# Data Structures and Algorithms

## 8. Search Algorithms

Basic Stuff You're Gonna Need to Search over a Graph
Where To Search Next?

Nir Lipovetzky

THE UNIVERSITY OF MELBOURNE

# Path-finding in graphs

**Search algorithms** over **directed graphs**:

- The **search nodes** (vertex) of the graph represent some information
- The edges $(v, v')$ capture transitions

One way to understand search algorithms is to think about **path-finding** over **graphs**.

# Classification of Search Algorithms

**Blind search vs. heuristic (or informed) search:**

- Blind search algorithms: Only use the basic ingredients for general search algorithms.
  - *e.g., Depth First Search (DFS), Breadth-first search (BrFS), Uniform Cost (Dijkstra), Iterative Deepening (ID)*
- Heuristic search algorithms: Additionally use heuristic functions which estimate the distance (or remaining cost) to reach an target vertex.
  - *e.g., A*, IDA*, Hill Climbing, Best First, WA*, DFS B&B, LRTA*, . . .*

**Systematic search vs. local search:**

- Systematic search algorithms: Consider a large number of search nodes simultaneously.
- Local search algorithms: Work with one (or a few) candidate solutions (search nodes) at a time.

$\rightarrow$ This is not a black-and-white distinction; there are *crossbreeds* (e.g., enforced hill-climbing).

# Blind Search

**Blind search:**

→ Here, we cover the subset of search algorithms. Only some Blind search algorithms are covered. Some Algorithms do not appear in Skiena's book.

# Agenda

1. Basics
2. Blind Systematic Search Algorithms

## Search Terminology

**Search node $n$:** Contains a *vertex* reached by the search, plus information about how it was reached.

**Path cost $g(n)$:** The cost of the path reaching $n$.

**Optimal cost $g^*$:** The cost of an optimal solution path. For a state $s$, $g^*(s)$ is the cost of a cheapest path reaching $s$.

# Search Terminology

**Search node $n$:** Contains a *vertex* reached by the search, plus information about how it was reached.

**Path cost $g(n)$:** The cost of the path reaching $n$.

**Optimal cost $g^*$:** The cost of an optimal solution path. For a state $s$, $g^*(s)$ is the cost of a cheapest path reaching $s$.

**Node expansion:** Generating all successors of a node, considering all adjacent transitions to the node's vertex. Afterwards, the *node/vertex* itself is also said to be expanded.

**Search strategy:** Method for deciding which node is expanded next.

## Search Terminology

**Search node $n$:** Contains a *vertex* reached by the search, plus information about how it was reached.

**Path cost $g(n)$:** The cost of the path reaching $n$.

**Optimal cost $g^*$:** The cost of an optimal solution path. For a state $s$, $g^*(s)$ is the cost of a cheapest path reaching $s$.

**Node expansion:** Generating all successors of a node, considering all adjacent transitions to the node's vertex. Afterwards, the *node/vertex* itself is also said to be expanded.

**Search strategy:** Method for deciding which node is expanded next.

**Open list:** Set of all *nodes* that currently are candidates for expansion. Also called frontier.

**Closed list:** Set of all *nodes/vertexes* that were already expanded. Used only in graph search, not in tree search (up next). Also called explored set.

## Search Terminology

**Search node $n$:** Contains a *vertex* reached by the search, plus information about how it was reached.

**Path cost $g(n)$:** The cost of the path reaching $n$.

**Optimal cost $g^*$:** The cost of an optimal solution path. For a state $s$, $g^*(s)$ is the cost of a cheapest path reaching $s$.

**Node expansion:** Generating all successors of a node, considering all adjacent transitions to the node's vertex. Afterwards, the *node/vertex* itself is also said to be expanded.

**Search strategy:** Method for deciding which node is expanded next.

**Open list:** Set of all *nodes* that currently are candidates for expansion. Also called frontier.

**Closed list:** Set of all *nodes/vertexes* that were already expanded. Used only in graph search, not in tree search (up next). Also called explored set.

