Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs
Introduction to Graphs

• Undirected Graphs
• Representation
• Depth-First Search
  • Connected Components
• Breadth-First Search
  • Bipartite Graphs
Undirected graph. Set of vertices pairwise joined by edges.

Why graphs?
- Models many natural problems from many different areas.
- Thousands of practical applications.
- Hundreds of well-known graph algorithms.
Visualizing the Internet
Visualizing Friendships on Facebook

"Visualizing friendships", Paul Butler
This diagram is an evolution of the original design conceived in 1931 by Harry Beck. No step-free access from late January 2014 until Saturdays. Closed Sundays and Public Holidays.

Step-free interchange between Underground, Canary Wharf DLR and Heron Quays DLR stations. Step-free interchange between Heron Quays and Sunday.

Opening hours are 0800-2200 Mondays to Fridays and 0830-0000 on Mondays to Fridays. Between Waterloo and Bank 0630-2130 Mondays to Fridays and 0800-0000 Public Holidays. Opening hours are 0800-2200 Sundays and after 0745 Sundays and after evening. At other times use District line.
Protein Interaction Networks

Protein-protein interaktionsnetværk,
Jeong et al, Nature Review | Genetics
### Applications of Graphs

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>computers</td>
<td>cables</td>
</tr>
<tr>
<td>transport</td>
<td>intersections</td>
<td>roads</td>
</tr>
<tr>
<td>transport</td>
<td>airports</td>
<td>flight routes</td>
</tr>
<tr>
<td>games</td>
<td>position</td>
<td>valid move</td>
</tr>
<tr>
<td>neural network</td>
<td>neuron</td>
<td>synapses</td>
</tr>
<tr>
<td>financial network</td>
<td>stocks</td>
<td>transactions</td>
</tr>
<tr>
<td>circuit</td>
<td>logical gates</td>
<td>connections</td>
</tr>
<tr>
<td>food chain</td>
<td>species</td>
<td>predator-prey</td>
</tr>
<tr>
<td>molecule</td>
<td>atom</td>
<td>bindings</td>
</tr>
</tbody>
</table>
**Terminology**

- **Undirected graph.** \( G = (V, E) \)
  - \( V \) = set of vertices
  - \( E \) = set of edges (each edge is a pair of vertices)
  - \( n = |V|, m = |E| \)
- **Path.** Sequence of vertices connected by edges.
- **Cycle.** Path starting and ending at the same vertex.
- **Degree.** \( \text{deg}(v) = \) the number of neighbors of \( v \), or edges incident to \( v \).
- **Connectivity.** A pair of vertices are connected if there is a path between them.

\[
V = \{0, 1, 2, \ldots, 12\}
\]
\[
E = \{(0, 1), (0, 2), (0, 4), (2, 3), \ldots, (11, 12)\}
\]
\[
n = 13, m = 15
\]
Undirected Graphs

- **Lemma.** $\sum_{v \in V} \deg(v) = 2m$.
- **Proof.** How many times is each edge counted in the sum?
Algoctmic Problems on Graphs

• **Path.** Is there a path connecting s and t?

• **Shortest path.** What is the shortest path connecting s and t?

• **Longest path.** What is the longest path connecting s and t?

• **Cycle.** Is there a cycle in the graph?

• **Euler tour.** Is there a cycle that uses each edge exactly once?

• **Hamilton cycle.** Is there a cycle that uses each vertex exactly once?

• **Connectivity.** Are all pairs of vertices connected?

• **Minimum spanning tree.** What is the best way of connecting all vertices?

• **Biconnectivity.** Is there a vertex whose removal would cause the graph to be disconnected?

• **Planarity.** Is it possible to draw the graph in the plane without edges crossing?

• **Graph isomorphism.** Do these sets of vertices and edges represent the same graph?
Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs
Representation

- Graph G with n vertices and m edges.

