4. Graphs and Digraphs II

- breadth-first search (in directed graphs)
- breadth-first search (in graphs)
- topological sort
- challenges
Graph search overview

Tree traversal. Many ways to explore nodes in a binary tree.

- Inorder:  \(A\ C\ E\ H\ M\ R\ S\ X\)
- Preorder:  \(S\ E\ A\ C\ R\ H\ M\ X\)
- Postorder:  \(C\ A\ M\ H\ R\ E\ X\ S\)
- Level–order:  \(S\ E\ X\ A\ R\ C\ H\ M\)

Graph search. Many ways to explore vertices in a graph or digraph.

- DFS preorder: vertices in order of calls to \(dfs(G, \nu)\).
- DFS postorder: vertices in order of returns from \(dfs(G, \nu)\).
- BFS order: vertices in increasing order of distance from \(s\).
4. Graphs and Digraphs II

- breadth-first search (in directed graphs)
- breadth-first search (in graphs)
- topological sort
- challenges
Problem. Find directed path from $s$ to each other vertex that uses the fewest edges.

directed paths from 0 to 6

- $0 \rightarrow 2 \rightarrow 7 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 3 \rightarrow 6$
- $0 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 3 \rightarrow 6$
- $0 \rightarrow 2 \rightarrow 7 \rightarrow 3 \rightarrow 6$
- $0 \rightarrow 2 \rightarrow 7 \rightarrow 0 \rightarrow 2 \rightarrow 7 \rightarrow 3 \rightarrow 6$

shortest path from 0 to 6 (length = 4)

- $0 \rightarrow 2 \rightarrow 7 \rightarrow 3 \rightarrow 6$

shortest path must be simple (no repeated vertices)
Shortest paths in a digraph

**Problem.** Find directed path from $s$ to each other vertex that uses the fewest edges.

**Key idea.** Visit vertices in increasing order of distance from $s$.

**How to implement?** Store vertices to visit in a queue.
Breadth-first search demo

Repeat until queue is empty:

- Remove vertex $v$ from queue.
- Add to queue all unmarked vertices adjacent from $v$ and mark them.

Graph $G$

```
tinyDG2.txt
6
8
5 0
2 4
3 2
1 2
0 1
4 3
3 5
0 2
```
Breadth-first search demo

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

vertices reachable from 0
(and shortest directed paths)
Breadth-first search

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

BFS (from source vertex $s$)

Add vertex $s$ to FIFO queue and mark $s$.
Repeat until the queue is empty:
- remove the least recently added vertex $v$
- for each unmarked vertex $w$ adjacent from $v$:
  add $w$ to queue and mark $w$
Breadth-first search: Java implementation

```java
public class BreadthFirstDirectedPaths {
    private boolean[] marked;
    private int[] edgeTo;
    private int[] distTo;
    ...

    private void bfs(Digraph G, int s) {
        Queue<Integer> queue = new Queue<>();
        queue.enqueue(s);
        marked[s] = true;
        distTo[s] = 0;
        while (!queue.isEmpty()) {
            int v = queue.dequeue();
            for (int w : G.adj(v)) {
                if (!marked[w]) {
                    queue.enqueue(w);
                    marked[w] = true;
                    edgeTo[w] = v;
                    distTo[w] = distTo[v] + 1;
                }
            }
        }
    }
}
```

- **initialize queue of vertices to explore**
- **also safe to stop as soon as all vertices marked**
- **found new vertex w via edge v→w**

Breadth-first search properties

Proposition. In the worst case, BFS takes $\Theta(E + V)$ time.

**Pf.** Each vertex reachable from $s$ is visited once.

Proposition. BFS computes shortest paths from $s$.

**Pf idea.** BFS examines vertices in increasing order of distance (number of edges) from $s$.

invariant: queue contains some vertices of distance $k$ from $s$, followed by $\geq 0$ vertices of distance $k+1$ (and no other vertices)

![digraph G](image)

$\text{dist} = 0 \quad \text{dist} = 1 \quad \text{dist} = 2 \quad \text{dist} = 3 \quad \text{dist} = 4$
What could happen if we mark a vertex when it is dequeued (instead of enqueued)?

