### 4.1 Undirected Graphs

#### Breadth-First Search

**Algorithm:**
- Create a queue and put the source vertex in it.
- Repeat until queue is empty:
  - Remove vertex \( v \) from queue.
  - Add to queue all unmarked vertices adjacent to \( v \) and mark them (so they are "visited").

**Graph Representation:**

```
Undirected Graph:
```

```
with added notes and slides by Betty O'Neill for cs310
```

#### Breadth-First Search Algorithm

- **Graph** \( G \)
- **Initialization:**
  - Queue for BFS from \( s \):
    - \([0]\)
  - \( distTo[0] = 0 \)
  - \( edgeTo[0] = -1 \)
- **Repeat** until queue is empty:
  - **Remove** vertex \( v \) from queue.
  - Add to queue all unmarked vertices adjacent to \( v \) and mark them.
  - Track visits with \( edgeTo \), \( distTo \):
    - \( distTo[v] = distTo[from - node] + 1 \)

**Example:**
- Graph \( G \)
- **Distances:**
  - \( distTo[0] = 0 \)
  - \( distTo[1] = 1 \)
  - \( distTo[2] = 2 \)
  - \( distTo[5] = 1 \)
  - \( distTo[3] = 2 \)
  - \( distTo[4] = 2 \)

#### Breadth-First Search Application: Routing

- **Proposition:**
  - In any connected graph \( G \), BFS computes shortest paths from \( s \) to all other vertices.

- In a communication network, BFS can be used to find the shortest path from a source to all other nodes.

**Algorithm in Java:**

```java
public class BreadthFirstPaths {
    private boolean[] marked;
    private int[] edgeTo;
    private int[] distTo;
    public BreadthFirstPaths(Graph G, int s) {
        Queue<Integer> q = new Queue<Integer>();
        q.enqueue(s);
        marked[s] = true;
        distTo[s] = 0;
        while (!q.isEmpty()) {
            int v = q.dequeue();
            for (int w : G.adj(v)) {
                if (!marked[w]) {
                    marked[w] = true;
                    edgeTo[w] = v;
                    distTo[w] = distTo[v] + 1;
                    q.enqueue(w);
                }
            }
        }
    }
}
```

**Graph Representation:**

```
with added notes and slides by Betty O'Neill for cs310
```

**Quiz:**
- In which order does BFS examine vertices?
  - A. Increasing distance (number of edges) from \( s \): \( v \) itself, all distance-1 vertices, all distance-2 vertices, …

**Breadth-First Search Properties**

- **Proposition:**
  - In any connected graph \( G \), BFS computes shortest paths from \( s \) to all other vertices in time proportional to \( V + E \).
MBTA subway system: subject of pa4

From https://cdn.mbta.com/sites/default/files/maps/2019-04-08-rapid-transit-key-bus-routes-map-v33.pdf

Breadth-first application: Kevin Bacon numbers

Kevin Bacon graph (page 549)

- Include one vertex for each performer and one for each movie.
- Connect a movie to all performers that appear in that movie.
- Compute shortest path from \( s = \text{Kevin Bacon} \).
- Data in movies.txt in algs4-data.zip

Kevin Bacon

Kathleen Quinlan
Meryl Streep
Nicole Kidman
John Gielgud
Kate Winslet
Bill Paxton
Donald Sutherland
Wives
Portrait of a Lady
Apollo 13
To Catch a Thief
The Eagle Has Landed
Cold Mountain
Murder on the Orient Express
Vernon Dobtcheff
An American Haunting
Jude Enigma
Eternal Sunshine of the Spotless Mind
The Woodsman
Wild Things
Hamlet
Titanic
Animal House
Grace Kelly
Caligola
The River
Wild Lloyd Bridges
High Noon
The Da Vinci Code
Joe Versus the Volcano
Dial M for Murder
Patrick Allen
Tom Hanks
Serretta Wilson
Close
Glenn
The Stepford Wives
John Belushi
Yves Aubert
Shane Zaza
Paul Herbert

Breadth-first application: Erdös numbers (mine is 2!)

hand-drawing of part of the Erdös graph by Ron Graham

4.1 Undirected Graphs

Algorithms

Def. Vertices \( v \) and \( w \) are connected if there is a path between them.