**Representation.** We need the following operations on graphs.

- \text{ADJACENT}(v, u): determine if u and v are neighbors.
- \text{NEIGHBORS}(v): return all neighbors of v.
- \text{INSERT}(v, u): add the edge \((v, u)\) to G (unless it is already there).
**Adjacency Matrix**

- Graph G with n vertices and m edges.
- **Adjacency matrix.**
  - 2D \( n \times n \) array \( A \).
  - \( A[i,j] = 1 \) if i and j are neighbors, 0 otherwise.
- **Complexity?**
- **Space.** \( O(n^2) \)
- **Time.**
  - **ADJACENT** and **INSERT** in \( O(1) \) time.
  - **NEIGHBOURS** in \( O(n) \) time.

---

### Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Adjacency List

- Graph $G$ with $n$ vertices and $m$ edges.
- **Adjacency list.**
  - Array $A[0..n-1]$.
  - $A[i]$ is a linked list of all neighbors of $i$.
- **Complexity?**
  - **Space.** $O(n + \sum_{v \in V} \text{deg}(v)) = O(n + m)$
- **Time.**
  - $\text{ADJACENT, NEIGHBOURS, INSERT}$ $O(\text{deg}(v))$ time.
Real world graphs are often **sparse**.

<table>
<thead>
<tr>
<th>Data structure</th>
<th>ADJACENT</th>
<th>NEIGHBOURS</th>
<th>INSERT</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacency matrix</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n²)</td>
</tr>
<tr>
<td>adjacency list</td>
<td>O(deg(v))</td>
<td>O(deg(v))</td>
<td>O(deg(v))</td>
<td>O(n+m)</td>
</tr>
</tbody>
</table>
Introduction to Graphs

• Undirected Graphs
• Representation
• Depth-First Search
  • Connected Components
• Breadth-First Search
  • Bipartite Graphs
Depth-First Search

- Algorithm for systematically visiting all vertices and edges.
- Depth first search from vertex s.
  - Unmark all vertices and visit s.
  - Visit vertex v:
    - Mark v.
    - Visit all unmarked neighbours of v recursively.

- Intuition.
  - Explore from s in some direction, until we read dead end.
  - Backtrack to the last position with unexplored edges.
  - Repeat.

- Discovery time. First time a vertex is visited.
- Finish time. Last time a vertex is visited.
**Depth-First Search**

<table>
<thead>
<tr>
<th>DFS(s)</th>
<th>value = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS-VISIT(s)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DFS-VISIT(v)</th>
<th>v.d = time++</th>
</tr>
</thead>
<tbody>
<tr>
<td>mark v</td>
<td>for each unmarked neighbor u</td>
</tr>
<tr>
<td>DFS-VISIT(u)</td>
<td>u.π = v</td>
</tr>
<tr>
<td>v.f = time++</td>
<td></td>
</tr>
</tbody>
</table>

- **Time.** (on adjacency list representation)
  - Recursion? once per vertex.
  - \( O(\text{deg}(v)) \) time spent on vertex \( v \).
  - \( \Rightarrow \) total \( O(n + \sum_{v \in V} \text{deg}(v)) = O(n + m) \) time.
  - Only visits vertices connected to \( s \).
Flood Fill

- **Flood fill.** Chance the color of a connected area of green pixels.

- **Algorithm.**
  - Build a grid graph and run DFS.
  - Vertex: pixel.
  - Edge: between neighboring pixels of same color.
  - Area: connected component
Connected Components

• **Definition.** A connected component is a maximal subset of connected vertices.

[Diagram of connected components]

• How to find all connected components?

• **Algorithm.**
  • Unmark all vertices.
  • While there is an unmarked vertex:
    • Chose an unmarked vertex \( v \), run DFS from \( v \).

• **Time.** \( O(n + m) \).
Introduction to Graphs

• Undirected Graphs
• Representation
• Depth-First Search
  • Connected Components
• Breadth-First Search
  • Bipartite Graphs
Breadth-First Search

- Breadth first search from s.
  - **Unmark** all vertices and initialize queue Q.
  - Mark s and Q.ENQUEUE(s).
  - While Q is not empty:
    - \( v = Q.DEQUEUE() \).
    - For each unmarked neighbor u of v
      - Mark u.
      - Q.ENQUEUE(u).