# Search Nodes

- **Search nodes $\sigma$**: Search vertexes, plus information on where/when/how they are encountered during search.

**What is in a search node?**

Different search algorithms store different information in a search node $\sigma$, but typical information includes:

- *Vertex*$(\sigma)$: Associated search vertex.
- *parent*$(\sigma)$: Pointer to search node from which $\sigma$ is reached.
- $g(\sigma)$: Cost of $\sigma$ (cost of path from the root node to $\sigma$).

For the root node, *parent*$(\sigma)$ and *action*$(\sigma)$ are undefined.

# Search Nodes

- **Search nodes** $\sigma$: Search vertexes, plus information on where/when/how they are encountered during search.

**What is in a search node?**

Different search algorithms store different information in a search node $\sigma$, but typical information includes:

- *Vertex*$(\sigma)$: Associated search vertex.
- *parent*$(\sigma)$: Pointer to search node from which $\sigma$ is reached.
- $g(\sigma)$: Cost of $\sigma$ (cost of path from the root node to $\sigma$).

For the root node, *parent*$(\sigma)$ and *action*$(\sigma)$ are undefined.

# Criteria for Evaluating Search Strategies

**Guarantees:**

Completeness: Is the strategy guaranteed to find a solution when there is one?

Optimality: Are the returned solutions guaranteed to be optimal?

**Complexity:**

Time Complexity: How long does it take to find a solution? (Measured in generated nodes.)

Space Complexity: How much memory does the search require? (Measured in nodes.)

# Criteria for Evaluating Search Strategies

**Guarantees:**

Completeness: Is the strategy guaranteed to find a solution when there is one?

Optimality: Are the returned solutions guaranteed to be optimal?

**Complexity:**

Time Complexity: How long does it take to find a solution? (Measured in generated nodes.)

Space Complexity: How much memory does the search require? (Measured in nodes.)

**Typical search space features governing complexity:**

Branching factor $b$: How many successors does each node have?

Goal depth $d$: The number of actions required to reach the shallowest goal vertex.

# Criteria for Evaluating Search Strategies

**Guarantees:**

Completeness: Is the strategy guaranteed to find a solution when there is one?

Optimality: Are the returned solutions guaranteed to be optimal?

**Complexity:**

Time Complexity: How long does it take to find a solution? (Measured in generated nodes.)

Space Complexity: How much memory does the search require? (Measured in nodes.)

**Typical search space features governing complexity:**

Branching factor $b$: How many successors does each node have?

Goal depth $d$: The number of actions required to reach the shallowest goal vertex.

# Agenda

1 Basics

2 Blind Systematic Search Algorithms

# Before We Begin

**Blind search vs. informed search:**

- Blind search does not require any input beyond the graph.

  → Pros and Cons? Pro: No additional work for the programmer. Con: It's not called "blind" for nothing ... same expansion order regardless what the problem actually is.

# Before We Begin

**Blind search vs. informed search:**

- Blind search does not require any input beyond the graph.

  $\rightarrow$ Pros and Cons? Pro: No additional work for the programmer. Con: It's not called "blind" for nothing . . . same expansion order regardless what the problem actually is.

- Informed search requires as additional input a heuristic function $h$ that maps nodes to estimates of their distance.

  $\rightarrow$ Pros and Cons?

# Before We Begin

**Blind search vs. informed search:**

- Blind search does not require any input beyond the graph.

  → Pros and Cons? Pro: No additional work for the programmer. Con: It's not called "blind" for nothing . . . same expansion order regardless what the problem actually is.

- Informed search requires as additional input a heuristic function $h$ that maps nodes to estimates of their distance.

  → Pros and Cons? Pro: Typically more effective in practice. Con: Somebody's gotta come up with/implement $h$.

# Before We Begin

**Blind search vs. informed search:**

- Blind search does not require any input beyond the graph.

  $\rightarrow$ Pros and Cons? Pro: No additional work for the programmer. Con: It's not called "blind" for nothing ... same expansion order regardless what the problem actually is.

