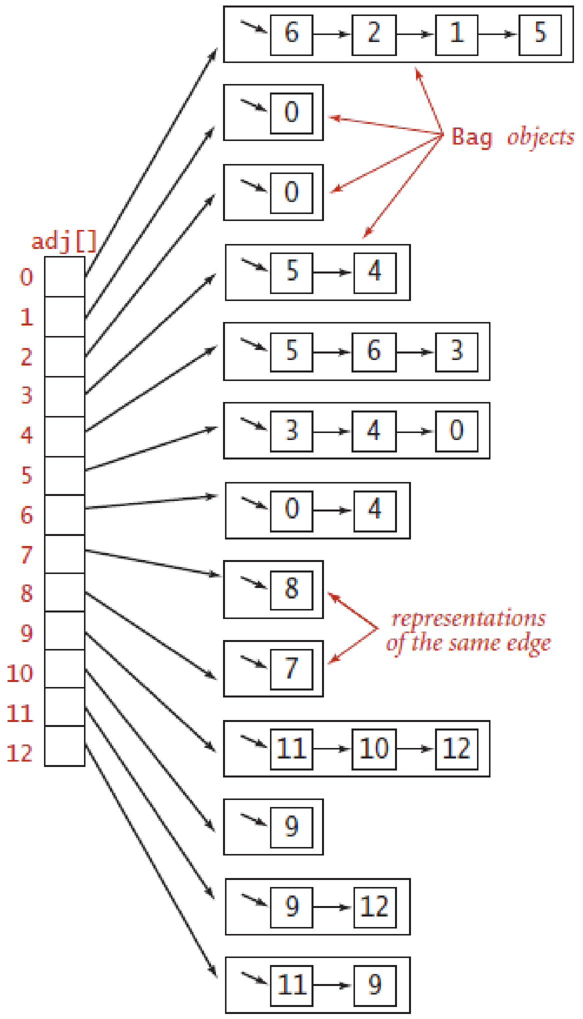


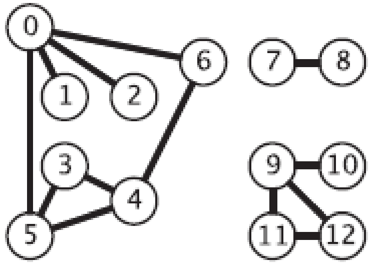
```
public void addVertex(Vertex v) {
    map.put(v, new LinkedList<>());
}
```



Two drawings of the same graph

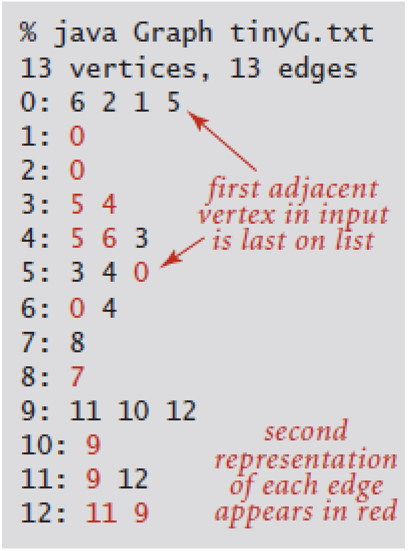
THE ARRAY OF ADJACENCY LISTS





tinyG.txt

```
V → 13
    13 ← E
    0 5
    4 3
    0 1
    9 12
    6 4
    5 4
    0 2
    11 12
    9 10
    0 6
    7 8
    9 11
    5 3
```



```
% java Graph tinyG.txt
13 vertices, 13 edges
0: 6 2 1 5
1: 0
2: 0
3: 5 4
4: 5 6 3
5: 3 4 0
6: 0 4
7: 8
8: 7
9: 11 10 12
10: 9
11: 9 12
12: 11 9
```

