Search

Chao Lan
Introduction
Find a path = Generate a tree (by expanding nodes)
Important Terminology and Concepts

- **expand a node** = consider adding it to the solution path.
- **frontier** = a set of nodes that can be expanded.
- **closed set** = a set of nodes that have been expanded.
### Example

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frontier</strong></td>
<td>Arad</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td><strong>Closed Set</strong></td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

- **Frontier**: Arad
- **Closed Set**: /
## Example

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>Arad</td>
<td>Sibiu, Timi, Zerind</td>
<td></td>
</tr>
<tr>
<td>Closed Set</td>
<td>/</td>
<td>Arad</td>
<td></td>
</tr>
</tbody>
</table>

![Tree Diagram]
## Example

<table>
<thead>
<tr>
<th>Frontier</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arad</td>
<td>Sibiu, Timi, Zerind</td>
<td>Faga, Orad, RV, Timi, Zerind</td>
</tr>
<tr>
<td>Closed Set</td>
<td>/</td>
<td>Arad</td>
<td>Arad, Sibiu</td>
</tr>
</tbody>
</table>

### Graph Representation

```
    Arad
     /   
   Sibiu /     
  /     /      
Arad   Fagaras  Oradea
        /   
       /     
      Rimnicu Vilea
```

```
    Timisoara
    /      
   Zerind

    Arad
     /   
   Lugoj
```

```
    Arad
    /   
   Oradea
```
Terminate search until target node is expanded.

<table>
<thead>
<tr>
<th></th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontier</td>
<td>Arad</td>
<td>Sibiu, Timi, Zerind</td>
<td>Faga, Orad, RV, Timi, Zerind</td>
</tr>
<tr>
<td>Closed Set</td>
<td>/</td>
<td>Arad</td>
<td>Arad, Sibiu</td>
</tr>
</tbody>
</table>
Five Basic Search Algorithms
Five Search Algorithms: **Uninformed** vs **Informed**

- Breadth-First
- Depth-First
- Uniform-Cost
- Greedy
- A* Search
1. Breadth-First Search (BFS)

Idea: always expand node at the shallowest depth.

Depth of a node is the number of edges from that node to the root node, e.g., $D(B)=1$, $D(F)=2$. 
Exercise: BFS from Arad to RV

Break ties by alphabetical order.
Exercise: BFS from Arad to RV (Solution)

1. Arad - \{S(1), T(1), Z(1)\}
2. Sibiu - \{F(2), O(2), R(2), T(1), Z(1)\}
3. Timi - \{L(2), F(2), Os(2), R(2), Z(1)\}
4. Zerind - \{Oz(2), L(2), F(2), Os(2), R(2)\}
5. Fagaras - \{B(3), Oz(2), L(2), Os(2), R(2)\}
6. Lugoj - \{M(3), B(3), Oz(2), Os(2), R(2)\}
7. Oradea_s - \{ M(3), B(3), Oz(2), R(2)\}
8. Expand RV, stop!

Break ties by alphabetical order. After step 7, we skip Oz because O has been expanded through Sibiu.
2. Depth-First Search (DFS)

Idea: always expand node at the deepest depth.
Exercise: DFS from Arad to RV

Break ties by alphabetical order.
Exercise: DFS from Arad to RV (Solution)

1. Arad - \{S(1), Ta(1), Z(1)\}
2. Sibiu - \{F(2), O(2), Rs(2), Ta(1), Z(1)\}
3. Fagaras - \{B(3), O(2), Rs(2), Ta(1), Z(1)\}
4. Bucharest - \{P(4), O(2), Rs(2), Ta(1), Z(1)\}
5. Pitesti - \{Rp(5), C(5), O(2), Rs(2), Ta(1), Z(1)\}
   then C(5), D(6), M(7), L(8), Tl(9), ….
10. Timi_L - \{Rp(5), O(2), Rs(2), Ta(1), Z(1)\}
11. Expand Rp, stop!

Break ties by alphabetical order.
3. Uniform-Cost Search (UCS)

Idea: always expand node n with the lowest path-cost $g(n)$.

- **path cost** is sum of step costs (from root note to examined node)
- **step cost** is edge number
Exercise: UCS from Sibiu to Bucharest

Break ties by alphabetical order.
Exercise: UCS from Sibiu to Bucharest

1. Sibiu - \{ F(99), RV(80) \}

2. RV - \{ F(99), P_{sr}(177) \}

3. Fagaras - \{ P_{sr}(177), B_{sf}(310) \}

4. Pitesti - \{ B_{srp}(278), B_{sf}(310) \}

5. Expand Bucharest through srp (278). Stop!

Break ties by alphabetical order.
Uninformed Search vs Informed Search

- Breadth-First
- Depth-First
- Uniform-Cost
- Greedy
- A* Search
4. Greedy Search (GS)

Idea: always expand the most promising node \( n \)

- promisingness is measured by a **heuristic function** \( h(n) \) -- smaller is better
- e.g., \( h(n) = \) straight-line distance from node \( n \) to the target node Bucharest

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Drobota</td>
<td>242</td>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: GS from Arad to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: GS from Arad to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>GS1</th>
<th>City</th>
<th>GS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Drobota</td>
<td>242</td>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

1. Arad - \{ S(253), T(329), Z(374) \}
2. Sibiu - \{ F(176), R(193), T(329), Z(374), O(380) \}
3. Faga - \{ B(0), R(193), T(329), Z(374), O(380) \}
4. expand Bucharest, stop!
Which node will greedy algorithm expand?