- Informed search requires as additional input a heuristic function $h$ that maps nodes to estimates of their distance.

  $\rightarrow$ Pros and Cons? Pro: Typically more effective in practice. Con: Somebody's gotta come up with/implement $h$.

# Before We Begin, ctd.

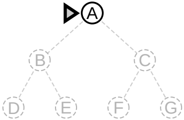
**Blind search strategies we'll discuss:**

- Breadth-first search. Advantage: time complexity.
  Variant: Uniform cost search.
- Depth-first search. Advantage: space complexity.
- Iterative deepening search. Combines advantages of breadth-first search and depth-first search. Uses depth-limited search as a sub-procedure.

# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

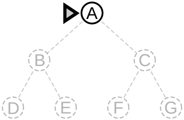
**Illustration:**

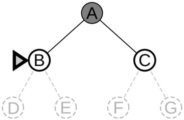


# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

**Illustration:**

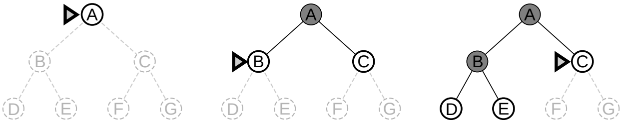


# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

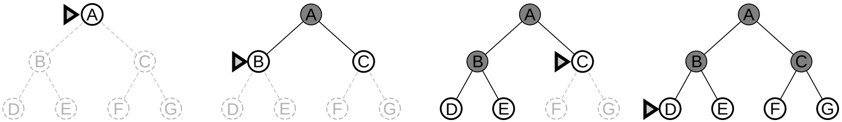
**Illustration:**



# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

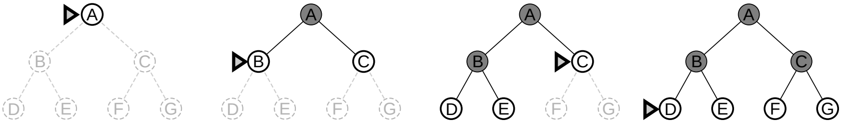
**Illustration:**



# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

**Illustration:**



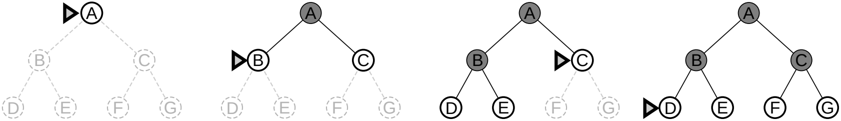

**Guarantees:**

- Completeness?

# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

**Illustration:**



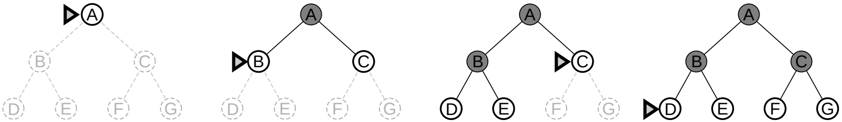

**Guarantees:**

- Completeness? Yes.
- Optimality?

# Breadth-First Search: Illustration and Guarantees

**Strategy:** Expand nodes in the order they were produced (FIFO frontier).

**Illustration:**




**Guarantees:**

- Completeness? Yes.
- Optimality? Yes, for uniform action costs. Breadth-first search always finds a shallowest goal vertex. If costs are not uniform, this is not necessarily optimal.

# Breadth-First Search: Complexity

**Time Complexity:** Say that $b$ is the maximal branching factor, and $d$ is the depth of goal vertex.

- Upper bound on the number of generated nodes? $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first $d$ layers.
- So the time complexity is $O(b^d)$.
- And if we were to check the goal test at node-expansion time, rather than node-generation time?

# Breadth-First Search: Complexity

**Time Complexity:** Say that $b$ is the maximal branching factor, and $d$ is the depth of goal vertex.

- Upper bound on the number of generated nodes? $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first $d$ layers.
- So the time complexity is $O(b^d)$.
- And if we were to check the goal test at node-expansion time, rather than node-generation time? $O(b^{d+1})$ because then we'd generate the first $d + 1$ layers in the worst case.

