Graphs

- A graph is a formalism for representing relationships among items
  - Very general definition because very general concept

- A graph is a pair
  \[ G = (V, E) \]
  - A set of vertices, also known as nodes
    \[ V = \{v_1, v_2, \ldots, v_n\} \]
  - A set of edges
    \[ E = \{e_1, e_2, \ldots, e_m\} \]
    - Each edge \( e_i \) is a pair of vertices
      \( (v_j, v_k) \)
    - An edge “connects” the vertices

- Graphs can be directed or undirected
An ADT?

• Can think of graphs as an ADT with operations like \( \text{isEdge}((v_j, v_k)) \)

• But it is unclear what the “standard operations” are

• Instead we tend to develop algorithms over graphs and then use data structures that are efficient for those algorithms

• Many important problems can be solved by:
  1. Formulating them in terms of graphs
  2. Applying a standard graph algorithm

• To make the formulation easy and standard, we have a lot of standard terminology for graphs
Some Graphs

For each, what are the vertices and what are the edges?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps
- Airline routes
- Family trees
- Course pre-requisites
- ...

Core algorithms that work across such domains is why we are CSE
Undirected Graphs

• In undirected graphs, edges have no specific direction
  – Edges are always “two-way”

• Thus, \((u, v) \in E\) implies \((v, u) \in E\).
  – Only one of these edges needs to be in the set
  – The other is implicit, so normalize how you check for it

• Degree of a vertex: number of edges containing that vertex
  – Put another way: the number of adjacent vertices
Directed Graphs

- In directed graphs (a.k.a. digraphs), edges have a direction.

Let \((u, v) \in E\) mean \(u \rightarrow v\).
- Call \(u\) the source and \(v\) the destination.

Thus, \((u, v) \in E\) does not imply \((v, u) \in E\).
- In-Degree of a vertex: number of in-bound edges, i.e., edges where the vertex is the destination.
- Out-Degree of a vertex: number of out-bound edges, i.e., edges where the vertex is the source.
Self-Edges, Connectedness

• A self-edge a.k.a. a loop edge is of the form \((u,u)\)
  – Depending on the use/algorithim, a graph may have:
    • No self edges
    • Some self edges
    • All self edges (often therefore implicit, but we will be explicit)

• A node can have a degree / in-degree / out-degree of zero

• A graph does not have to be connected
  – Even if every node has non-zero degree
  – More discussion of this to come
More Notation

For a graph $G = (V, E)$:

- $|V|$ is the number of vertices
- $|E|$ is the number of edges
  - Minimum?
  - Maximum for undirected?
  - Maximum for directed?

- If $(u, v) \in E$
  - Then $v$ is a neighbor of $u$ (i.e., $v$ is adjacent to $u$)
  - Order matters for directed edges
    - $u$ is not adjacent to $v$ unless $(v, u) \in E$

$V = \{A, B, C, D\}$
$E = \{(C, B), (A, B), (B, A), (C, D)\}$
More Notation

For a graph $G = (V, E)$:

- $|V|$ is the number of vertices
- $|E|$ is the number of edges
  - Minimum? 0
  - Maximum for undirected? $|V| (|V| + 1) / 2 \in O(|V|^2)$
  - Maximum for directed? $|V|^2 \in O(|V|^2)$
    (assuming self-edges allowed, else subtract $|V|$)
- If $(u, v) \in E$
  - Then $v$ is a neighbor of $u$ (i.e., $v$ is adjacent to $u$)
  - Order matters for directed edges
    - $u$ is not adjacent to $v$ unless $(v, u) \in E$
Examples again

Which would use directed edges?
Which would have self-edges?
Which could have 0-degree nodes?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps
- Airline routes
- Family trees
- Course pre-requisites
- …
Weighted Graphs

- In a weighted graph, each edge has a weight a.k.a. cost
  - Typically numeric (our examples use ints, but not required)
  - Orthogonal to whether graph is directed
  - Some graphs allow negative weights; many do not

Diagram:

