The slides are loosely based on those of Dr Kevin Wayne, Princeton University and Prof. Mary Wootters, Stanford University
1. Basic definitions and applications

2. Breadth-First Search

3. Depth-First Search

4. Topological ordering
### Undirected Graph

Graph $G = (V, E)$:
- $V$ is the set of nodes (or vertices)
- $E$ is the set of edges

For example:

- $V = \{\{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$
- $E = \{1, 2, 3, 4\}$

The neighbors of 4 are 2 and 3.

$\text{deg}(4) = 2$
Terminology

A path from node 4 to node 5, of length 4

A cycle of length 4
Terminology

A connected graph

A disconnected graph of 2 connected components
Terminology

A tree: a connected graph that has no cycle

A rooted tree
Graph Example: Internet Topology

Visualization of Internet circa 1999.
Graph Example: Facebook Social Network
Graph Example: Delta Air Lines Route Map
Directed Graph

Each directed edge \((u, v)\) goes from node \(u\) to node \(v\)

For example:

\[ E = \{1, 2, 3, 4\} \]

\[ V = \{(1, 3), (2, 3), (3, 4), (4, 2), (4, 3)\} \]

The **outgoing** neighbors of 4 are 2 and 3

\[ \text{outdeg}(4) = 2 \]

The **incoming** neighbor of 4 is 3

\[ \text{indeg}(4) = 1 \]
 Directed Graph Example: Road Network

Nodes = intersection, edge = one-way street
Directed Graph Example: Political Blogosphere

Nodes = political blog, edge = link

The political blogs and 2004 U.S. Election
How Do We Represent Graphs?

Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
How Do We Represent Graphs?

**Adjacency Matrix**

<table>
<thead>
<tr>
<th>Source</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The graph shows the adjacency matrix representation of a directed graph. The matrix entries indicate the presence of edges between nodes: a 1 indicates an edge, and a 0 indicates no edge. For example, there is an edge from node 1 to node 3, as indicated by the 1 in the matrix at the intersection of row 1 and column 3.
How Do We Represent Graphs?

**Adjacency List**

How would you modify this for directed graphs?
### Trade-off

**n nodes and m edges**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Edge membership**

\( e = \{u, v\} \in E? \)

\( O(1) \)

**Neighbor query**

Give me neighbors of \( v \)

\( O(n) \)

\( O(\text{deg}(v)) \)

**Space requirements**

\( O(n^2) \)

\( O(m + n) \)

- Generally better for **sparse** graphs

We’ll assume this representation for the rest of the class
1. Basic definitions and applications

2. Breadth-First Search

3. Depth-First Search

4. Topological ordering
Breadth-First Search

Exploring the world with a bird’s-eye view

Not been there yet
Breadth-First Search

Exploring the world with a bird’s-eye view
Breadth-First Search
Exploring the world with a bird’s-eye view
Breadth-First Search
Exploring the world with a bird’s-eye view

Not been there yet
Can reach there in 0 steps
Can reach there in 1 step
Can reach there in 2 steps
Breadth-First Search

Exploring the world with a bird’s-eye view
Breadth-First Search

Exploring the world with a bird’s-eye view

World explored!
Why Is It Called Breadth-First?

• We are implicitly building a tree:

Call this the “BFS tree”

• First we go as broadly as we can.
Timing Analysis

$n$ nodes and $m$ edges

Visiting all nodes: $O(n)$

At each node $v$, explore all of its neighbors: $O(\text{deg}(v))$

Naïve estimation:

$\text{deg}(v) \leq n - 1$

$\sum_{v \in V} \text{deg}(v) = O(n^2)$

Example:

$\sum_{v \in V} \text{deg}(v) = 18 \ll n^2 = 49$
Timing Analysis

$n$ nodes and $m$ edges

Visiting all nodes: $O(n)$

At each node $v$, explore all of its neighbors: $O(\text{deg}(v))$

**Handshaking Lemma:** For an undirected graph

$$\sum_{v \in V} \text{deg}(v) = 2m$$

Counting the edges Each edge is counted twice, from both endpoints

Running time of BFS: $O(m + n)$
Application: Finding Shortest Paths

Find a shortest path from $A$ to $G$?

