4.2 Directed Graphs

- introduction
- digraph API
- digraph search
- topological sort
- strong components

With added notes and slides by Betty O’Neil for cs310
4.2 Directed Graphs

- introduction
- digraph API
- digraph search
- topological sort
- strong components
Directed graphs

**Digraph.** Set of vertices connected pairwise by directed edges.
Road network

Vertex = intersection; edge = one-way street.
Vertex = political blog; edge = link.

The Political Blogosphere and the 2004 U.S. Election: Divided They Blog, Adamic and Glance, 2005
Overnight interbank loan graph

Vertex = bank; edge = overnight loan.

The Topology of the Federal Funds Market, Bech and Atalay, 2008
Uber taxi graph

Vertex = taxi pickup; edge = taxi ride.

http://blog.uber.com/2012/01/09/uberdata-san-franciscomics/
Implication graph

Vertex = variable; edge = logical implication.
Combinational circuit

Vertex = logical gate; edge = wire.
WordNet graph

Vertex = synset; edge = hypernym relationship.

http://wordnet.princeton.edu
## Digraph applications

<table>
<thead>
<tr>
<th>digraph</th>
<th>vertex</th>
<th>directed edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>transportation</td>
<td>street intersection</td>
<td>one-way street</td>
</tr>
<tr>
<td>web</td>
<td>web page</td>
<td>hyperlink</td>
</tr>
<tr>
<td>food web</td>
<td>species</td>
<td>predator-prey relationship</td>
</tr>
<tr>
<td>WordNet</td>
<td>synset</td>
<td>hypernym</td>
</tr>
<tr>
<td>scheduling</td>
<td>task</td>
<td>precedence constraint</td>
</tr>
<tr>
<td>financial</td>
<td>bank</td>
<td>transaction</td>
</tr>
<tr>
<td>cell phone</td>
<td>person</td>
<td>placed call</td>
</tr>
<tr>
<td>infectious disease</td>
<td>person</td>
<td>infection</td>
</tr>
<tr>
<td>game</td>
<td>board position</td>
<td>legal move</td>
</tr>
<tr>
<td>citation</td>
<td>journal article</td>
<td>citation</td>
</tr>
<tr>
<td>object graph</td>
<td>object</td>
<td>pointer</td>
</tr>
<tr>
<td>inheritance hierarchy</td>
<td>class</td>
<td>inherits from</td>
</tr>
<tr>
<td>control flow</td>
<td>code block</td>
<td>jump</td>
</tr>
</tbody>
</table>
Some digraph problems

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s \rightarrow t$ path</td>
<td>Is there a path from $s$ to $t$ ?</td>
</tr>
<tr>
<td>shortest $s \rightarrow t$ path</td>
<td>What is the shortest path from $s$ to $t$ ?</td>
</tr>
<tr>
<td>directed cycle</td>
<td>Is there a directed cycle in the graph ?</td>
</tr>
<tr>
<td>topological sort</td>
<td>Can the digraph be drawn so that all edges point upwards?</td>
</tr>
<tr>
<td>strong connectivity</td>
<td>Is there a directed path between all pairs of vertices ?</td>
</tr>
<tr>
<td>transitive closure</td>
<td>For which vertices $v$ and $w$ is there a directed path from $v$ to $w$ ?</td>
</tr>
<tr>
<td>PageRank</td>
<td>What is the importance of a web page ?</td>
</tr>
</tbody>
</table>
4.2 Directed Graphs

- introduction
- digraph API
- digraph search
- topological sort
- strong components
Digraph API

Almost identical to Graph API.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Digraph(int V)</code></td>
<td>create an empty digraph with V vertices</td>
</tr>
<tr>
<td><code>Digraph(In in)</code></td>
<td>create a digraph from input stream</td>
</tr>
<tr>
<td><code>void addEdge(int v, int w)</code></td>
<td>add a directed edge ( v \rightarrow w )</td>
</tr>
<tr>
<td><code>Iterable&lt;Integer&gt; adj(int v)</code></td>
<td>vertices pointing from ( v )</td>
</tr>
<tr>
<td><code>int V()</code></td>
<td>number of vertices</td>
</tr>
<tr>
<td><code>int E()</code></td>
<td>number of edges</td>
</tr>
<tr>
<td><code>Digraph reverse()</code></td>
<td>reverse of this digraph</td>
</tr>
<tr>
<td><code>String toString()</code></td>
<td>string representation</td>
</tr>
</tbody>
</table>
What about an interface?