A. Doesn’t find a shortest path.
B. Takes exponential time.
C. Both A and B.
D. Neither A nor B.
Single-target shortest paths

Given a digraph and a target vertex \( t \), find shortest path from every vertex to \( t \).

**Ex.** \( t = 0 \)
- Shortest path from 7 is 7→6→0.
- Shortest path from 5 is 5→4→2→0.
- Shortest path from 12 is 12→9→11→4→2→0.
- ...

**Q.** How to implement single-target shortest paths algorithm?
Multiple-source shortest paths

Given a digraph and a set of source vertices, find shortest path from any vertex in the set to every 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?
Suppose that you want to design a web crawler. Which algorithm should you use?

A. Depth-first search.
B. Breadth-first search.
C. Either A or B.
D. Neither A nor B.
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 marked websites.
- Dequeue the next website and enqueue any unmarked websites to which it links.

Caveat. Industrial-strength web crawlers use same basic idea, but more sophisticated techniques.
Bare-bones web crawler: Java implementation

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

String root = "https://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.readString();

    String regexp = "https://(\w+)/+(\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
- `marked` set of marked websites
- Start crawling from the root website
- Read in raw HTML from the next website in the queue
- Use regular expression to find all URLs in websites of form `https://xxx.yyy.zzz`
  [crude pattern misses relative URLs]
- If unmarked, mark and enqueue
4. Graphs and Digraphs II

- breadth-first search (in directed graphs)
- breadth-first search (in undirected graphs)
- topological sort
- challenges
Application: routing in a communication network

Fewest number of hops in a communication network.
Breadth-first search in undirected graphs

**Problem.** Find path between $s$ and each other vertex that uses fewest edges.

**Solution.** Use BFS.

**Proposition.** BFS finds shortest paths between $s$ and every other vertex in $\Theta(E + V)$ time.

---

### BFS (from source vertex $s$)

Add vertex $s$ to FIFO queue and mark $s$.

Repeat until the queue is empty:

- remove the least recently added vertex $v$
- for each unmarked vertex $w$ adjacent to $v$:
  - add $w$ to queue and mark $w$

But now, for each undirected edge $v\rightarrow w$:

$v$ is adjacent to $w$ and $w$ is adjacent to $v$
Application: Kevin Bacon numbers

**The Oracle of Bacon**

Bernard Chazelle has a Bacon number of 3.

- Bernard Chazelle
  - Guy and Madeline on a Park Bench (2009)
  - Anna Chazelle
  - La La Land (2016/I)
  - Ryan Gosling
  - Crazy, Stupid, Love. (2011)
  - Kevin Bacon

https://oracleofbacon.org

**Endless Games board game**

**SixDegrees of Hollywood**
Kevin Bacon graph

- Include one vertex for each performer and one for each movie.
- Connect a movie to all performers that appear in that movie.
- Compute shortest paths between $s = \text{Kevin Bacon}$ and every other performer.
4. **Graphs and Digraphs II**

- breadth-first search (in directed graphs)
- breadth-first search (in undirected graphs)
- topological sort
- challenges

https://algs4.cs.princeton.edu
Directed acyclic graphs

**Directed acyclic graph (DAG).** A digraph with no directed cycles.

**Remark.** DAGs are an important subclass of digraphs that arise in many applications.
Vertex = synset; edge = hypernym relationship.

WordNet DAG

no directed cycles
Vertex = person; edge = biological child.
Bayesian networks

Vertex = variable; edge = conditional dependency.

Using DAGs for Investigating Causal Paths for Cardiovascular Disease
Combinational circuits

Digital logical circuit. Vertex = logic gate; edge = wire.

no directed cycles $\Rightarrow$ combinational circuit
Precedence scheduling

**Goal.** Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?

**Digraph model.** vertex = task; edge = precedence constraint.

0. Math for CS  
1. Complexity Theory  
2. Machine Learning  
3. Intro to CS  
4. Cryptography  
5. Scientific Computing  
6. Algorithms

0.  
1.  
2.  
3.  
4.  
5.  
6.  