Distance = Level of $G$
To find the path, backtrack from $G$ to its parent and so on
Application: Finding Connected Components

A connected component $\rightarrow$ a BFS tree

BFS forest
Exercise: Fish Into Tanks

Connected fish will fight if they are in the same tank

How to put the fish in two tanks and avoid their fighting?
BFS in Directed Graph

Not been there yet
BFS in Directed Graph

- Not been there yet
- Can reach there in 0 steps
BFS in Directed Graph

Can’t go further, but haven’t explored the entire graph

Need to restart in an unvisited node
BFS in Directed Graph

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
BFS in Directed Graph

Can’t go further, but haven’t explored the entire graph

Need to restart in an unvisited node
BFS in Directed Graph

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
- Can reach there in 0 step, first restart
BFS in Directed Graph

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
- Can reach there in 0 step, first restart
- Can reach there in 0 step, second restart
BFS in Directed Graph

Graph explored!

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
- Can reach there in 0 step, first restart
- Can reach there in 0 step, second restart
- Can reach there in 1 step, second restart
BFS Forest
Timing Analysis

$n$ nodes and $m$ edges

| Visiting all nodes: $O(n)$ | At node $v$, explore its outgoing neighbors: $O(\text{outdeg}(v))$ |

**Handshaking Lemma**: For a directed graph

$$\sum_{v \in V} \text{outdeg}(v) = \sum_{v \in V} \text{indeg}(v) = m$$

Each edge is counted once, from starting node (for outgoing degree) or ending node (for incoming degree)

**Running time of BFS**: $O(m + n)$
Exercise: Testing Strong Connectivity

A directed graph is **strongly connected** if any node $u$ is reachable from any other node $v$

**Strongly connected**

**Not strongly connected**

How to check if a directed graph is strongly connected?
Strong Connectivity Checking

Is this graph strongly connected?
Strong Connectivity Checking

**Step 1:** Can we visit every node from $A$?

BFS from $A$
Step 1: Can we visit every node from $A$?

Not been there yet
Strong Connectivity Checking

**Step 1:** Can we visit every node from $A$?

- Not been there yet
- Can reach there in 0 steps

[Diagram of a directed graph with nodes labeled A, B, C, D, E, and arrows indicating connectivity.]
Strong Connectivity Checking

**Step 1:** Can we visit every node from \( A \)?

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
Strong Connectivity Checking

**Step 1:** Can we visit every node from \( A \)?

- **Not been there yet**
- **Can reach there in 0 steps**
- **Can reach there in 1 step**
- **Can reach there in 2 steps**
Strong Connectivity Checking

**Step 1:** Can we visit every node from $A$?

- Not been there yet
- Can reach there in 0 steps
- Can reach there in 1 step
- Can reach there in 2 steps
- Can reach there in 3 steps

Yes, we can reach every node from $A$
Strong Connectivity Checking

**Step 2:** Can we visit $A$ from every node?
Strong Connectivity Checking

**Step 2:** Can we visit $A$ from every node?

Construct reverse graph: Reverse the direction of every arrow

Construct **reverse graph**: Reverse the direction of every arrow
Strong Connectivity Checking

Step 2: Can we visit $A$ from every node?