Since Graph and Digraph have many of the same methods, we can put them in an interface:

```java
public interface BaseGraph {
    int V();
    int E();
    void addEdge(int v, int w);
    Iterable<Integer> adj(int v);
}
```

```java
public class Graph implements BaseGraph {...}
public class Digraph implements BaseGraph {...}
```

- Then we could write generic code across a group of graphs, some of which are Graphs and others Digraphs
- However, this is a rare thing to do, so we’ll adopt S&W’s simpler approach of treating them separately as types.
- Another way to handle a group of Graphs and Digraphs: convert the Graphs to equivalent Digraphs (all edges both ways)
Digraph API

```java
In in = new In(args[0]);
Digraph G = new Digraph(in);

for (int v = 0; v < G.V(); v++)
    for (int w : G.adj(v))
        StdOut.println(v + "->" + w);
```

% java Digraph tinyDG.txt
0->5
0->1
2->0
2->3
3->5
3->2
4->3
4->2
5->4
... 
11->4
11->12
12->9

read digraph from input stream
print out each edge (once)
public static Digraph createDigraphFromFile(String filePath) {
    Scanner in = null;
    try {
        in = new Scanner(new File(filePath));
    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + filePath);
        return null;
    }
    int nV = Integer.parseInt(in.nextLine());
    int nE = Integer.parseInt(in.nextLine());
    System.out.println("creating graph of " + nV + " nodes " + nE + " edges");
    Digraph G = new Digraph(nV);
    while (in.hasNextLine()) {
        String line1 = in.nextLine().trim();  // trim off spaces on either end
        String[] tokens = line1.split("\s+");  // allow multiple spaces between
        G.addEdge(Integer.parseInt(tokens[0]), Integer.parseInt(tokens[1].trim()));
    }
    in.close();
    return G;
}
Digraph representation: adjacency lists

Maintain vertex-indexed array of lists.
Digraph representations

In practice. Use adjacency-lists representation.
- Algorithms based on iterating over vertices pointing from \( v \).
- Real-world digraphs tend to be sparse.

<table>
<thead>
<tr>
<th>representation</th>
<th>space</th>
<th>insert edge from ( v ) to ( w )</th>
<th>edge from ( v ) to ( w )?</th>
<th>iterate over vertices pointing from ( v )?</th>
</tr>
</thead>
<tbody>
<tr>
<td>list of edges</td>
<td>( E )</td>
<td>1</td>
<td>( E )</td>
<td>( E )</td>
</tr>
<tr>
<td>adjacency matrix</td>
<td>( V^2 )</td>
<td>1( ^\dagger )</td>
<td>1</td>
<td>( V )</td>
</tr>
<tr>
<td>adjacency lists</td>
<td>( E + V )</td>
<td>1</td>
<td>( \text{outdegree}(v) )</td>
<td>( \text{outdegree}(v) )</td>
</tr>
</tbody>
</table>

\( ^\dagger \) disallows parallel edges

huge number of vertices, small average vertex degree
Adjacency-lists **Undirected** Graph representation (review):

```java
public class Graph {
    private final int V;
    private final Bag<Integer>[] adj;

    public Graph(int V) {
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w) {
        adj[v].add(w);
        adj[w].add(v);
    }

    public Iterable<Integer> adj(int v) {
        return adj[v];
    }
}
```
Adjacency-lists Digraph representation: Java implementation

```java
public class Digraph {
    private final int V;
    private final Bag<Integer>[] adj;

    public Digraph(int V) {
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w) {
        adj[v].add(w);
    }

    public Iterable<Integer> adj(int v) {
        return adj[v];
    }
}
```
4.2 Directed Graphs

- Introduction
- digraph API
- digraph search
- topological sort
- strong components
## Recall Digraph API

Almost identical to Graph API. One additional method: reverse()
Reachability

Problem. Find all vertices reachable from $s$ along a directed path.
Depth-first search in digraphs

Same method as for undirected graphs.

