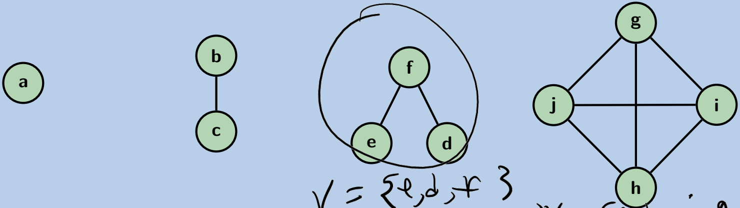


$V = \{a\}$, $E = \varnothing$

$V = \{b,c\}$, $E = \{\{b,c\}\}$

$V = \{e,d,f\}$

$E = \{\{e,f\}\{f,d\}\}$

$V = \{j,h,i,g\}$

$E = (V \times V) \setminus \{\{x\} \mid x \in V\}$

We call the circles **vertices** and the lines **edges**.

Definition (Graph)

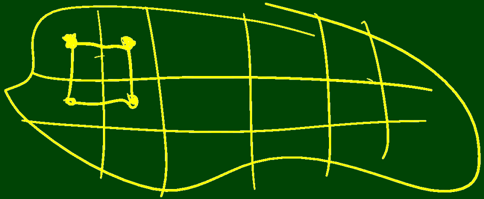
A **Graph** is a pair, $G = (V,E)$, where:

- V is a set of **vertices**, and
- E is a set of **edges** (pairs of vertices).

To model a problem with a graph, you need to make two choices

1. What are the vertices?
2. What are the edges?

- Maps
- The Internet
- Social Networks
- A Running Program
- A Chess Game
- Telephone Lines
- CSE Courses



With these in mind, let's talk about more crucial definitions.

To model a problem with a graph, you need to make two choices

1. What are the vertices?
2. What are the edges?

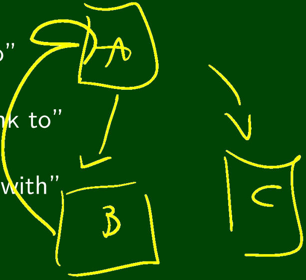
- Maps

  Vertices: regions; Edges: "is next to"
- The Internet

  Vertices: websites; Edges: "has a link to"
- Social Networks

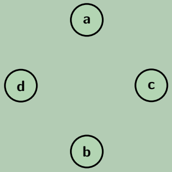
  Vertices: people; Edges: "is friends with"
- A Running Program

  Vertices: methods; Edges: "calls"
- A Chess Game
- Telephone Lines
- CSE Courses



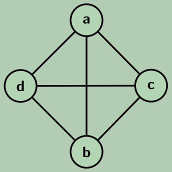
With these in mind, let's talk about more crucial definitions.

Empty Graph



…

Complete Graph (K_n)



Some Questions

- How many edges can a graph with $|V| = n$ have?

$$\binom{n}{2} = \frac{n(n-1)}{2} \in O(n^2)$$

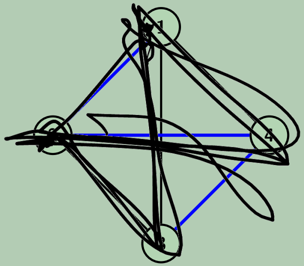
- If we have $|E| = n$, what is the smallest number of vertices we can have? The largest?
  - Smallest:
  - ~~Largest:~~

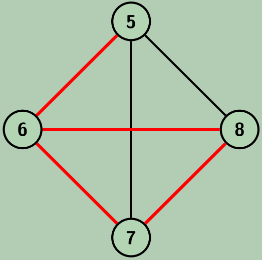
Definition (Walk)

A **walk** in a graph $G = (V, E)$ is a list of vertices:
$v_0, v_1, \dots, v_n$ such that $\{v_i, v_{i+1}\} \in E$.
Intuitively, a path from u to v is a continuous line drawn without picking up your pencil.

Definition (Path)

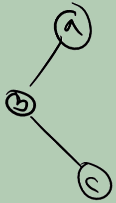
A **path** in a graph $G = (V, E)$ is a walk with no repeated vertices.



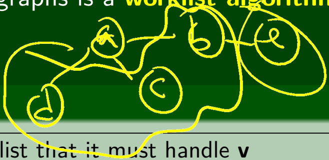


Definition (Connected Graph)

We say a graph is connected if for every pair of vertices, $u, v \in V$, there is a path from u to v.



A very common type of algorithm on graphs is a **worklist algorithm**.



Recall the `WorkList` ADT:

`WorkList` ADT

`add(`**`v`**`)`	Notifies the worklist that it must handle **v**
`next()`	Returns the next vertex to work on
`hasWork()`	Returns true if there’s any work left and false otherwise

Importantly, we **do not care how** the worklist manages the work.
(Okay, we do, but not when coming up with the algorithm.)

Worklist algorithms will always look like the following:

```
worklist = /* add initial work to worklist */
while (worklist.hasWork()) {
   v = worklist.next();
   doWork(v);
}
```