Not been there yet
Strong Connectivity Checking

**Step 2:** Can we visit $A$ from every node?

Can’t go further! Graph is not strongly connected
Agenda

1. Basic definitions and applications

2. Breadth-First Search

3. Topological ordering
Exercise: How to Wear Clothes

- left sock
- right sock
- underwear
- left shoe
- right shoe
- shirt
- pants
- belt
- tie
- jacket

A → B: Must wear A before wearing B

How can I find an order to wear clothes?
A topological ordering of a directed graph is an ordering of its nodes 1, 2, ... so that \( i \rightarrow j \) only if \( i < j \).
Directed Acyclic Graph (DAG)

DAG: a directed graph that contains no directed cycle

Topological ordering is only possible in DAGs

Is a DAG

Is not a DAG
Problem: How to Detect Cycles?

Given a directed graph, check if it’s a DAG
If not, output a directed cycle

Application: Deadlock detection

How to realize that we have a deadlock?

\[ P \rightarrow R \] : Process $P$ is waiting for resource $R$

\[ R \rightarrow P \] : Resource $R$ is being used by process $P$
Checking If A Directed Graph Has Cycle

![Diagram of a directed graph with nodes A, B, C, D, and E, showing directed edges between them.](image-url)
Checking If A **Directed** Graph Has Cycle

Perform DFS

![Directed Graph Diagram]

- **Unvisited**
- **Visited, but not fully explored**
- **Finished**
Checking If A **Directed** Graph Has Cycle

![Directed Graph Diagram]

- **Unvisited**
- **Finished**
- **Visited, but not fully explored**
Checking If A Directed Graph Has Cycle

- Unvisited
- Visited, but not fully explored
- Finished

Diagram:
- Start at A
- A → B
- B → C
- C → D
- D → E
- E → C
Checking If A Directed Graph Has Cycle

![Directed Graph Diagram]

- **Unvisited**
- **Visited, but not fully explored**
- **Finished**
Checking If a Directed Graph Has Cycle
Checking If A **Directed** Graph Has Cycle

![Diagram of a directed graph with nodes A, B, C, D, E and edges A → B, B → C, C → E, E → D, D → B, start at A. Nodes are colored: Unvisited, Visited, but not fully explored, and Finished.](image)

- **Unvisited**
- **Visited, but not fully explored**
- **Finished**
Checking If A Directed Graph Has Cycle

Start

A

B

C

D

E

Unvisited

Visited, but not fully explored

Finished
Checking If A *Directed* Graph Has Cycle

![Directed Graph Diagram]

- Unvisited
- Visited, but not fully explored
- Finished

---

65
Checking If A Directed Graph Has Cycle

There’s an outgoing neighbor under processing

DFS tree

- Unvisited
- Visited, but not fully explored
- Finished

Has cycle
Problem: How to Topologically Sort?

Given DAG, output a topological order

Application: Package Installation

How to install packages without violating dependency requirements?
Topological Sort in Clothing Graph

Maintain indegrees of the nodes
Topological Sort in Clothing Graph

Track nodes of indegree 0
Topological Sort in Clothing Graph

- Left sock
- Right sock
- Underwear
- Shirt
- Left shoe
- Right shoe
- Belt
- Pants
- Tie
- Jacket

Order: 1

Active node of indegree 0

Unprocessed node

Node under processing

Finished node
Topological Sort in Clothing Graph

order: 1

- left sock
- right sock
- underwear
- left shoe
- right shoe
- belt
- pants
- shirt
- tie
- jacket

Active node of indegree 0
Unprocessed node
Node under processing
Finished node
Topological Sort in Clothing Graph

- **left sock** (order: 1)
- **right sock** (order: 1)
- **underwear** (order: 2)
- **shirt**
- **pants**
- **tie** (order: 2)
- **jacket**

**Legend:**
- Orange: Active node of indegree 0
- Purple: Unprocessed node
- Green: Node under processing
- White: Finished node
Topological Sort in Clothing Graph

- left sock
- right sock
- underwear
- shirt
- left shoe
- right shoe
- pants
- belt
- tie
- jacket

Order:
- left sock: 1
- right sock: 0
- underwear: 0
- shirt: 0
- left shoe: 0
- right shoe: 0
- pants: 1
- belt: 2
- tie: 1
- jacket: 2