first adjacent vertex in input is last on list

second representation of each edge appears in red

EDGE

Edge is a connection between two vertices

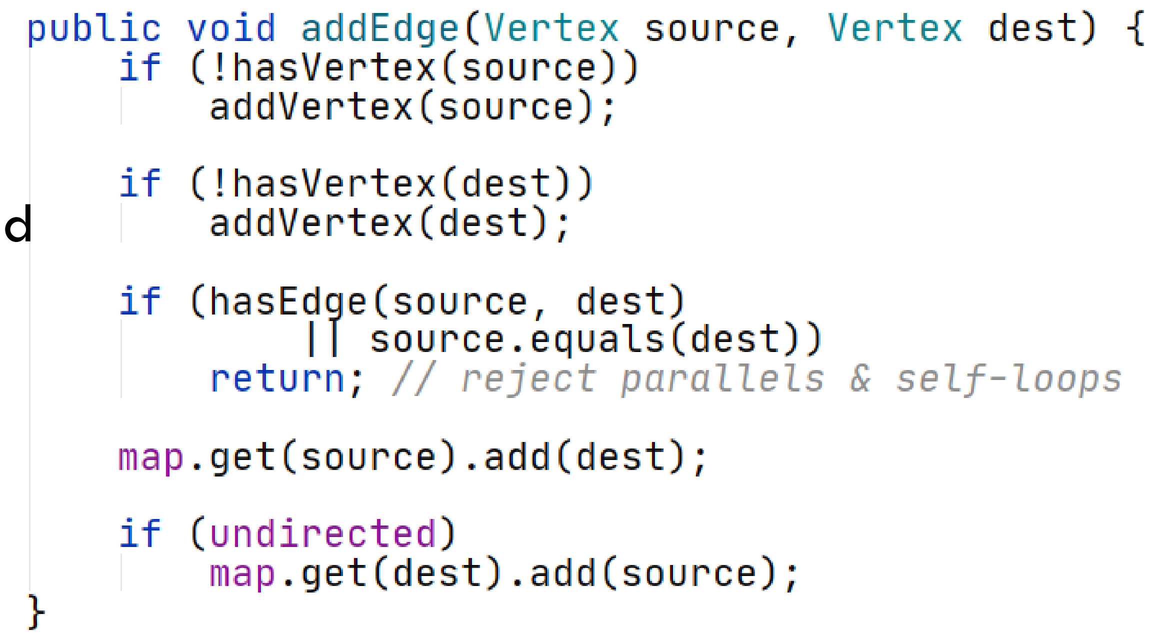
To create an edge from A to B, it is needed to add vertex B to the adjacency list of vertex A

- Both A and B must exist

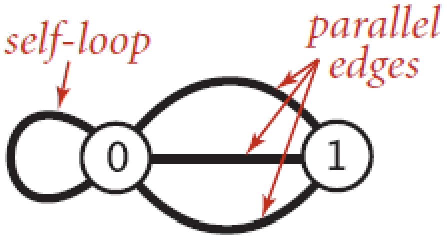
If a graph is **undirected**, the vertex A should also be added to the adjacency list of the vertex B

- This creates connections from A to B and vice versa

*The **degree** of a vertex is the number of edges incident to it*



```java
public void addEdge(Vertex source, Vertex dest) {
    if (!hasVertex(source))
        addVertex(source);

    if (!hasVertex(dest))
        addVertex(dest);

    if (hasEdge(source, dest)
            || source.equals(dest))
        return; // reject parallels & self-loops

    map.get(source).add(dest);

    if (undirected)
        map.get(dest).add(source);
}
```



Anomalies

GLOSSARY

A **path** in a graph is a sequence of vertices connected by edges

A **cycle** is a path with at least one edge whose first and last vertices are the same

The **length** of a path or a cycle is its number of edges.

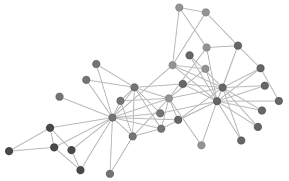
An **acyclic graph** is a graph with no cycles

A **tree** is an acyclic connected graph

A disjoint set of trees is called a **forest**

The **density** of a graph is the proportion of possible pairs of vertices that are connected by edges

A graph is **connected** if there is a path from every vertex to every other vertex in the graph



LITERATURE

Algorithms, 4th Edition, by Robert Sedgewick and Kevin Wayne, Addison-Wesley

- Chapter 4

GOOD LUCK!