Problem-solving as search

- Many search problems can be converted to graph search problems
- A graph search problem can be described in terms of:
  - A set of states representing different world situations
  - Initial state
  - Goal condition
  - Operators defining valid moves between states
- Two types of search:
  - Path search: solution is a path to a goal state
  - Configuration search: solution is a state satisfying the goal condition
- Optimal solution = a solution with the optimal value
  - shortest path between the two cities, or
  - a desired n-queen configuration
Formulating a search problem

- **Search (process)**
  - The process of exploration of the search space

- **The efficiency of the search depends on:**
  - The search space and its size
  - Method used to explore (traverse) the search space
  - Condition to test the satisfaction of the search objective
  (what it takes to determine I found the desired goal object)

- **Think twice before solving the problem by search:**
  - Choose the **search space** and the **exploration policy**

---

Search process

- Exploration of the state space through successive application of operators from the initial state
- A **search tree** = a kind of (search) exploration trace, branches corresponding to explored paths, and leaf nodes corresponding to exploration fringe
Search tree

- A search tree = a (search) exploration trace
  - It is different from the graph defining the problem
  - States can repeat in the search tree
General search algorithm

General-search \((\text{problem, strategy})\)
initialize the search tree with the initial state of \text{problem}

loop
  if there are no candidate states to explore return failure
  choose a leaf node of the tree to expand next according to \text{strategy}
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop
General search algorithm

**General-search** *(problem, strategy)*
- initialize the search tree with the initial state of problem
- loop
  - if there are no candidate states to explore return failure
  - choose a leaf node of the tree to expand next according to strategy
  - if the node satisfies the goal condition return the solution
  - expand the node and add all of its successors to the tree
- end loop
**General search algorithm**

**General-search** *(problem, strategy)*
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore return failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop
General search algorithm

**General-search** (*problem, strategy*)
initialize the search tree with the initial state of *problem*
loop
  if there are no candidate states to explore return failure
  choose a leaf node of the tree to expand next according to *strategy*
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop

• **Search methods** differ in how they explore the space, that is how they choose the node to expand next!!!!!
Implementation of search

- A **search tree node** is a data-structure constituting part of a search tree

![Diagram of search tree node](image)

<table>
<thead>
<tr>
<th>State</th>
<th>ST Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 4 6 1 8 7 3 2</td>
<td>parent</td>
</tr>
</tbody>
</table>

- Expand function – applies Operators to the state represented by the search tree node. Together with Queuing-fn it fills the attributes.

<table>
<thead>
<tr>
<th>Other attributes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>state value (cost)</td>
</tr>
<tr>
<td>depth</td>
</tr>
<tr>
<td>path cost</td>
</tr>
</tbody>
</table>

Uninformed search methods

- rely only on the information available in the problem definition
  - Breadth first search
  - Depth first search
  - Iterative deepening
  - Bi-directional search

**For the minimum cost path problem:**
- Uniform cost search
Search methods

Properties of search methods:

• **Completeness.**
  – Does the method find the solution if it exists?

• **Optimality.**
  – Is the solution returned by the algorithm optimal? Does it give a minimum length path?

• **Space and time complexity.**
  – How much time it takes to find the solution?
  – How much memory is needed to do this?

Parameters to measure complexities.

• **Space and time complexity.**
  – **Complexities** are measured in terms of parameters:
    • $b$ – maximum branching factor
    • $d$ – depth of the optimal solution
    • $m$ – maximum depth of the state space

Branching factor

![Branching factor diagram](image)
Breadth first search (BFS)

- The shallowest node is expanded first

Breadth-first search

- Expand the shallowest node first
- Implementation: put successors to the end of the queue (FIFO)
Breadth-first search

queue ➔ 
Zerind
Sibiu
Timisoara

queue ➔ 
Sibiu
Timisoara
Arad
Oradea
Breadth-first search

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea
Vilcea

queue →
Properties of breadth-first search

- Completeness: Yes. The solution is reached if it exists.
- Optimality: Yes, for the shortest path.
- Time complexity: ?

- Memory (space) complexity: ?
BFS – time complexity

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1=2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2=4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3=8$</td>
</tr>
<tr>
<td>$d$</td>
<td>$2^d$ ($b^d$)</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$2^{d+1}$ ($b^{d+1}$)</td>
</tr>
</tbody>
</table>

Total nodes: $O(b^d)$  

Expanded nodes: $O(b^d)$  

Total nodes: $O(b^{d+1})$
Properties of breadth-first search

- **Completeness**: Yes. The solution is reached if it exists.
- **Optimality**: Yes, for the shortest path.
- **Time complexity**:
  \[1 + b + b^2 + \ldots + b^d = O(b^d)\]
  exponential in the depth of the solution \(d\)
- **Memory (space) complexity**: ?