- Intuition.
  - Explore, starting from s, in all directions - in increasing distance from s.

- Shortest paths from s.
  - Distance to s in **BFS tree** = shortest distance to s in the original graph.
Shortest Paths

- **Lemma.** BFS finds the length of the shortest path from \( s \) to all other vertices.

- **Intuition.**
  - BFS assigns vertices to **layers**. Layer number \( i \) contains all vertices of distance \( i \) to \( s \).

  - What does each layer contain?
  - \( L_0 : \{s\} \)
  - \( L_1 : \) all neighbours of \( L_0 \).
  - \( L_2 : \) all neighbours if \( L_1 \) that are not neighbours of \( L_0 \)
  - \( L_3 : \) all neighbours of \( L_2 \) that neither are neighbours of \( L_0 \) nor \( L_1 \).
  - ...
  - \( L_i : \) all neighbours til \( L_{i-1} \) not neighbouring \( L_j \) for \( j < i-1 \)
    - \( = \) all vertices of distance \( i \) from \( s \).
**Breadth-First Search**

\[
\text{BFS}(s)
\begin{align*}
&\text{mark } s \\
&s.d = 0 \\
&\text{Q.} \text{ENQUEUE}(s) \\
&\text{repeat until Q.ISEMPTY()} \\
&\quad v = \text{Q.DEQUEUE()} \\
&\quad \text{for each unmarked neighbor } u \\
&\quad \quad \text{mark } u \\
&\quad \quad u.d = v.d + 1 \\
&\quad \quad u.\pi = v \\
&\quad \text{Q.} \text{ENQUEUE}(u)
\end{align*}
\]

- **Time.** (on adjacency list representation)
  - Each vertex is visited at most once.
  - \(O(\text{deg}(v))\) time spent on vertex \(v\).
  - \(\Rightarrow\) total \(O(n + \sum_{v \in V} \text{deg}(v)) = O(n + m)\) time.
- Only vertices connected to \(s\) are visited.
Bipartite Graphs

• **Definition.** A graph is *bipartite* if and only if all vertices can be colored red and blue such that every edge has exactly one red endpoint and one blue endpoint.

• **Equivalent definition.** A graph is bipartite if and only if its vertices can be partitioned into two sets $V_1$ and $V_2$ such that all edges go between $V_1$ and $V_2$.

• **Application.**
  - Scheduling, matching, assigning clients to servers, assigning jobs to machines, assigning students to advisors/labs, ...
  - Many graph problems are *easier* on bipartite graphs.
Bipartite Graphs

- **Challenge.** Given a graph $G$, determine whether $G$ is bipartite.
Lemma. A graph $G$ is bipartite if and only if all cycles in $G$ have even length.

Proof. $\Rightarrow$

- If $G$ is bipartite, all cycles start and end on the same side.
Bipartite Graphs

- **Lemma.** A graph $G$ is bipartite if and only if all cycles in $G$ have even length.
- **Proof.** $\Leftarrow$
  - Choose a vertex $v$ and consider BFS layers $L_0, L_1, \ldots, L_k$.
  - All cycles have even length
  - $\Rightarrow$ There is no edge between vertices of the same layer
  - $\Rightarrow$ We can assign alternating (red, blue) colours to the layers
  - $\Rightarrow$ $G$ is bipartite.
Bipartite Graphs

• **Algorithm.**
  • Run BFS on $G$.
  • For each edge in $G$, check if it's endpoints are in the same layer.

• **Time.**
  • $O(n + m)$
Graph Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth first search</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Breadth first search</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Connected components</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Bipartite</td>
<td>$O(n + m)$</td>
<td>$O(n + m)$</td>
</tr>
</tbody>
</table>

- All on the adjacency list representation.
Introduction to Graphs

- Undirected Graphs
- Representation
- Depth-First Search
  - Connected Components
- Breadth-First Search
  - Bipartite Graphs