Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return false;
        } else {
            Node node = frontier.selectOne();
            if (p.isGoalState(node.getState())) {
                return node.getSolution();
            }
            explored.add(node);
            for (Node n : node.expand()) {
                if (!explored.contains(n)) {
                    frontier.add(n);
                }
            }
        }
    }
}
Kinds of Strategies
Kinds of Strategies

- Uninformed
  - Only considers the structure of the search space given by the transition function
  - Does not consider the state information associated with a node
Kinds of Strategies

Uninformed
- Only considers the structure of the search space given by the transition function
- Does not consider the state information associated with a node

Heuristic
- Uses the state information to help select promising nodes to expand
b: branching factor
b: branching factor

d: depth of shallowest goal
b: branching factor

d: depth of shallowest goal

m: length of longest path
Uninformed Strategies
Uninformed Strategies
A kind of uninformed search that

Is guaranteed to find a goal state if one exists, and the branching factor is finite

Is guaranteed to find a goal that has smallest depth
A kind of uninformed search that

- Is guaranteed to find a goal state if one exists, and the branching factor is finite
- Is guaranteed to find a goal that has smallest depth

What is breadth-first search?
A data structure used to represent the frontier in breadth-first search that allows “select_one” to be performed in constant time.
$200

A data structure used to represent the frontier in breadth-first search that allows “select_one” to be performed in constant time.

What is a FIFO (first-in first-out) queue?
$300
$300

The time complexity of breadth first-search, where b is the maximum branching factor and d is the depth of the shallowest goal.
$300

The time complexity of breadth first-search, where $b$ is the maximum branching factor and $d$ is the depth of the shallowest goal.

What is $O(b^d)$?
The space complexity of breadth first-search, where $b$ is the maximum branching factor and $d$ is the depth of the shallowest goal.
The space complexity of breadth first-search, where $b$ is the maximum branching factor and $d$ is the depth of the shallowest goal.

What is $O(b^d)$?
If the state space has this property, then breadth-first search may fail to find a solution, even when one exists.
If the state space has this property, then breadth-first search may fail to find a solution, even when one exists.

What is having an infinite branching factor?
$y = b^d \quad (b = 10)$
<table>
<thead>
<tr>
<th>Time (1(\mu)s/node)</th>
<th>Storage (1KB/node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 years</td>
<td>1 Exabyte</td>
</tr>
<tr>
<td>11 days</td>
<td>1 Petabyte</td>
</tr>
<tr>
<td>17 mins</td>
<td>1 TB</td>
</tr>
<tr>
<td>1 sec</td>
<td>1 GB</td>
</tr>
<tr>
<td>1 ms</td>
<td>1 MB</td>
</tr>
</tbody>
</table>

\[
y = b^d \quad (b = 10)
\]
A kind of uninformed search that

Is guaranteed to find a goal state only if the maximum depth of the tree (m) is finite

The maximum size of the frontier is mb, where b is the maximum branching factor
A kind of uninformed search that

Is guaranteed to find a goal state only if the maximum depth of the tree (m) is finite

The maximum size of the frontier is mb, where b is the maximum branching factor

What is depth-first search?
$200
$200

A data structure used to represent the frontier in depth-first search that allows “select_one” to be performed in constant time.
A data structure used to represent the frontier in depth-first search that allows “select_one” to be performed in constant time.

What is a stack (or what is a LIFO last-in first-out queue)?
$300
$300

The time complexity of depth-first-search, where $b$ is the branching factor, $d$ is the depth of the shallowest goal, and $m$ is the maximum depth of node.
The time complexity of depth first-search, where \( b \) is the branching factor, \( d \) is the depth of the shallowest goal, and \( m \) is the maximum depth of node

What is \( O(b^m) \)?
$400
$400

You are usually off performing tree depth first search instead of graph depth first search because this data structure can grow to size $b^m$. 
You are usually off performing tree depth first search instead of graph depth first search because this data structure can grow to size $b^m$.

What is the explored list?
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return false;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());

    while (true) {
        if (frontier.isEmpty()) {
            return false;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
$500
This simple trick enables breadth-first search to be complete, even if the maximum depth m is infinite.
This simple trick enables breadth-first search to be complete, even if the maximum depth $m$ is infinite.