BFS – memory complexity

- Count nodes kept in the tree structure or in the queue

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(2^1 = 2)</td>
</tr>
<tr>
<td>2</td>
<td>(2^2 = 4)</td>
</tr>
<tr>
<td>3</td>
<td>(2^3 = 8)</td>
</tr>
<tr>
<td>(d)</td>
<td>(2^d) ((b^d))</td>
</tr>
<tr>
<td>(d+1)</td>
<td>(2^{d+1}) ((b^{d+1}))</td>
</tr>
</tbody>
</table>

Total nodes: ?
BFS – memory complexity

- Count nodes kept in the tree structure or in the queue

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1=2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2=4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3=8$</td>
</tr>
<tr>
<td>d</td>
<td>$2^d$ ($b^d$)</td>
</tr>
<tr>
<td>d+1</td>
<td>$2^{d+1}$ ($b^{d+1}$)</td>
</tr>
</tbody>
</table>

Expanded nodes: $O(b^d)$

Total nodes: $O(b^{d+1})$

Properties of breadth-first search

- **Completeness**: Yes. The solution is reached if it exists.
- **Optimality**: Yes, for the shortest path.
- **Time complexity**: 
  
  $1 + b + b^2 + \ldots + b^d = O(b^d)$
  
  exponential in the depth of the solution $d$
- **Memory (space) complexity**: 
  
  $O(b^d)$
  
  nodes are kept in the memory
Depth-first search (DFS)

- The deepest node is expanded first
- Backtrack when the path cannot be further expanded
Depth-first search

Depth-first search
Depth-first search

Properties of depth-first search

- Completeness: Does it always find the solution if it exists?

- Optimality: ?

- Time complexity: ?

- Memory (space) complexity: ?

Note: Arad – Zerind – Arad cycle
Properties of depth-first search

- **Completeness**: No. Infinite loops can occur. Infinite loops imply -> Infinite depth search tree.
- **Optimality**: does it find the minimum length path?
- **Time complexity**: ?
- **Memory (space) complexity**: ?

Properties of depth-first search

- **Completeness**: No. Infinite loops can occur.
- **Optimality**: No. Solution found first may not be the shortest possible.
- **Time complexity**: ?
- **Memory (space) complexity**: ?
Properties of depth-first search

- **Completeness**: No. Infinite loops can occur.

- **Optimality**: No. Solution found first may not be the shortest possible.

- **Time complexity**: ?

- **Memory (space) complexity**: ?

DFS – time complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1 = 2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2 = 4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3 = 8$</td>
</tr>
<tr>
<td>d</td>
<td>$2^d$</td>
</tr>
<tr>
<td>m</td>
<td>$2^m$, $2^{m-d}$</td>
</tr>
</tbody>
</table>

Complexity:
DFS – time complexity

Properties of depth-first search

- **Completeness:** No. Infinite loops can occur.
- **Optimality:** No. Solution found first may not be the shortest possible.
- **Time complexity:**
  \[ O(b^m) \]
  exponential in the maximum depth of the search tree \( m \)
- **Memory (space) complexity:** ?
Properties of depth-first search

- **Completeness**: No. Infinite loops can occur.

- **Optimality**: No. Solution found first may not be the shortest possible.

- **Time complexity**: 
  
  \[ O(b^n) \]

  exponential in the maximum depth of the search tree \( m \)

- **Memory (space) complexity**: ?

---

**DFS – memory complexity**

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2 = b</td>
</tr>
</tbody>
</table>

![DFS memory complexity](image)

### DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 = (b-1)</td>
</tr>
<tr>
<td>2</td>
<td>2 = b</td>
</tr>
</tbody>
</table>

![DFS memory complexity](image)
DFS – memory complexity

<table>
<thead>
<tr>
<th>Depth</th>
<th>Number of Nodes Kept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>2=b</td>
</tr>
</tbody>
</table>

Complexity: $O(b^m)$
Properties of depth-first search

- **Completeness:** No. Infinite loops can occur.

- **Optimality:** No. Solution found first may not be the shortest possible.

- **Time complexity:**
  \[ O(b^m) \]
  exponential in the maximum depth of the search tree \( m \)

- **Memory (space) complexity:**
  \[ O(bm) \]
  linear in the maximum depth of the search tree \( m \)

---

**DFS – memory complexity**

Count nodes kept in the tree structure or the queue

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2 = b</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>( m )</td>
<td>2</td>
</tr>
</tbody>
</table>

Total nodes: \( O(bm) \)
Properties of depth-first search

• **Completeness:** No. Infinite loops can occur.

• **Optimality:** No. Solution found first may not be the shortest possible.

• **Time complexity:**
  \[ O(b^m) \]
  exponential in the maximum depth of the search tree \( m \)

• **Memory (space) complexity:**
  \[ O(bm) \]
  the tree size we need to keep is linear in the maximum depth of the search tree \( m \)