• Every undirected graph is a digraph (with edges in both directions).
• DFS is a digraph algorithm.

**DFS (to visit a vertex v)**

- Mark v as visited.
- Recursively visit all unmarked vertices w pointing from v.
Depth-first search algorithm

To visit a vertex $v$:
- Mark vertex $v$ as visited.
- Recursively visit all unmarked vertices pointing from $v$.

A directed graph
Depth-first search algorithm

To visit a vertex $v$:

- Mark vertex $v$ as visited.
- Recursively visit all unmarked vertices pointing from $v$.

Dfs from 0: 1, done with 1, back to 0, 5, 4, 3, 2, done with 2, back to 3, done with 3, back to 4, try 2 but marked, done with 4, back to 5, done with 5, back to 0, done with 0

Marked so far: 0, 1, 2, 3, 4, 5

edgeTo filled in, in order:

- edgeTo[1] = 0 for 0->1
- edgeTo[5] = 0 for 0->5
- edgeTo[4] = 5 ...
- edgeTo[3] = 4
- edgeTo[2] = 3

reachable from 0
Depth-first search algorithm

To visit a vertex $v$:

- Mark vertex $v$ as visited.
- Recursively visit all unmarked vertices pointing from $v$. 

### Table

<table>
<thead>
<tr>
<th>$v$</th>
<th>marked[]</th>
<th>edgeTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>F</td>
<td>-</td>
</tr>
</tbody>
</table>
Recall code for **undirected** graphs.

```java
public class DepthFirstSearch {
    private boolean[] marked;

    public DepthFirstSearch(Graph G, int s) {
        marked = new boolean[G.V()];
        dfs(G, s);
    }

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

    public boolean visited(int v) {
        return marked[v];
    }
}
```

**Depth-first search** (in undirected graphs)
- True if connected to s
- Constructor marks vertices connected to s
- Recursive DFS does the work
- Client can ask whether any vertex is connected to s
Depth-first search (in directed graphs)

Code for directed graphs identical to undirected one.
[substitute Digraph for Graph]

```java
public class DirectedDFS {
    private boolean[] marked;

    public DirectedDFS(Digraph G, int s) {
        marked = new boolean[G.V()];
        dfs(G, s);
    }

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

    public boolean visited(int v) {
        return marked[v];
    }
}
```
Reachability application: program control-flow analysis

Every program is a digraph.
- Vertex = basic block of instructions (straight-line program).
- Edge = jump.

Dead-code elimination.
Find (and remove) unreachable code.

Infinite-loop detection.
Determine whether exit is unreachable.
Reachability application: mark-sweep garbage collector

Every data structure is a digraph.

- Vertex = object.
- Edge = reference.

Roots. Objects known to be directly accessible by program (e.g., stack).

Reachable objects. Objects indirectly accessible by program (starting at a root and following a chain of pointers).
Reachability application: mark-sweep garbage collector

**Mark-sweep algorithm.** [McCarthy, 1960]
- Mark: mark all reachable objects.
- Sweep: if object is unmarked, it is garbage (so add to free list).

**Memory cost.** Uses 1 extra mark bit per object (plus DFS stack).
Depth-first search in digraphs summary

**DFS enables direct solution of simple digraph problems.**

✓ • Reachability.
   • Path finding.
   • Topological sort.
   • Directed cycle detection.

**Basis for solving difficult digraph problems.**

• 2-satisfiability (in implication graphs, not covered)
• Directed Euler path (not covered).
• Strongly-connected components (covered by clever algorithm, not that difficult)
Breadth-first search in digraphs

Same method as for undirected graphs.
- Every undirected graph is a digraph (with edges in both directions).
- BFS is a digraph algorithm.

**BFS (from source vertex s)**

Put s onto a FIFO queue, and mark s as visited.
Repeat until the queue is empty:
- remove the least recently added vertex v
- for each unmarked vertex pointing from v:
  add to queue and mark as visited.

**Proposition.** BFS computes shortest paths (fewest number of edges) from s to all other vertices in a digraph in time proportional to $E + V$. 
Directed breadth-first search algorithm

Repeat until queue is empty:
  • Remove vertex \( v \) from queue.
  • Add to queue all unmarked vertices pointing from \( v \) and mark them.