**Space Complexity:**

# Breadth-First Search: Complexity

**Time Complexity:** Say that $b$ is the maximal branching factor, and $d$ is the depth of goal vertex.

- Upper bound on the number of generated nodes? $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first $d$ layers.
- So the time complexity is $O(b^d)$.
- And if we were to check the goal test at node-expansion time, rather than node-generation time? $O(b^{d+1})$ because then we'd generate the first $d+1$ layers in the worst case.

**Space Complexity:** Same as time complexity since all generated nodes are kept in memory.

# Breadth-First Search: Complexity

**Time Complexity:** Say that $b$ is the maximal branching factor, and $d$ is the depth of goal vertex.

- Upper bound on the number of generated nodes? $b + b^2 + b^3 + \cdots + b^d$: In the worst case, the algorithm generates all nodes in the first $d$ layers.
- So the time complexity is $O(b^d)$.
- And if we were to check the goal test at node-expansion time, rather than node-generation time? $O(b^{d+1})$ because then we'd generate the first $d+1$ layers in the worst case.

**Space Complexity:** Same as time complexity since all generated nodes are kept in memory.

# Breadth-First Search: Example Data

**Setting:** $b = 10$; $10000$ nodes/second; $1000$ bytes/node.

**Yields data:** (inserting values into previous equations)

| Depth | Nodes | Time | | Memory | |
|---|---|---|---|---|---|
| 2 | 110 | .11 | milliseconds | 107 | kilobytes |
| 4 | 11110 | 11 | milliseconds | 10.6 | megabytes |
| 6 | $10^6$ | 1.1 | seconds | 1 | gigabyte |
| 8 | $10^8$ | 2 | minutes | 103 | gigabytes |
| 10 | $10^{10}$ | 3 | hours | 10 | terabytes |
| 12 | $10^{12}$ | 13 | days | 1 | petabyte |
| 14 | $10^{14}$ | 3.5 | years | 99 | petabytes |

$\rightarrow$ So, which is the worse problem, time or memory? Memory. (In my own experience, typically exhausts RAM memory within a few minutes.)

# Breadth-First Search: Example Data

**Setting:** $b = 10$; $10000$ nodes/second; $1000$ bytes/node.

**Yields data:** (inserting values into previous equations)

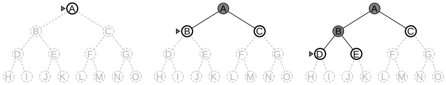
| Depth | Nodes | Time | | Memory | |
|---|---|---|---|---|---|
| 2 | 110 | .11 | milliseconds | 107 | kilobytes |
| 4 | 11110 | 11 | milliseconds | 10.6 | megabytes |
| 6 | $10^6$ | 1.1 | seconds | 1 | gigabyte |
| 8 | $10^8$ | 2 | minutes | 103 | gigabytes |
| 10 | $10^{10}$ | 3 | hours | 10 | terabytes |
| 12 | $10^{12}$ | 13 | days | 1 | petabyte |
| 14 | $10^{14}$ | 3.5 | years | 99 | petabytes |

$\rightarrow$ So, which is the worse problem, time or memory? Memory. (In my own experience, typically exhausts RAM memory within a few minutes.)

# Depth-First Search: Illustration

**Strategy:** Expand the most recent nodes in (LIFO frontier).

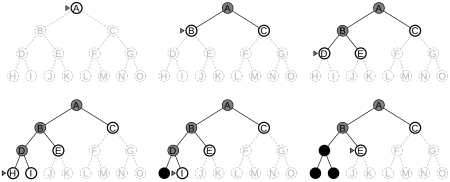
**Illustration:** (Nodes at depth 3 are assumed to have no successors)