- Clinton to Mukilteo: 20
- Kingston to Edmonds: 30
- Bainbridge to Seattle: 35
- Bremerton to Seattle: 60
Examples

What, if anything, might **weights** represent for each of these? Do **negative weights** make sense?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps
- Airline routes
- Family trees
- Course pre-requisites
- ...
**Paths and Cycles**

- A path is a list of vertices \([v_0, v_1, ..., v_n]\) such that \((v_i, v_{i+1}) \in E\) for all \(0 \leq i < n\). We say “a path from \(v_0\) to \(v_n\)”

- A cycle is a path that begins and ends at the same node \((v_0 == v_n)\)

Example path (that also happens to be a cycle):

[Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]
**Path Length and Cost**

- **Path length**: Number of edges in a path
- **Path cost**: Sum of the weights of each edge

Example where

\[ P = \text{[Seattle, Salt Lake City, Chicago, Dallas, San Francisco, Seattle]} \]

\[
\begin{align*}
\text{length}(P) &= 5 \\
\text{cost}(P) &= 11.5
\end{align*}
\]

Length can sometimes be called “unweighted cost”
Simple Paths and Cycles

• A simple path repeats no vertices, (except the first might be the last):
  [Seattle, Salt Lake City, San Francisco, Dallas]
  [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]

• Recall, a cycle is a path that ends where it begins:
  [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]
  [Seattle, Salt Lake City, Seattle, Dallas, Seattle]

• A simple cycle is a cycle and a simple path:
  [Seattle, Salt Lake City, San Francisco, Dallas, Seattle]
Paths and Cycles in Directed Graphs

Example:

Is there a path from A to D? No

Does the graph contain any cycles? No
Undirected Graph Connectivity

- An undirected graph is **connected** if for all pairs of vertices $u, v$, there exists a *path* from $u$ to $v$.

![Connected graph](image1)

![Disconnected graph](image2)

- An undirected graph is **complete**, a.k.a. *fully connected*, if for all pairs of vertices $u, v$, there exists an *edge* from $u$ to $v$.

![Complete graph](image3)
Directed Graph Connectivity

• A directed graph is **strongly connected** if there is a path from every vertex to every other vertex.

• A directed graph is **weakly connected** if there is a path from every vertex to every other vertex *ignoring direction of edges*.

• A direct graph is **complete**, a.k.a. **fully connected**, if for all pairs of vertices $u, v$, there exists an *edge* from $u$ to $v$. 
Examples

For undirected graphs: connected?
For directed graphs: strongly connected? weakly connected?

- Web pages with links
- Facebook friends
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Road maps (e.g., Google maps)
- Airline routes
- Family trees
- Course pre-requisites
- …
Trees as Graphs

When talking about graphs, we say a tree is a graph that is:

- undirected
- acyclic
- connected

So all trees are graphs, but not all graphs are trees.

How does this relate to the trees we know and love?

Example:
Rooted Trees

• We are more accustomed to rooted trees where:
  – We identify a unique root
  – We think of edges as directed: parent to children

• Given a tree, picking a root gives a unique rooted tree
  – The tree is simply drawn differently and with undirected edges
**Rooted Trees**

- We are more accustomed to **rooted trees** where:
  - We identify a unique root
  - We think of edges as directed: parent to children
- Given a tree, picking a root gives a unique rooted tree
  - The tree is simply drawn differently and with undirected edges

```
D  E
B   
A   
C   
F

redrawn

F
G  H  C
A
B
D  E
```
Directed Acyclic Graphs (DAGs)

- A **DAG** is a directed graph with no directed cycles
  - Every rooted directed tree is a DAG
  - But not every DAG is a rooted directed tree

- Every DAG is a directed graph
  - But not every directed graph is a DAG
Examples

Which of our directed-graph examples do you expect to be a DAG?