Active node of indegree 0
Unprocessed node
Node under processing
Finished node
Topological Sort in Clothing Graph
Topological Sort in Clothing Graph

- **left sock**
- **right sock**
- **underwear**
- **shirt**
- **left shoe**
- **right shoe**
- **pants**
- **belt**
- **tie**
- **jacket**

**Order:**
- **left sock:** order: 1
- **right sock:** order: 2
- **underwear:** order: 3
- **shirt:**
- **left shoe:**
- **right shoe:**
- **pants:**
- **belt:**
- **tie:** order: 2
- **jacket:** order: 3

**Legend:**
- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

- **left sock**
  - order: 1
- **right sock**
  - order: 4
- **underwear**
  - order: 2
- **shirt**
  - order: 3
- **pants**
- **belt**
  - order: 3
- **tie**
  - order: 2
- **jacket**

Legend:
- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

- **Active node of indegree 0**: left sock, right sock, underwear, shirt, pants, tie, jacket
- **Unprocessed node**: left shoe, right shoe, belt
- **Node under processing**: None
- **Finished node**: None

Order:
- **order: 1**: left sock
- **order: 2**: shirt
- **order: 3**: pants
- **order: 4**: right sock
Topological Sort in Clothing Graph

- left sock
  - order: 1
- right sock
  - order: 4
- underwear
- shirt
  - order: 2
- pants
  - order: 3
- belt
  - order: 5
- left shoe
  - order: 5
- right shoe
- jacket

Legend:
- Orange: Active node of indegree 0
- Purple: Unprocessed node
- Green: Node under processing
- White: Finished node
Topological Sort in Clothing Graph

Active node of indegree 0
Unprocessed node
Node under processing
Finished node
Topological Sort in Clothing Graph

- **left sock** (order: 1)
- **right sock** (order: 4)
- **underwear** (order: 2)
- **pants** (order: 3)
- **shirt** (order: 6)
- **left shoe** (order: 5)
- **right shoe**
- **belt**
- **tie** (order: 1)
- **jacket** (order: 2)

- **Active node of indegree 0**
- **Unprocessed node**
- **Node under processing**
- **Finished node**
Topological Sort in Clothing Graph

order: 1

order: 2

order: 3

order: 4

order: 5

order: 6
Topological Sort in Clothing Graph

Order: 1
- Underwear
  - Order: 2
  - Pants
    - Order: 3
    - Tie
      - Order: 4
      - Belt
        - Order: 5
        - Left shoe
          - Order: 6
          - Right shoe
            - Order: 7
            - Right sock
              - Order: 1
              - Left sock

Legend:
- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

- left sock (order: 1)
- right sock (order: 4)
- underwear (order: 2)
- pants (order: 3)
- shirt (order: 5)
- left shoe (order: 6)
- right shoe (order: 7)
- belt
- tie
- jacket

- Active node of indegree 0
- Node under processing
- Unprocessed node
- Finished node
Topological Sort in Clothing Graph

- left sock (order: 1)
- right sock (order: 4)
- underwear
- shirt (order: 8)
- pants (order: 3)
- tie (order: 6)
- left shoe (order: 5)
- right shoe (order: 7)
- belt

Diagram:

- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph
Topological Sort in Clothing Graph

- left sock (order: 1)
- right sock (order: 4)
- underwear
- shirt (order: 8)
- pants (order: 3)
- left shoe
- right shoe
- belt (order: 7)
- tie (order: 9)
- jacket

Symbols:
- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

- left sock (order: 1)
- right sock (order: 4)
- underwear
- shirt (order: 8)
- pants (order: 3)
- tie (order: 9)
- jacket (order: 10)
- left shoe (order: 5)
- right shoe
- belt (order: 7)

Legend:
- Active node of indegree 0
- Unprocessed node
- Node under processing
- Finished node
Topological Sort in Clothing Graph

Finish topological sorting!