Graphs
What’s a Graph?

A bunch of vertices connected by edges.
Why Graph Algorithms?

• They’re fun.
• They’re interesting.
• They have surprisingly many applications.
Graphs are Everywhere
Adjacency as a Graph

Each vertex represents a state, country, etc.

There is an edge between two vertices if the corresponding areas share a border.
When a Graph?

Graphs are a good representation for any collection of objects and binary relation among them:

- The relationship in space of places or objects
- The ordering in time of events or activities
- Family relationships
- Taxonomy (e.g. animal - mammal - dog)
- Precedence (x must come before y)
- Conflict (x conflicts or is incompatible with y)
- Etc.
Our Menu

Depth-First Search
- Connected components
- Cycle detection
- Topological sort

Minimal Spanning Tree
- Kruskal’s
- Prim’s

Single-Source Shortest Paths
- Dijkstra’s
- Bellman-Ford
- DAG-SSSP

All-Pairs Shortest Paths
- Floyd-Warshall
- Johnson’s
Basic Concepts

A *graph* is an ordered pair \((V, E)\).

\(V\) is the set of vertices. (You can think of them as integers 1, 2, \ldots, n.)

\(E\) is the set of edges. An edge is a pair of vertices: \((u, v)\).

Note: since \(E\) is a set, there is at most one edge between two vertices. (*Hypergraphs* permit multiple edges.)

Edges can be labeled with a *weight*:

\[\begin{array}{c}
\text{10}
\end{array}\]
Concepts: Directedness

In a *directed* graph, the edges are “one-way.” So an edge \((u, v)\) means you can go from \(u\) to \(v\), but not vice versa.

In an *undirected* graph, there is no direction on the edges: you can go either way. (Also, no self-loops.)
Concepts: Adjacency

Two vertices are *adjacent* if there is an edge between them.

For a directed graph, $u$ is adjacent to $v$ iff there is an edge $(v, u)$.

- $u$ is adjacent to $v$.
- $v$ is adjacent to $u$ and $w$.
- $w$ is adjacent to $v$.

- $u$ is adjacent to $v$.
- $v$ is adjacent to $w$. 
Concepts: Degree

Undirected graph: The *degree* of a vertex is the number of edges touching it.

For a directed graph, the *in-degree* is the number of edges entering the vertex, and the *out-degree* is the number leaving it. The *degree* is the *in-degree* + the *out-degree*. 
A path is a sequence of adjacent vertices. The length of a path is the number of edges it contains, i.e. one less than the number of vertices.

Is there a path from 1 to 4? What is its length? What about from 4 to 1? How many paths are there from 2 to 3? From 2 to 2? From 1 to 1?

We write $u \Rightarrow v$ if there is path from $u$ to $v$. (The correct symbol, a wiggly arrow, is not available in standard fonts.) We say $v$ is reachable from $u$.
A cycle is a path of length at least 1 from a vertex to itself.

A graph with no cycles is acyclic.

A path with no cycles is a simple path.

The path <2, 3, 4, 2> is a cycle.
An undirected graph is *connected* iff there is a path between any two vertices.

The adjacency graph of U.S. states has three connected components. Name them.

(We say a directed graph is *strongly connected* iff there is a path between any two vertices.)
A *free tree* is a connected, acyclic, undirected graph.

To get a *rooted tree* (the kind we’ve used up until now), designate some vertex as the root.

If the graph is disconnected, it’s a *forest*.

**Facts about free trees:**

- \(|E| = |V| - 1\)
- Any two vertices are connected by exactly one path.
- Removing an edge disconnects the graph.
- Adding an edge results in a cycle.
Graph Size

We describe the time and space complexity of graph algorithms in terms of the number of vertices, \(|V|\), and the number of edges, \(|E|\).

\(|E|\) can range from 0 (a totally disconnected graph) to \(|V|^2\) (a directed graph with every possible edge, including self-loops).

Because the vertical bars get in the way, we drop them most of the time. E.g. we write \(\Theta(V + E)\) instead of \(\Theta(|V| + |E|)\).
Representing Graphs

Adjacency matrix: if there is an edge from vertex i to j, $a_{ij} = 1$; else, $a_{ij} = 0$.

Space: $\Theta(V^2)$

Adjacency list: Adj[v] lists the vertices adjacent to v.

Space: $\Theta(V+E)$

Represent an undirected graph by a directed one:

```
1 2 3 4
1 [0 1 0 1]
2 [0 0 1 0]
3 [0 0 0 1]
4 [0 1 0 0]
```

Adj:
Depth-First Search

A way to “explore” a graph. Useful in several algorithms.

Remember preorder traversal of a binary tree?

Binary-Preorder(x):
1  number x
2  Binary-Preorder(left[x])
3  Binary-Preorder(right[x])

Can easily be generalized to trees whose nodes have any number of children.

This is the basis of depth-first search. We “go deep.”
DFS on Graphs

The wrong way:

Bad-DFS(u)
1 number u
2 for each v in Adj[u] do
3 Bad-DFS(v)

What’s the problem?
Fixing Bad-DFS

We’ve got to indicate when a node has been visited.

Following CLRS, we’ll use a color:

- **WHITE** never seen
- **GRAY** discovered but not finished (still exploring its descendants)
- **BLACK** finished
A Better DFS

- initially, all vertices are WHITE

Better-DFS(u)

    color[u] ← GRAY

    number u with a “discovery time”

    for each v in Adj[u] do
        if color[v] = WHITE then ▶ avoid looping!
            Better-DFS(v)

    color[u] ← BLACK

    number u with a “finishing time”
Depth-First Spanning Tree

As we’ll see, DFS creates a tree as it explores the graph. Let’s keep track of the tree as follows (actually it creates a forest not a tree):

When \( v \) is explored directly from \( u \), we will make \( u \) the parent of \( v \), by setting the predecessor, aka, parent \( (\pi) \) field of \( v \) to \( u \):

\[
\pi[v] \leftarrow u
\]
Two More Ideas

1. We will number each vertex with discovery and finishing times—these will be useful later. The “time” is just a unique, increasing number. The book calls these fields $d[u]$ and $f[u]$.

2. The recursive routine we’ve written will only explore a connected component. We will wrap it in another routine to make sure we explore the entire graph.
DFS($G$)
1. for each vertex $u \in V[G]$
2. \hspace{1em} do $\text{color}[u] \leftarrow \text{WHITE}$
3. \hspace{1em} $\pi[u] \leftarrow \text{NIL}$
4. $\text{time} \leftarrow 0$
5. for each vertex $u \in V[G]$
6. \hspace{1em} do if $\text{color}[u] = \text{WHITE}$
7. \hspace{2em} then DFS-VISIT($u$)

DFS-VISIT($u$)
1. $\text{color}[u] \leftarrow \text{GRAY}$ \hspace{1em} $\triangleright$ White vertex $u$ has just been discovered.
2. $\text{time} \leftarrow \text{time} + 1$
3. $d[u] \leftarrow \text{time}$
4. for each $v \in \text{Adj}[u]$ \hspace{1em} $\triangleright$ Explore edge $(u, v)$.
5. \hspace{1em} do if $\text{color}[v] = \text{WHITE}$
6. \hspace{2em} then $\pi[v] \leftarrow u$
7. \hspace{2em} DFS-VISIT($v$)
8. $\text{color}[u] \leftarrow \text{BLACK}$ \hspace{1em} $\triangleright$ Blacken $u$; it is finished.
9. $f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$
graphs from p. 1081

(a)

(b)

(c)