- Web pages with links
- “Input data” for the Kevin Bacon game
- Methods in a program that call each other
- Airline routes
- Family trees
- Course pre-requisites
- ...
**Density / Sparsity**

- Recall: In an undirected graph, $0 \leq |E| < |V|^2$
- Recall: In a directed graph, $0 \leq |E| \leq |V|^2$
- So for any graph, $|E|$ is $O(|V|^2)$
- Another fact: If an undirected graph is *connected*, then $|E| \geq |V|-1$
- Because $|E|$ is often much smaller than its maximum size, we do not always approximate as $|E|$ as $O(|V|^2)$
  - This is a correct bound, it just is often not tight
  - If it is tight (i.e., $|E|$ is $\Theta(|V|^2)$), we say the graph is *dense*
    - More sloppily, dense means “lots of edges”
  - If $|E|$ is $O(|V|)$ we say the graph is *sparse*
    - More sloppily, sparse means “most possible edges missing”
What’s the Data Structure?

• So graphs are really useful for lots of data and questions
  – For example, “what’s the lowest-cost path from x to y”

• But we need a data structure that represents graphs

• Which data structure is “best” can depend on:
  – properties of the graph
    (e.g., dense versus sparse)
  – the common queries about the graph
    (e.g., “is (u, v) an edge?” vs “what are the neighbors of node u?”)

• So we will discuss the two standard graph representations
  – Adjacency Matrix and Adjacency List
  – Different trade-offs, particularly time versus space
Adjacency Matrix

- Assign each node a number from 0 to \(|V| - 1\)
- A \(|V| \times |V|\) matrix of Booleans (or 0 vs. 1)
  - Then \(M[u][v] == \text{true}\)
    means there is an edge from \(u\) to \(v\)
Adjacency Matrix Properties

- Running time to:
  - Get a vertex’s out-edges:
  - Get a vertex’s in-edges:
  - Decide if some edge exists:
  - Insert an edge:
  - Delete an edge:

- Space requirements:

- Best for sparse or dense graphs?
## Adjacency Matrix Properties

- Running time to:
  - Get a vertex’s out-edges: $O(|V|)$
  - Get a vertex’s in-edges: $O(|V|)$
  - Decide if some edge exists: $O(1)$
  - Insert an edge: $O(1)$
  - Delete an edge: $O(1)$

- Space requirements:
  - $|V|^2$ bits

- Best for sparse or dense graphs?
  - Best for dense graphs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>


**Adjacency Matrix Properties**

- How will the adjacency matrix vary for an undirected graph?
  - Undirected will be symmetric about diagonal axis

- How can we adapt the representation for weighted graphs?
  - Instead of a Boolean, store an number in each cell
  - Need some value to represent ‘not an edge’
    - 0, -1, or some other value based on how you are using the graph

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
```
Adjacency List

- Assign each node a number from 0 to $|V| - 1$
- An array of length $|V|$ in which each entry stores a list of all adjacent vertices (e.g., linked list)
**Adjacency List Properties**

- Running time to:
  - Get all of a vertex’s out-edges:
  - Get all of a vertex’s in-edges:
  - Decide if some edge exists:
  - Insert an edge:
  - Delete an edge:

- Space requirements:
  -

- Best for dense or sparse graphs?
Adjacency List Properties

- Running time to:
  - Get all of a vertex’s out-edges: $O(d)$ where $d$ is out-degree of vertex
  - Get all of a vertex’s in-edges: $O(|E|)$ (but could keep a second adjacency list for this!)
  - Decide if some edge exists: $O(d)$ where $d$ is out-degree of source
  - Insert an edge: $O(1)$ (unless you need to check if it’s there)
  - Delete an edge: $O(d)$ where $d$ is out-degree of source

- Space requirements:
  - $O(|V|+|E|)$

- Best for dense or sparse graphs?
  - Best for sparse graphs, so usually just stick with linked lists
Undirected Graphs

Adjacency matrices & adjacency lists both do fine for undirected graphs

• Matrix: Could save space by using only about half the array
  – How would you “get all neighbors”?
• Lists: Each edge in two lists to support efficient “get all neighbors”

Example: