Uninformed Search Strategies
Uninformed Search Strategies

• What are the six things known in the problem definition?
  – A set of states
  – Start State
  – Available actions for each state
  – Successor Functions
  – Goal Test
  – Path Cost Criterion
General Graph Search: Text

1: procedure Search(G, S, goal)
2:   Inputs
3:     G: graph with nodes N and arcs A
4:     s: start node
5:     goal: Boolean function of nodes
6:   Output
7:     path from s to a node for which goal is true
8:     or ⊥ if there are no solution paths
9:   Local
10:      Frontier: set of paths
11:     Frontier := {⟨s⟩}
12:   while Frontier ≠ {} do
13:     select and remove ⟨n₀, ..., nₖ⟩ from Frontier
14:     if goal ⟨nₖ⟩ then
15:        return ⟨n₀, ..., nₖ⟩
16:     Frontier := Frontier ∪ {⟨n₀, ..., nₖ, n⟩: ⟨nₖ, n⟩ ∈ A}
17:   return ⊥

Figure 3.4: Search: generic graph searching algorithm
General Graph Search: Text

```
1: procedure Search(G, S, goal)
2: Inputs
3: G: graph with nodes N and arcs A
4: s: start node
5: goal: Boolean function of nodes
6: Output
7: path from s to a node for which goal is true
8: or ⊥ if there are no solution paths
9: Local
10: Frontier: set of paths
11: Frontier := {⟨s⟩}
12: while Frontier ≠ {} do
13:   select and remove ⟨n₀, ..., nₖ⟩ from Frontier
14:   if goal(⟨nₖ⟩) then
15:     return ⟨n₀, ..., nₖ⟩
16:   Frontier := Frontier ∪ {⟨n₀, ..., nₖ, n⟩ : ⟨nₖ, n⟩ ∈ A}
17: return ⊥
```

Figure 3.4: Search: generic graph searching algorithm
General Graph Search: Alternate

function Graph-Search(problem) returns a solution or failure
    initialize the Frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the Frontier is empty then return failure
        choose a leaf node and remove it from the Frontier
        if the node contains a goal state then return the solution
        add the node to the explored set
        expand the node, adding the resulting nodes to the frontier
        iff not in the explored set [or Frontier]
function Graph-Search(problem) returns a solution or failure
initialize the Frontier using the initial state of problem
initialize the explored set to be empty
loop do
  if the Frontier is empty then return failure
  choose a leaf node and remove it from the Frontier
  if the node contains a goal state then return the solution
  add the node to the explored set
  expand the node, adding the resulting nodes to the frontier iff not in the explored set [or Frontier]
Uninformed Search Strategies

• Uninformed strategies use only information available in the problem definition
  – Breadth-first search
  – Lowest-cost-first search
  – Depth-first search
  – Iterative deepening search
    • Depth-bounded search
Order of Nodes visited
### Breadth-First Search

<table>
<thead>
<tr>
<th>Current Node</th>
<th>Frontier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arad</strong></td>
<td></td>
</tr>
<tr>
<td>Arad</td>
<td><strong>Sibiu, Timisoara, Zerind</strong></td>
</tr>
<tr>
<td><strong>Sibiu</strong></td>
<td>Timisoara, Zerind, <em>Arad</em>, Fagaras, Oradea, Rimnicu Vilcea</td>
</tr>
<tr>
<td><strong>Timisoara</strong></td>
<td>Zerind, Fagaras, Oradea, Rimnicu Vilcea, <em>Arad</em>, Lugoj</td>
</tr>
<tr>
<td><strong>Zerind</strong></td>
<td>Fagaras, Oradea, Rimnicu Vilcea, Lugoj, <em>Arad</em>, Oradea(??)</td>
</tr>
<tr>
<td><strong>Fagaras</strong></td>
<td>Oradea, Rimnicu Vilcea, Lugoj, Bucharest, Sibiu</td>
</tr>
<tr>
<td><strong>Oradea</strong></td>
<td>Rimnicu Vilcea, Lugoj, Bucharest, Sibiu, <em>Zerind</em></td>
</tr>
<tr>
<td><strong>Rimnicu Vilcea</strong></td>
<td>Lugoj, Bucharest, <em>Craiova</em>, Pitesti, Sibiu</td>
</tr>
<tr>
<td><strong>Lugoj</strong></td>
<td>Bucharest, Craiova, Pitesti, <em>Mehadia</em>, Timisoara</td>
</tr>
<tr>
<td><strong>Bucharest</strong></td>
<td>Craiova, Pitesti,</td>
</tr>
</tbody>
</table>
# Depth-First Search

<table>
<thead>
<tr>
<th>Current Node</th>
<th>Frontier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>Arad</td>
</tr>
<tr>
<td>Zerind</td>
<td>Zerind, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Oradea</td>
<td>Oradea, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Sibiu</td>
<td>Sibiu, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>Rimnicu Vilcea, Fagaras, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Pitesti</td>
<td>Pitesti, Craiova, Fagaras, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Bucharest</td>
<td>Bucharest, Craiova, Fagaras, Timisoara, Sibiu</td>
</tr>
<tr>
<td>Bucharest</td>
<td></td>
</tr>
</tbody>
</table>
Search Strategies

• A strategy is defined by picking the order of node expansion

• Strategies are evaluated along the following dimensions:
  - completeness – does it always find a solution if one exists?
  - time complexity – number of nodes generated/expanded
  - space complexity – maximum number of nodes in memory
  - optimality – does it always find a least-cost solution
Search Strategies

• Time and space complexity are measured in terms of

b – maximum branching factor of the search tree
d – depth of the least-cost solution
m – maximum depth of the state space (may be infinite)
Properties of Breadth-First Search

• Complete??
• Time??
• Space??
• Optimal??
Properties of Breadth-First Search

• Complete?? Yes (if b is finite)
• Time?? \(1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})\), ie, exp. in d
• Space?? \(O(b^{d+1})\) (keep every node in memory)
• Optimal?? Yes (if cost = 1 per step); not optimal in general
Properties of Depth-First Search

- Complete??
- Time??
- Space??
- Optimal??
Properties of Depth-first Search

• Complete?? No: fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path
  ⇒ complete in finite spaces

• Time?? $O(b^m)$: terrible if $m$ is much larger than $d$
  but if solutions are dense, may be much faster than breadth-first

• Space?? $O(bm)$, i.e., linear space!

• Optimal?? No.
Uninformed Search Strategies

- Uninformed strategies use only information available in the problem definition
  - Breadth-first search
  - Lowest-cost-first search
  - Depth-first search
  - Iterative deepening search
    - Depth-bounded search
What did you say?

- List the type of search domain where breadth first is a good solution.
  - A bad solution

- List the type of search domain where depth first search is a good solution.
  - A bad solution
Do we need to work another example?