Learning Objectives

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

- justify why depth-bounded search is useful
- demonstrate how iterative-deepening works for a particular problem
- demonstrate how depth-first branch-and-bound works for a particular problem
### Summary of Search Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier</th>
<th>Complete</th>
<th>Halts</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first w/o CP</td>
<td>Last added</td>
<td>No</td>
<td>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Depth-first w CP</td>
<td>Last added</td>
<td>No</td>
<td>Yes</td>
<td>Linear</td>
</tr>
<tr>
<td>Depth-first w MPP</td>
<td>Last added</td>
<td>No</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Breadth-first w/o MPP</td>
<td>First added</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Breadth-first w MPP</td>
<td>First added</td>
<td>Yes</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Best-first w/o MPP</td>
<td>Min $h(p)$</td>
<td>No</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Best-first w MPP</td>
<td>Min $h(p)$</td>
<td>No</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>$A^*$ w/o MPP</td>
<td>Min $f(p)$</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>$A^*$ w MPP</td>
<td>Min $f(p)$</td>
<td>Yes</td>
<td>Yes</td>
<td>Exp</td>
</tr>
</tbody>
</table>

**Complete** — if there a path to a goal, it can find one, even on infinite graphs.

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

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

Assume graph satisfies the assumptions of $A^*$ proof + monotonocity.
A bounded depth-first search takes a bound (cost or depth) and does not expand paths that exceed the bound.
  ▶ explores part of the search graph
  ▶ uses space linear in the depth of the search.

How does this relate to other searches?

How can this be extended to be complete?
Which shaded goal will a depth-bounded search find first?
Iterative-deepening search:

- Start with a bound \( b = 0 \).
- Do a bounded depth-first search with bound \( b \).
- If a solution is found return that solution.
- Otherwise increment \( b \) and repeat.
Iterative-deepening search:

- Start with a bound $b = 0$.
- Do a bounded depth-first search with bound $b$.
- If a solution is found return that solution.
- Otherwise increment $b$ and repeat.

This will find the same first solution as what other method?
Iterative-deepening search:

- Start with a bound $b = 0$.
- Do a bounded depth-first search with bound $b$.
- If a solution is found return that solution.
- Otherwise increment $b$ and repeat.

This will find the same first solution as what other method?

How much space is used?
Iterative-deepening search:

- Start with a bound $b = 0$.
- Do a bounded depth-first search with bound $b$.
- If a solution is found return that solution.
- Otherwise increment $b$ and repeat.

This will find the same first solution as what other method?

How much space is used?

What happens if there is no path to a goal?
Iterative-deepening search:

- Start with a bound $b = 0$.
- Do a bounded depth-first search with bound $b$.
- If a solution is found return that solution.
- Otherwise increment $b$ and repeat.

This will find the same first solution as what other method?

How much space is used?

What happens if there is no path to a goal?

Surely recomputing paths is wasteful!!!
Complexity with solution at depth $k$ & branching factor $b$:

<table>
<thead>
<tr>
<th>level</th>
<th>breadth-first</th>
<th>iterative deepening</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$k$</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$k - 1$</td>
<td>$b^2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k-1$</td>
<td>1</td>
<td>2</td>
<td>$b^{k-1}$</td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
<td>1</td>
<td>$b^k$</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\geq b^k \leq (b-1)b^k$
Complexity with solution at depth $k$ & branching factor $b$:

<table>
<thead>
<tr>
<th>level</th>
<th>breadth-first</th>
<th>iterative deepening</th>
<th># nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$k$</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$k - 1$</td>
<td>$b^2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$k - 1$</td>
<td>1</td>
<td>2</td>
<td>$b^{k-1}$</td>
</tr>
<tr>
<td>$k$</td>
<td>1</td>
<td>1</td>
<td>$b^k$</td>
</tr>
<tr>
<td>total</td>
<td>$\geq b^k$</td>
<td>$\leq b^k \left(\frac{b}{b-1}\right)^2$</td>
<td></td>
</tr>
</tbody>
</table>
Depth-first Branch-and-Bound

- Combines depth-first search with heuristic information.
- Finds optimal solution.
- Most useful when there are multiple solutions, and we want an optimal one.
- Uses the space of depth-first search.
Depth-first Branch-and-Bound

Suppose we want to find a single optimal solution.
  - Suppose *bound* is the cost of the lowest-cost path found to a goal so far.
  - What if the search encounters a path *p* such that
    \[ \text{cost}(p) + h(p) \geq \text{bound} \]?
Suppose we want to find a single optimal solution.

- Suppose *bound* is the cost of the lowest-cost path found to a goal so far.
- What if the search encounters a path *p* such that
  \[ \text{cost}(p) + h(p) \geq \text{bound} \]?
  *p* can be pruned.
- What can we do if a non-pruned path to a goal is found?
Suppose we want to find a single optimal solution.

- Suppose $bound$ is the cost of the lowest-cost path found to a goal so far.
- What if the search encounters a path $p$ such that $\text{cost}(p) + h(p) \geq bound$? $p$ can be pruned.
- What can we do if a non-pruned path to a goal is found? $bound$ can be set to the cost of $p$, and $p$ can be remembered as the best solution so far.
- Why should this use a depth-first search?
Suppose we want to find a single optimal solution.

- Suppose \( \text{bound} \) is the cost of the lowest-cost path found to a goal so far.

- What if the search encounters a path \( p \) such that \( \text{cost}(p) + h(p) \geq \text{bound} \)?
  
  \( p \) can be pruned.

- What can we do if a non-pruned path to a goal is found? \( \text{bound} \) can be set to the cost of \( p \), and \( p \) can be remembered as the best solution so far.

- Why should this use a depth-first search?
  
  Uses linear space.

- What can be guaranteed when the search completes?
Suppose we want to find a single optimal solution.

- Suppose \textit{bound} is the cost of the lowest-cost path found to a goal so far.
- What if the search encounters a path \( p \) such that \( \text{cost}(p) + h(p) \geq \text{bound} \)? \( p \) can be pruned.
- What can we do if a non-pruned path to a goal is found? \( \text{bound} \) can be set to the cost of \( p \), and \( p \) can be remembered as the best solution so far.
- Why should this use a depth-first search? Uses linear space.
- What can be guaranteed when the search completes? It has found an optimal solution.
Suppose we want to find a single optimal solution.

- Suppose *bound* is the cost of the lowest-cost path found to a goal so far.
- What if the search encounters a path *p* such that \( \text{cost}(p) + h(p) \geq \text{bound} \)?
  - *p* can be pruned.
- What can we do if a non-pruned path to a goal is found?
  - *bound* can be set to the cost of *p*, and *p* can be remembered as the best solution so far.
- Why should this use a depth-first search?
  - Uses linear space.
- What can be guaranteed when the search completes?
  - It has found an optimal solution.
- How should the bound be initialized?
The bound can be initialized to $\infty$.

The bound can be set to an estimate of the optimal path cost. After depth-first search terminates either:
The bound can be initialized to $\infty$.

The bound can be set to an estimate of the optimal path cost. After depth-first search terminates either:

- A solution was found.
- No solution was found, and no path was pruned
- No solution was found, and a path was pruned.
Which shaded goals will be best solutions so far?