What is iterative deepening?
Iterative Deepening
Iterative Deepening

For $d = 1, 2, 3, \ldots$

Do DFS to depth $d$

Until a goal is found.
<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of nodes expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(b)</td>
</tr>
<tr>
<td>2</td>
<td>(b + b^2)</td>
</tr>
<tr>
<td>3</td>
<td>(b + b^2 + b^3)</td>
</tr>
<tr>
<td>(d)</td>
<td>(b + b^2 + b^3 + \cdots b^d)</td>
</tr>
<tr>
<td>Depth</td>
<td>Number of nodes expanded</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>1</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>$b + b^2$</td>
</tr>
<tr>
<td>3</td>
<td>$b + b^2 + b^3$</td>
</tr>
<tr>
<td>$d$</td>
<td>$b + b^2 + b^3 + \ldots + b^d$</td>
</tr>
</tbody>
</table>

$$(d)b + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d$$
<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of nodes expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$b$</td>
</tr>
<tr>
<td>2</td>
<td>$b + b^2$</td>
</tr>
<tr>
<td>3</td>
<td>$b + b^2 + b^3$</td>
</tr>
<tr>
<td>$d$</td>
<td>$b + b^2 + b^3 + \cdots + b^d$</td>
</tr>
</tbody>
</table>

$$(d)b + (d - 1)b^2 + (d - 2)b^3 + \cdots + (1)b^d = O(b^d)$$
# Uninformed Strategies

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS (tree)</th>
<th>IDS (tree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Optimal?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^m)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(bm)$</td>
<td>$O(bd)$</td>
</tr>
</tbody>
</table>

* If step costs are identical (see book)
Daily Double
Daily Double

Replacing the queue in breadth-first search with this data structure enables it to find the shortest path to a goal where edges have weights.
Daily Double

Replacing the queue in breadth-first search with this data structure enables it to find a shortest path to a goal where edges have weights.

What is a heap, where the key is the cost of the path from the start node?
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return false;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Heap<Node>(p.getInitialState(), 0);
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return false;
        }
        node = frontier.selectOne();
        pathcost = node.cost;
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for ((n, c) in node.expand()) {
            newcost = pathcost + c;
            if ((!explored.contains(n) OR newcost < n.cost)) {
                n.cost = newcost
                if (frontier.contains(n))
                    frontier.changeKey(n, newcost)
                else frontier.add(n, newcost)
            }
        }
    }
}
“Exponential complexity search problems cannot be solved by uninformed methods for any but the smallest instances.”
Heuristic Strategies
heuristic |hyoō'ristik|
adjective

1. Enabling a person to discover or learn something for themselves: *a “hands-on” or interactive heuristic approach to learning.*
2. Computing proceeding to a solution by trial and error or by rules that are only loosely defined.

ORIGIN early 19th cent.: formed irregularly from Greek *heuriskein* ‘find’
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return false;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return n.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Heuristic Function

$h(n)$

Estimated cost of cheapest path from $n$ to a goal node
Heuristic function

\[ h(n) \]

Estimated cost of cheapest path from \( n \) to a goal node
Heuristic function

\[ h(n) \]

Estimated cost of cheapest path from \( n \) to a goal node
Heuristic function

\[ h(n) \]

Estimated cost of cheapest path from \( n \) to a goal node
\[ h_{SLD} \]

*(straight-line distance 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>
Frontier: $h_{SLD}(n)$
Frontier: $h_{SLD}(n)$
Frontier: $h_{SLD}(n)$
Frontier: $h_{SLD}(n)$

Romania

Urziceni

A

S

O

R

B

The diagram represents a graph with cities labeled with their connections and distances. The frontier is indicated with a specific path.
MR. GREEDY
by Roger Hargreaves
Greedy Best-First Search
Greedy Best-First Search

Completeness
Greedy Best-First Search

Completeness

Optimality
Greedy Best-First Search

- Completeness
  - \( O(b^m) \)
- Optimality

Time Complexity
Greedy Best-First Search

- Completeness
  \[ O(b^m) \]
- Optimality
  \[ O(b^m) \]
- Time Complexity
  \[ O(b^m) \]
- Space Complexity
  \[ O(b^m) \]
Evaluation function

\[ f(n) = h(n) \]
Evaluation function

\[ f(n) = h(n) \]

Estimated cost of cheapest path from \( n \) to a goal node
Evaluation function

\[ f(n) = g(n) + h(n) \]

- Known cost of path from start node to node \( n \)
- Estimated cost of cheapest path from \( n \) to a goal node
Evaluation function

\[ f(n) = g(n) + h(n) \]

Known cost of path from start node to node \( n \)

Estimated cost of cheapest path from \( n \) to a goal node
$g(n)$
Evaluation function

\[ f(n) = g(n) + h(n) \]

- Estimated cost of cheapest solution through \( n \)
$$\text{Frontier: } A$$

$$f(n) = g(n) + h_{SLD}(n)$$
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: \[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]

\[ 140 + 253 = 393 \]
\[ 118 + 329 = 447 \]
\[ 75 + 374 = 449 \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: 