Goal. Preprocess graph to answer queries of the form \( u \) is connected to \( v \) in constant time. Provide processed graph info by setting up an API...

API on page 543:

- public class CC
- boolean connected(int v, int w)
- int count()
- int id(int v)

Connectivity queries

Union-Find? Not quite.
Depth-first search. Yes. [next few slides]
The relation "is connected to" is an equivalence relation:

- Reflexive: v is connected to v.
- Symmetric: if v is connected to w, then w is connected to v.
- Transitive: if v connected to w and w connected to x, then v connected to x.

Def. A connected component is a maximal set of connected vertices.

Remark. Given connected components, can answer queries in constant time.

Connected components algorithm

To visit a vertex v: do dfs from v:

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v.

Graph G

Finding connected components with DFS

public class CC {
  private boolean[] marked;
  private int[] id;
  private int count;
  public CC(Graph G) {
    marked = new boolean[G.V()];
    id = new int[G.V()];
    for (int v = 0; v < G.V(); v++)
      if (!marked[v]) {
        dfs(G, v);
        count++;
      }
  }
  public int count()
  public int id(int v)
  public boolean connected(int v, int w)
  private void dfs(Graph G, int v) {
    marked[v] = true;
    id[v] = id[v];
    for (int w : G.adj(v))
      if (!marked[w])
        dfs(G, w);
  }

Goal. Partition vertices into connected components.

Connected components

For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component.
Finding connected components with DFS (continued)

```java
public int count()
    { return count; }

public int id(int v)
    { return id[v]; }

public boolean connected(int v, int w)
    { return id[v] == id[w]; }

private void dfs(Graph G, int v)
    {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }
```

All vertices discovered in same call of dfs have same id number of components

id of component containing v v and w connected iff same id

Connected components application: study spread of STDs

Relationship graph at “Jefferson High”


Particle tracking. Track moving particles over time.

Particle detection. Given grayscale image of particles, identify "blobs."

- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value > 70.
- Blob: connected component of 20–30 pixels.

4.1 UNDIRECTED GRAPHS

Graph-processing challenge 1

Problem. Is a graph bipartite?

Definition, page 521: vertices can be divided into two groups such that all edges connect a vertex in one group with a vertex in the other group, i.e., you can "color the graph" in two colors.

How difficult?

Typical diligent algorithms student could do it. How are expert.

Intractable.

No one knows. Impossible.

Bipartiteness application: is dating graph bipartite?
Graph processing challenge 2

Problem. Find a cycle.

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Idea: if there are no cycles, the graph is tree-like and the dfs just keeps finding new vertices, never finding marked vertices as it processes adjacency lists. If there’s a cycle, the dfs will encounter a marked vertex, but it’s tricky, sometimes it’s just seeing the same edge.

Graph processing challenge 5

Problem. Are two graphs identical except for vertex names?

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Graph isomorphism is longstanding open problem.

Graph processing challenge 6

Problem. Lay out a graph in the plane without crossing edges?

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Typical diligence algorithms student could do it.

Linear-time DFS-based planarity algorithm discovered by Tarjan in 1970s (too complicated for most practitioners).

Graph traversal summary

BFS and DFS enable efficient solution of many (but not all) graph problems.

<table>
<thead>
<tr>
<th>problem</th>
<th>BFS</th>
<th>DFS</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>path between s and t</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>shortest path between s and t</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>connected components</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>biconnected components</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>cycle</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>Euler cycle</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>Hamilton cycle</td>
<td>✓</td>
<td>✓</td>
<td>$</td>
</tr>
<tr>
<td>bipartiteness</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>planarity</td>
<td>✓</td>
<td>✓</td>
<td>$E + V$</td>
</tr>
<tr>
<td>graph isomorphism</td>
<td>✓</td>
<td>✓</td>
<td>$\log V$</td>
</tr>
</tbody>
</table>