**tasks**

**precedence constraint graph**

**feasible schedule**

no directed cycles
Topological sort

**Topological sort.** Given a DAG, find a linear ordering of the vertices so that for every edge $v \rightarrow w$, $v$ comes before $w$ in the ordering.

*edges in DAG define a “partial order” for vertices*

```
0→5  0→2
0→1  3→6
3→5  3→4
5→2  6→4
6→0  3→2
1→4
```

Directed edges

```
DAG

no directed cycles
```

topological ordering: 3 6 0 5 2 1 4
Suppose that you want to topologically sort the vertices in a DAG. Which graph-search algorithm should you use?

A. Depth-first search.
B. Breadth-first search.
C. Either A or B.
D. Neither A nor B.
Topological sort demo

- Run depth-first search.
- Return vertices in reverse DFS postorder.

```
0
1
2
3
4
5
6
```

`tinyDAG7.txt`

| 7  |
| 11 |
| 0 5 |
| 0 2 |
| 0 1 |
| 3 6 |
| 3 5 |
| 3 4 |
| 5 2 |
| 6 4 |
| 6 0 |
| 3 2 |

A directed acyclic graph
Topological sort demo

- Run depth-first search.
- Return vertices in reverse DFS postorder.

DFS postorder

4 1 2 5 0 6 3

topological ordering (reverse DFS postorder)

3 6 0 5 2 1 4
Depth-first search: reverse postorder

```java
public class DepthFirstOrder {
    private boolean[] marked;
    private Stack<Integer> reversePostorder;

    public DepthFirstOrder(Digraph G) {
        reversePostorder = new Stack<>();
        marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v])
                dfs(G, v);
    }

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

    public Iterable<Integer> reversePostorder() {
        return reversePostorder;
    }
}
```

- **run DFS from all vertices**
- **return vertices in reverse DFS postorder**
Topological sort in a DAG: intuition

Why is the reverse DFS postorder a topological ordering?

- First vertex in DFS postorder has outdegree 0.
- Second vertex in DFS postorder can point only to first vertex.
- ...

DFS postorder

```
4 1 2 5 0 6 3
```

topological ordering (reverse DFS postorder)

```
3 6 0 5 2 1 4
```
Topological sort in a DAG: correctness proof

**Proposition.** Reverse DFS postorder of a DAG is a topological ordering.

**Pf.** Consider any edge \( v \rightarrow w \). When \( \text{dfs}(v) \) is called:

- **Case 1:** \( \text{dfs}(w) \) has already been called and returned.
  - thus, \( w \) appears before \( v \) in DFS postorder

- **Case 2:** \( \text{dfs}(w) \) has not yet been called.
  - \( \text{dfs}(w) \) will get called directly or indirectly by \( \text{dfs}(v) \)
  - so, \( \text{dfs}(w) \) will return before \( \text{dfs}(v) \) returns
  - thus, \( w \) appears before \( v \) in DFS postorder

- **Case 3:** \( \text{dfs}(w) \) has already been called, but has not yet returned.
  - function-call stack contains directed path from \( w \) to \( v \)
  - adding edge \( v \rightarrow w \) to that path would complete a directed cycle
  - contradiction (it’s a DAG)
Topological sort in a DAG: running time

**Proposition.** For any DAG, the DFS algorithm computes a topological ordering in $\Theta(E + V)$ time.

**Pf.** For every vertex $v$, there is exactly one call to $dfs(v)$.

critical that vertices are marked
(and never unmarked)

**Q.** What if we run algorithm on a digraph that is not a DAG?

**A.** Reverse DFS postorder is still well defined, but it won’t be a topological ordering.
Directed cycle detection

**Proposition.** A digraph has a topological ordering if and only if contains no directed cycle.

**Pf.**
- Directed cycle $\Rightarrow$ topological ordering impossible.
- No directed cycle $\Rightarrow$ reverse DFS postorder is a topological ordering.

**Goal.** Given a digraph, find a directed cycle (if one exists).

**Solution.** DFS. What else? See textbook/precept.
Directed cycle detection application: precedence scheduling

Scheduling. Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?

