Uninformed search methods I.

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

A branch in the search tree
= path in the graph
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 \((\text{problem}, \text{strategy})\)
initialize the search tree with the initial state of \text{problem}
loop
\[\text{if there are no candidate states to explore return failure}\]
\[\text{choose a leaf node of the tree to expand next according to strategy}\]
\[\text{if the node satisfies the goal condition return the solution}\]
\[\text{expand the node and add all of its successors to the tree}\]
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

- Search methods can be implemented using *queue* structure

**General search** *(problem, Queuing-fn)*
- nodes ← Make-queue(Make-node(Initial-state(problem)))
- loop
  - if nodes is empty then return failure
  - node ← Remove-node(nodes)
  - if Goal-test(*problem*) applied to State(*node*) is satisfied then return node
  - nodes ← Queuing-fn(nodes, Expand(node, Operators(node)))
- end loop

- Candidates are added to *nodes* representing the queue structure
Implementation of search

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

  ![ST Node Diagram]

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

  **Other attributes:**
  - State value (cost)
  - Depth
  - Path cost

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**

[Diagram of branching factor]
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

```
Arad
  |   |
  |   v
  Zerind  Sibiu  Timisoara
```

```
queue

```
Sibiu  Timisoara  Arad  Oradea
```

```
queue

```
Sibiu  Timisoara  Arad  Oradea
```

CS 1571 Intro to AI
M. Hauskrecht

11
Breadth-first search

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

queue

Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

Arad
Lugoj

Arad
Lugoj

CS 1571 Intro to AI
M. Hauskrecht
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

**Depth** | **Number of Nodes**
--- | ---
0 | 1
1 | $2^1=2$
2 | $2^2=4$
3 | $2^3=8$

- $b^d$ (BD)
- $b^{d+1}$ (BD+1)

**Total nodes:** 

**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>$2^d$</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$2^{d+1}$</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

- The deepest node is expanded first
- Implementation: put successors to the beginning of the queue

![Diagram of Depth-first search](image)
Depth-first search

queue: Zerind, Sibiu, Timisoara

Arad
- Zerind
- Sibiu
- Timisoara

Oradea
- Fagaras
- Rimnicu Vilcea

Rimnicu Vilcea
- Arad
- Lugoj

Arad
- Oradea

Sibiu
- Oradea

Timisoara
- Arad

Oradea
- Arad

Fagaras
- Arad

Vilcea
- Arad

Lugoj
- Arad
Depth-first search

Note: Arad – Zerind – Arad cycle

Properties of depth-first search

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

- Optimality: ?

- Time complexity: ?

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

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>$2^{d=2}$</td>
</tr>
<tr>
<td>2</td>
<td>$2^{d=4}$</td>
</tr>
<tr>
<td>3</td>
<td>$2^{d=8}$</td>
</tr>
</tbody>
</table>

Complexity: $2^m$, $2^{m-d}$
**DFS – time complexity**

- **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:**?
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:** ?

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

<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

<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>d</td>
<td>1 = (b-1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>2 = b</td>
</tr>
</tbody>
</table>

Complexity: $O(bm)$
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:**
  \[ O(bm) \]
  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>...</td>
<td>...</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 \( m \)