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:  
\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]

```
71
Oradea

75
Zerind

140
Sibiu

99
Fagaras

211
Rimnicu Vilcea

85
Pitesti

98
Urziceni

92
Iasi

90
Giurgiu

86
Bucharest

120
Craiova

87
Neamt

97
Fagaras

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea

Oradea

Zerind

Arad

Timisoara

Drobeta

Logoj

Mehadia

Craiova

Giurgiu

Bucharest

Urziceni

111
Arad

118
Zerind

176
Sibiu

366
Fagaras

380
Iasi

70
Mehadia

75
Logoj

146
Fagaras

239
Oradea

75
Zerind

118
Sibiu

280
Iasi

291
Iasi

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

87
Neamt

97
Fagaras

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea

Oradea

Zerind

Arad

Timisoara

Drobeta

Logoj

Mehadia

Craiova

Giurgiu

Bucharest

Urziceni

111
Arad

118
Zerind

176
Sibiu

366
Fagaras

380
Iasi

70
Mehadia

75
Logoj

146
Fagaras

239
Oradea

75
Zerind

118
Sibiu

280
Iasi

291
Iasi

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

87
Neamt

97
Fagaras

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea

Oradea

Zerind

Arad

Timisoara

Drobeta

Logoj

Mehadia

Craiova

Giurgiu

Bucharest

Urziceni

111
Arad

118
Zerind

176
Sibiu

366
Fagaras

380
Iasi

70
Mehadia

75
Logoj

146
Fagaras

239
Oradea

75
Zerind

118
Sibiu

280
Iasi

291
Iasi

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

87
Neamt

97
Fagaras

80
Sibiu

87
Logoj

85
Pitesti

86
Bucharest

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea
```
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: \( f(n) = g(n) + h_{SLD}(n) \)

\[
\begin{align*}
F & : 239 + 176 = 415 \\
P & : 317 + 100 = 417 \\
T & : 118 + 329 = 447 \\
Z & : 75 + 374 = 449 \\
C & : 366 + 160 = 526 \\
S & : 300 + 253 = 553 \\
O & : 291 + 380 = 671 \\
A & : 280 + 366 = 646 \\
\end{align*}
\]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: $f(n) = g(n) + h_{SLD}(n)$
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]

Diagram with cities and distances.
Frontier:
\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:

\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier:
\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: 
\[ f(n) = g(n) + h_{SLD}(n) \]
Frontier: $f(n) = g(n) + h_{SLD}(n)$

Romania

Oradea

Bucharest

Zerind

Arad

Timisoara

Logoj

Mehadia

Drobeta

Craiova

Giurgiu

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea

Pitesti

Bucharest

Zerind

Arad

Timisoara

Logoj

Mehadia

Drobeta

Craiova

Giurgiu

Eforie

Hirsova

Vaslui

Iasi

Neamt

Fagaras

Sibiu

Rimnicu Vilcea

Pitesti

Bucharest

Zerind

Arad

Timisoara

Logoj

Mehadia

Drobeta

Craiova

Giurgiu

Eforie
Frontier:
\[ f(n) = g(n) + h_{SLD}(n) \]
Evaluation function

\[ f(n) = g(n) + h(n) \]

- **Known cost of path from start node to node** \( n \)
- **Estimated cost of cheapest path from** \( n \) **to a goal node**
Evaluation function

\[ f(n) = g(n) + h(n) \]

---

Estimated cost of cheapest solution through \( n \)
A* Search
A* Search

Completeness ✓

Optimality ✓
A* Search

- Completeness
- Optimality

If $h(n)$ is admissible
Admissible Heuristic

Never overestimates the true cost of a solution
Admissible Heuristic

Never overestimates the true cost of a solution

\[ f(n) = g(n) + h_{SLD}(n) \]
A* Search

Completeness

Time Complexity

Optimality

Space Complexity

If $h(n)$ is admissible
A* Search

Completeness

Optimality

Time Complexity: $O(b^d)$

Space Complexity: $O(b^d)$

If $h(n)$ is admissible
A* Search

Completeness

Optimality

If \( h(n) \) is admissible

Time Complexity

Space Complexity

\( O(b^d) \)
Time / Space Advantage of A*
A* does much better than its worst case complexity when the heuristic is close to the true shortest cost to a goal.
Time / Space Advantage of A*

A* does much better than its worst case complexity when the heuristic is close to the true shortest cost to a goal.

Where $h^*$ is the optimal (perfect) heuristic, A* runs in provably polynomial time when

$$\left|h(x) - h^*(x)\right| = O(\log h^*(x))$$
Now It’s Time to Play...
Heuristic Functions

Where do good heuristic functions come from?
Heuristic Functions

Where do good heuristic functions come from?

Good question...
(See Section 3.6 for some ideas)
# Search Strategies

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>DFS</th>
<th>IDS</th>
<th>Greedy</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete?</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td>(O(b^d))</td>
<td>(O(b^m))</td>
<td>(O(b^d))</td>
<td>(O(b^m))</td>
<td>(O(b^{ed}))</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>(O(b^d))</td>
<td>(O(b^m))</td>
<td>(O(b^d))</td>
<td>(O(b^m))</td>
<td>(O(b^d))</td>
</tr>
</tbody>
</table>

* If step costs are identical
† With an admissible heuristic
Thursday: Homework Assignment – Check Website

Next Tuesday: Quiz