# Depth-First Search: Illustration

**Strategy:** Expand the most recent nodes in (LIFO frontier).

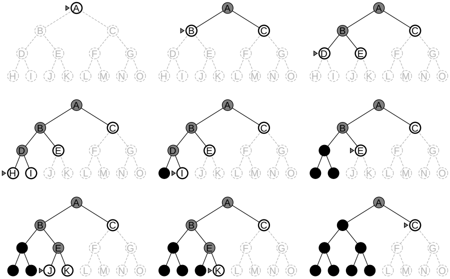
**Illustration:** (Nodes at depth 3 are assumed to have no successors)



# Depth-First Search: Illustration

**Strategy:** Expand the most recent nodes in (LIFO frontier).

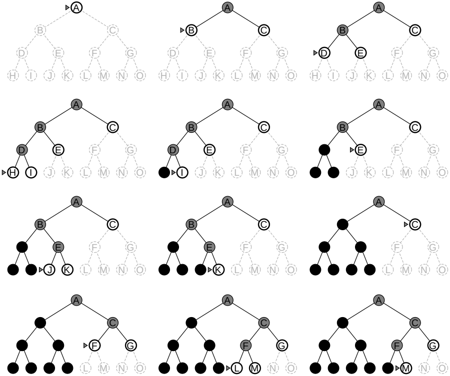
**Illustration:** (Nodes at depth 3 are assumed to have no successors)



# Depth-First Search: Illustration

**Strategy:** Expand the most recent nodes in (LIFO frontier).

**Illustration:** (Nodes at depth 3 are assumed to have no successors)



# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just “chooses some direction and hopes for the best”. (Depth-first search is a way of “hoping to get lucky”.)
- Completeness?

# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just “chooses some direction and hopes for the best”. (Depth-first search is a way of “hoping to get lucky”.)
- Completeness? No, because search branches may be infinitely long: No check for cycles along a branch!

  $\rightarrow$ Depth-first search is complete in case the search space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite search spaces.

# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just “chooses some direction and hopes for the best”. (Depth-first search is a way of “hoping to get lucky”.)
- Completeness? No, because search branches may be infinitely long: No check for cycles along a branch!

  $\rightarrow$ Depth-first search is complete in case the search space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite search spaces.

**Complexity:**

- Space: Stores nodes and applicable actions on the path to the current node. So if $m$ is the maximal depth reached, the complexity is

# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just "chooses some direction and hopes for the best". (Depth-first search is a way of "hoping to get lucky".)
- Completeness? No, because search branches may be infinitely long: No check for cycles along a branch!

  $\rightarrow$ Depth-first search is complete in case the search space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite search spaces.

**Complexity:**

- Space: Stores nodes and applicable actions on the path to the current node. So if $m$ is the maximal depth reached, the complexity is $O(b\,m)$.
- Time: If there are paths of length $m$ in the search space, $O(b^m)$ nodes can be generated. Even if there are solutions of depth $1$!

# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just “chooses some direction and hopes for the best”. (Depth-first search is a way of “hoping to get lucky”.)
- Completeness? No, because search branches may be infinitely long: No check for cycles along a branch!

  $\rightarrow$ Depth-first search is complete in case the search space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite search spaces.

**Complexity:**

- Space: Stores nodes and applicable actions on the path to the current node. So if $m$ is the maximal depth reached, the complexity is $O(b\,m)$.
- Time: If there are paths of length $m$ in the search space, $O(b^m)$ nodes can be generated. Even if there are solutions of depth $1$!

  $\rightarrow$ If we happen to choose “the right direction” then we can find a length-$l$ solution in time $O(b\,l)$ regardless how big the search space is.

# Depth-First Search: Guarantees and Complexity

**Guarantees:**

- Optimality? No. After all, the algorithm just "chooses some direction and hopes for the best". (Depth-first search is a way of "hoping to get lucky".)
- Completeness? No, because search branches may be infinitely long: No check for cycles along a branch!

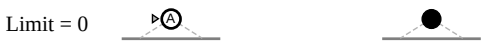
  $\rightarrow$ Depth-first search is complete in case the search space is acyclic, e.g., Constraint Satisfaction Problems. If we do add a cycle check, it becomes complete for finite search spaces.

