Learning Objectives

At the end of the class you should be able to:

- explain how a generic searching algorithm works
- demonstrate how depth-first search will work on a graph
- demonstrate how breadth-first search will work on a graph
- predict the space and time requirements for depth-first and breadth-first searches
Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a **frontier** of paths from the start node that have been explored.
Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a **frontier** of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.
Problem Solving by Graph Searching

- **Start node**
- **Explored nodes**
- **Unexplored nodes**
- **Ends of paths on frontier**
Graph Search Algorithm

**Input:** a graph, a set of start nodes, Boolean procedure \( \text{goal}(n) \) that tests if \( n \) is a goal node.

\( \text{frontier} := \{ \langle s \rangle : s \text{ is a start node} \} \)

**while** \( \text{frontier} \) is not empty:

- **select** and **remove** path \( \langle n_0, \ldots, n_k \rangle \) from \( \text{frontier} \)
- **if** \( \text{goal}(n_k) \)
  - **return** \( \langle n_0, \ldots, n_k \rangle \)
- **for every** neighbor \( n \) of \( n_k \)
  - **add** \( \langle n_0, \ldots, n_k, n \rangle \) to \( \text{frontier} \)

**end while**
Graph Search Algorithm

- Which value is selected from the frontier at each stage defines the search strategy.
- The neighbors define the graph.
- *goal* defines what is a solution.
- If more than one answer is required, the search can continue from the return.
Often we don’t want any solution, but the best solution or optimal solution.

Costs on arcs give costs on paths. We want the least-cost path to a goal.
Depth-first Search

- Depth-first search treats the frontier as a stack.
- It always selects one of the last elements added to the frontier.
- If the list of paths on the frontier is \([p_1, p_2, \ldots]\)
  - \(p_1\) is selected. Paths that extend \(p_1\) are added to the front of the stack (in front of \(p_2\)).
  - \(p_2\) is only selected when all paths from \(p_1\) have been explored.
Illustrative Graph - Depth-first search

Start node is at the top.
Illustrative Graph — Depth-first Search
Which shaded goal will depth-first search find first?
Complexity of Depth-first Search

- Does depth-first search guarantee to find the path with fewest arcs?
Does depth-first search guarantee to find the path with fewest arcs?

What happens on infinite graphs or on graphs with cycles if there is a solution?
Does depth-first search guarantee to find the path with fewest arcs?

What happens on infinite graphs or on graphs with cycles if there is a solution?

What is the time complexity as a function of length of the path selected?
Does depth-first search guarantee to find the path with fewest arcs?

What happens on infinite graphs or on graphs with cycles if there is a solution?

What is the time complexity as a function of length of the path selected?

What is the space complexity as a function of length of the path selected?
Does depth-first search guarantee to find the path with fewest arcs?

What happens on infinite graphs or on graphs with cycles if there is a solution?

What is the time complexity as a function of length of the path selected?

What is the space complexity as a function of length of the path selected?

How does the goal affect the search?
Breadth-first Search

- **Breadth-first search** treats the frontier as a queue.
- It always selects one of the earliest elements added to the frontier.
- If the list of paths on the frontier is \([p_1, p_2, \ldots, p_r]\):
  - \(p_1\) is selected. Its neighbors are added to the end of the queue, after \(p_r\).
  - \(p_2\) is selected next.
Illustrative Graph - Breadth-first search

Start node is at the top.
Illustrative Graph — Breadth-first Search
Which shaded goal will breadth-first search find first?
Does breadth-first search guarantee to find the path with fewest arcs?
Does breadth-first search guarantee to find the path with fewest arcs?

What happens on infinite graphs or on graphs with cycles if there is a solution?
Complexity of Breadth-first Search

- Does breadth-first search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the length of the path selected?
Complexity of Breadth-first Search

- Does breadth-first search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the length of the path selected?
- What is the space complexity as a function of the length of the path selected?
Complexity of Breadth-first Search

- Does breadth-first search guarantee to find the path with fewest arcs?
- What happens on infinite graphs or on graphs with cycles if there is a solution?
- What is the time complexity as a function of the length of the path selected?
- What is the space complexity as a function of the length of the path selected?
- How does the goal affect the search?
Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

\[ \text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle) \]

An optimal solution is one with minimum cost.
Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

\[
\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)
\]

An optimal solution is one with minimum cost.

At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.

The frontier is a priority queue ordered by path cost.

The first path to a goal is
Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

\[
\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)
\]

An optimal solution is one with minimum cost.

At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.

The frontier is a priority queue ordered by path cost.

The first path to a goal is a least-cost path to a goal node.

When arc costs are equal.
Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.

\[
\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)
\]

An optimal solution is one with minimum cost.

At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.

The frontier is a priority queue ordered by path cost.

The first path to a goal is a least-cost path to a goal node.

When arc costs are equal \(\implies\) breadth-first search.
Summary of Search Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Complete</th>
<th>Halts</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Minimal $cost(p)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complete** — guaranteed to find a solution if there is one (for graphs with finite number of neighbours, even on infinite graphs)

**Halts** — on finite graph (perhaps with cycles).

**Space** — as a function of the length of current path
## Summary of Search Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Complete</th>
<th>Halts</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td>No</td>
<td>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Minimal $cost(p)$</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
</tbody>
</table>

**Complete** — guaranteed to find a solution if there is one (for graphs with finite number of neighbours, even on infinite graphs)

**Halts** — on finite graph (perhaps with cycles).

**Space** — as a function of the length of current path.