- Arad to Bucharest: \( h = 100 \)
- Arad to Fagaras: \( h = 176 \)
What if we additionally consider path cost?

<table>
<thead>
<tr>
<th></th>
<th>Path Cost Calculation</th>
<th>h Value</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>140 + 80 + 97 = 317</td>
<td>100</td>
<td>Bucharest (417)</td>
</tr>
<tr>
<td>Pitesti</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arad</td>
<td>140 + 99 = 239</td>
<td>176</td>
<td>Bucharest (415)</td>
</tr>
</tbody>
</table>
A* Search

Idea: always expand node $n$ with the smallest (heuristic + path cost).

<table>
<thead>
<tr>
<th>City</th>
<th>Heuristic</th>
<th>Path Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>380</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>374</td>
</tr>
</tbody>
</table>
Summary of Five Search Algorithms

1. Breadth-First Search
2. Depth-First Search
3. Uniform-Cost Search
4. Greedy Search
5. A* Search
Three Basic Properties of Search Algorithms
Three Properties

Q1: if there is a solution, is an algorithm guaranteed to find it?
   - yes → algorithm is complete

Q2: if an algorithm finds a solution, is it guaranteed to be optimal?
   - an optimal solution is a path with the lowest path cost.
   - yes → algorithm is optimal

Q3: how many nodes need to be expanded by an algorithm to find a solution?
   - time complexity of the algorithm (big O)
Exercise: True or False?

Breadth-first search cannot find a solution from WA to T, thus it is not complete.
Exercise: True or False?

Si-Fa-Bu is the optimal path because it travels through the fewest edges.
Exercise

If we apply BFS to find a path from A to G, how many nodes will be expanded?
Branching factor is the effective number of child nodes (per node), e.g., $b=2$. 
Complexity analysis is a very complicated task.

We will make some assumptions to simplify discussion…

1. breadth of the tree is finite
2. depth of the tree may be infinite
3. no revisit of expanded nodes
4. step cost may be zero
1. Breadth-First Search

Q1: complete?

Q2: optimal?

Q3: time complexity?
1. Breadth-First Search

Q1: complete?
- yes

Q2: optimal?
- no

Q3: time complexity?
- $O(b^d)$

b is branching factor, d is depth of the shallowest solution.
2. Depth-First Search

Q1: complete?

Q2: optimal?

Q3: time complexity?
2. Depth-First Search

Q1: complete?
- yes, if tree is finite

Q2: optimal?
- no

Q3: time complexity?
- $O(b^m)$

$m$ is maximum depth of the search tree. $O(b^m)$ means searching through the entire tree.
Exercise: True or False?

Depth-first search finds a solution for the following infinite tree, thus it is complete.
3. Uniform-Cost Search

Q1: complete?

Q2: optimal?

Q3: time complexity?
3. Uniform-Cost Search

Q1: complete?
- yes, if step cost > e

Q2: optimal?
- yes

Q3: time complexity?
- \( O(b^{(1+[C^*/e])}) \)

Roughly speaking, \((1+[C^*/e])\) is depth of the target node, where \(C^*\) is its path-cost, \(e\) is step cost.
4. Greedy Search

Q1: complete?

Q2: optimal?

Q3: time complexity?
4. Greedy Search

Q1: complete?
- yes, if tree is finite (+no node revisit)

Q2: optimal?
- no

Q3: time complexity?
- \( O(b^m) \)
5. A* Search

Q1: complete?

Q2: optimal?

Q3: time complexity?
5. A* Search

Q1: complete?
   - yes, if tree is finite or step cost > ε

Q2: optimal?
   - yes, if heuristic is consistent

Q3: time complexity?
   - O(b^m)
A heuristic is **consistent** if it satisfies the following triangular inequality

\[ h(n) \leq c(n, a, n') + h(n') \]

c\((n,a,n')\) is step cost to move from node \(n\) to node \(n'\).
Q: is straight-line distance a consistent heuristic?

A heuristic is consistent if it satisfies the following triangular inequality

\[ h(n) \leq c(n, a, n') + h(n') \]
# Summary of Algorithm Properties

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS</th>
<th>UCS</th>
<th>GS</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>(finite tree)</td>
<td></td>
<td></td>
<td>(cost&gt;e)</td>
<td>(finite tree)</td>
<td>(finite tree or cost&gt;e)</td>
</tr>
<tr>
<td>Optimal?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(consistent heuristic)</td>
</tr>
<tr>
<td>Time Complexity</td>
<td>$O(b^d)$</td>
<td>$O(b^m)$</td>
<td>$O(b^{1+C*/e})$</td>
<td>$O(b^m)$</td>
<td>$O(b^m)$</td>
</tr>
</tbody>
</table>
Applications of Search
Routing Problem
8-Puzzle Problem

Start State

Goal State
Automatic Assembly Sequencing Problem
VLSI Layout Problem