Graph G

tinyDG2.txt

\[
\begin{array}{c|c}
V & 6 \\
\hline
E & 5 0 \\
& 2 4 \\
& 3 2 \\
& 1 2 \\
& 0 1 \\
& 4 3 \\
& 3 5 \\
& 0 2 \\
\end{array}
\]
Directed breadth-first search algorithm

Repeat until queue is empty:
  • Remove vertex $v$ from queue.
  • Add to queue all unmarked vertices pointing from $v$ and mark them.

Queue states:
- [0]
- [1]
- [2,1] dist 1 verts
- [2]
- [4] dist 2 vert
- [3] dist 3 vert
- [5] dist 4 vert

In this case, the queue has only one distance at each step, but it can have two.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>edgeTo[]</th>
<th>distTo[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

done
Multiple-source shortest paths

**Multiple-source shortest paths.** Given a digraph and a set of source vertices, find shortest path from any vertex in the set to each other vertex.

**Ex.** $S = \{ 1, 7, 10 \}$.
- Shortest path to 4 is $7 \rightarrow 6 \rightarrow 4$.
- Shortest path to 5 is $7 \rightarrow 6 \rightarrow 0 \rightarrow 5$.
- Shortest path to 12 is $10 \rightarrow 12$.
- ...

**Q.** How to implement multi-source shortest paths algorithm?

**A.** Use BFS, but initialize by enqueuing all source vertices.
Breadth-first search in digraphs application: web crawler


Solution. [BFS with implicit digraph]
• Choose root web page as source $s$.
• Maintain a Queue of websites to explore.
• Maintain a SET of discovered websites.
• Dequeue the next website and enqueue websites to which it links (provided you haven't done so before).

Q. Why not use DFS?
Bare-bones web crawler: Java implementation
(CS310: don’t worry about details here)

```java
Queue<String> queue = new Queue<String>();
SET<String> marked = new SET<String>();

String root = "http://www.princeton.edu";
queue.enqueue(root);
marked.add(root);

while (!queue.isEmpty())
{
    String v = queue.dequeue();
    StdOut.println(v);
    In in = new In(v);
    String input = in.readAll();
    String regexp = "http://(\w+.)+(.+)";
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(input);
    while (matcher.find())
    {
        String w = matcher.group();
        if (!marked.contains(w))
        {
            marked.add(w);
            queue.enqueue(w);
        }
    }
}
```

- Queue of websites to crawl
- Set of marked websites
- Start crawling from root website
- Read in raw HTML from next website in queue
- `In(URL)` can do a web request
- Use regular expression to find all URLs in website of form `http://xxx.yyy.zzz`
  [crude pattern misses relative URLs]
  \w (word character) matches any single letter, number or underscore (same as [a-zA-Z0-9_])
- If unmarked, mark it and put on the queue
Web crawler output

**BFS crawl**

http://www.princeton.edu
http://www.w3.org
http://ogp.me
http://giving.princeton.edu
http://www.princetonartmuseum.org
http://www.goprincetontigers.com
http://library.princeton.edu
http://helpdesk.princeton.edu
http://tigernet.princeton.edu
http://alumni.princeton.edu
http://gradschool.princeton.edu
http://vimeo.com
http://princetonusg.com
http://artmuseum.princeton.edu
http://jobs.princeton.edu
http://odoc.princeton.edu
http://blogs.princeton.edu
http://www.facebook.com
http://twitter.com
http://www.youtube.com
http://deimos.apple.com
http://qeprize.org
http://en.wikipedia.org
...

**DFS crawl**

http://www.princeton.edu
http://deimos.apple.com
http://www.youtube.com
http://www.google.com
http://news.google.com
http://csi.gstatic.com
http://googlenewsblog.blogspot.com
http://labs.google.com
http://groups.google.com
http://img1.blogblog.com
http://feeds.feedburner.com
http://buttons.googlesyndication.com
http://fusion.google.com
http://insidesearch.blogspot.com
http://agoogleaday.com
http://static.googleusercontent.com
http://searchresearch1.blogspot.com
http://feedburner.google.com
http://www.dot.ca.gov
http://www.TahoeRoads.com
http://www.LakeTahoeTransit.com
http://www.laketahoe.com
http://ethel.tahoeguide.com
...

...