Remark. A directed cycle implies scheduling problem is infeasible.

https://xkcd.com/754
Directed cycle detection application: cyclic inheritance

The Java compiler does directed cycle detection.

```
public class A extends B {
    ...
}
```

```
public class B extends C {
    ...
}
```

```
public class C extends A {
    ...
}
```

```
~/cos226/graph> javac A.java
A.java:1: cyclic inheritance involving A
public class A extends B {
  }
  ^
1 error
```
Directed cycle detection application: spreadsheet recalculation

Microsoft Excel does directed cycle detection.
4. **Graphs and Digraphs II**

- breadth-first search (in directed graphs)
- breadth-first search (in undirected graphs)
- topological sort
- challenges
Problem. Identify connected components.

How difficult?

A. Diligent algorithms student could do it.

B. Hire an expert.

C. Intractable.

D. No one knows.
Problem. Identify connected components.

Particle detection. Given grayscale image of particles, identify “blobs.”
- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value \( \geq 70 \).
- Blob: connected component of 20–30 pixels.
Graph-processing challenge 2

Problem. Is a graph bipartite?

How difficult?
A. Diligent algorithms student could do it.
B. Hire an expert.
C. Intractable.
D. No one knows.
Graph-processing challenge 3

Problem. Is there a (non-simple) cycle that uses every edge exactly once?

How difficult?
A. Diligent algorithms student could do it.
B. Hire an expert.
C. Intractable.
D. No one knows.

0-1 0-2 0-5 0-6 1-2 2-3 2-4 3-4 4-5 4-6
0-1 2-3 4-2 0-6 4-5 0-1-2-3-4-2-0-6-4-5-0

A Euler cycle exists if and only if graph is connected and every vertex has even degree (Euler 1786)
much more, if graph is Eulerian, can find a Euler cycle via DFS (Hierholzer 1873)
Problem. Is there a cycle that uses every vertex exactly once?

How difficult?

A. Diligent algorithms student could do it.
B. Hire an expert.
C. Intractable.
D. No one knows.
Graph-processing challenge 5

**Problem.** Are two graphs identical except for vertex names?

**How difficult?**

A. Diligent algorithms student could do it.

B. Hire an expert.

C. Intractable.

D. No one knows.
Problem. Can you draw a graph in the plane with no crossing edges?

How difficult?

A. Diligent algorithms student could do it.

B. Hire an expert.

C. Intractable.

D. No one knows
Graph processing summary

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

<table>
<thead>
<tr>
<th>graph problem</th>
<th>BFS</th>
<th>DFS</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-t path</td>
<td>✔</td>
<td>✔</td>
<td>$E + V$</td>
</tr>
<tr>
<td>shortest s-t path</td>
<td>✔</td>
<td></td>
<td>$E + V$</td>
</tr>
<tr>
<td>shortest directed 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>$2^{1.657V}$</td>
</tr>
<tr>
<td>bipartiteness (odd cycle)</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>strong components</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>$2e\ln^3V$</td>
</tr>
<tr>
<td>Single-source reachability</td>
<td>DFS/BFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shortest paths</td>
<td>BFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topological sort</td>
<td>DFS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Credits

<table>
<thead>
<tr>
<th>image</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPANET</td>
<td>Wikimedia</td>
<td>CC BY-SA 4.0</td>
</tr>
<tr>
<td>Oracle of Bacon</td>
<td>oracleofbacon.org</td>
<td></td>
</tr>
<tr>
<td>Kevin Bacon Game</td>
<td>Endless Games</td>
<td></td>
</tr>
<tr>
<td>Six Degrees of Hollywood</td>
<td>Paradox Apps</td>
<td></td>
</tr>
<tr>
<td>Ancestry of King Charles II</td>
<td>Waterford Treasures</td>
<td></td>
</tr>
<tr>
<td>Habsburg Coat of Arms</td>
<td>Wikimedia</td>
<td>CC BY-SA 3.0</td>
</tr>
<tr>
<td>Bayesian Network</td>
<td>Thornley et. al</td>
<td></td>
</tr>
</tbody>
</table>
BFS visualization (by Gerry Jenkins)

https://www.youtube.com/watch?v=x-VTfcmrLEQ