**Complexity:**

- Space: Stores nodes and applicable actions on the path to the current node. So if $m$ is the maximal depth reached, the complexity is $O(b\,m)$.
- Time: If there are paths of length $m$ in the search space, $O(b^m)$ nodes can be generated. Even if there are solutions of depth $1$!

  $\rightarrow$ If we happen to choose "the right direction" then we can find a length-$l$ solution in time $O(b\,l)$ regardless how big the search space is.

# Iterative Deepening Search: Illustration

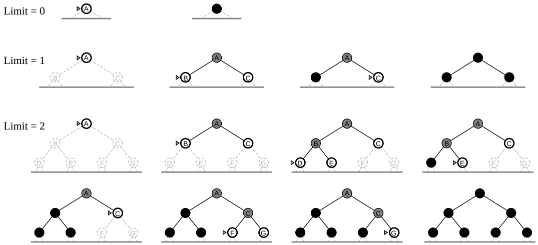


# Iterative Deepening Search: Illustration

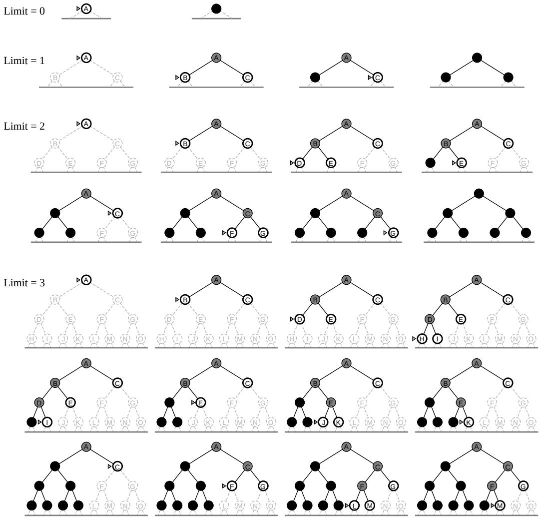
Limit = 0


Limit = 1

# Iterative Deepening Search: Illustration



# Iterative Deepening Search: Illustration



# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality?

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness?

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness? Yes! Space complexity?

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness? Yes! Space complexity? $O(b\,d)$.

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=*
*Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness? Yes! Space complexity? $O(b\,d)$.

**Time complexity:**

| | |
|---|---|
| Breadth-First-Search | $b + b^2 + \cdots + b^{d-1} + b^d \in O(b^d)$ |
| Iterative Deepening Search | $(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d \in O(b^d)$ |

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness? Yes! Space complexity? $O(b\,d)$.

**Time complexity:**

| | |
|---|---|
| Breadth-First-Search | $b + b^2 + \cdots + b^{d-1} + b^d \in O(b^d)$ |
| Iterative Deepening Search | $(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d \in O(b^d)$ |

**Example:** $b = 10, d = 5$

| | |
|---|---|
| Breadth-First Search | $10 + 100 + 1,000 + 10,000 + 100,000 = 111,110$ |
| Iterative Deepening Search | $50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$ |

$\rightarrow$ IDS combines the advantages of breadth-first and depth-first search. It is the preferred blind search method in large search spaces with unknown solution depth.

# Iterative Deepening Search: Guarantees and Complexity

*"Iterative Deepening Search=
Keep doing the same work over again until you find a solution."*

**BUT:** Optimality? Yes! Completeness? Yes! Space complexity? $O(b\,d)$.

**Time complexity:**

| | |
|---|---|
| Breadth-First-Search | $b + b^2 + \cdots + b^{d-1} + b^d \in O(b^d)$ |
| Iterative Deepening Search | $(d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d \in O(b^d)$ |

**Example:** $b = 10, d = 5$

| | |
|---|---|
| Breadth-First Search | $10 + 100 + 1,000 + 10,000 + 100,000 = 111,110$ |
| Iterative Deepening Search | $50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$ |

$\rightarrow$ IDS combines the advantages of breadth-first and depth-first search. It is the preferred blind search method in large search spaces with unknown solution depth.