CS 1571 Introduction to AI
Lecture 5

Uninformed search methods II.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Announcements

Homework assignment 1 is out

- in the electronic form on the course web page
- Due on Wednesday, September 20, 2006 before the class
- **Two parts:**
  - Pen and pencil part
  - Programming part (Puzzle 8)

Course web page:

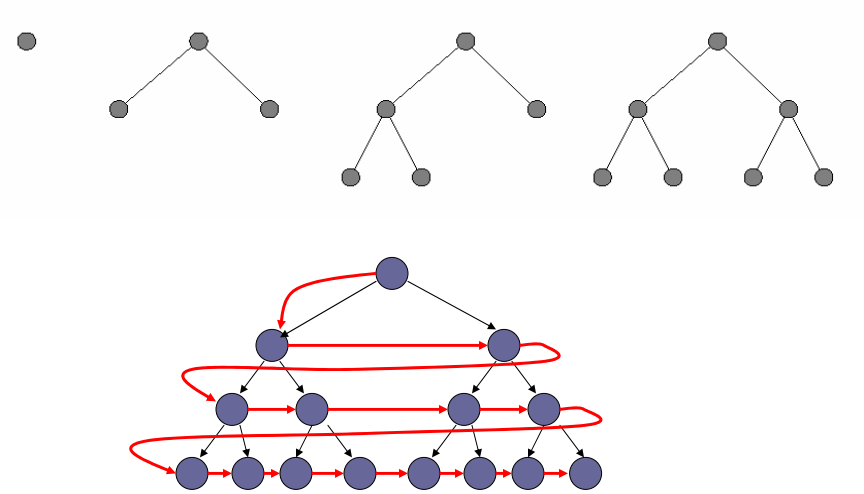
http://www.cs.pitt.edu/~milos/courses/cs1571/

Uninformed methods

- Uninformed search methods use only information available in the problem definition
  - **Breadth-first search (BFS)**
  - **Depth-first search (DFS)**
  - **Iterative deepening (IDA)**
  - **Bi-directional search**
- **For the minimum cost path problem:**
  - **Uniform cost search**

Breadth first search (BFS)

- The shallowest node is expanded first



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*

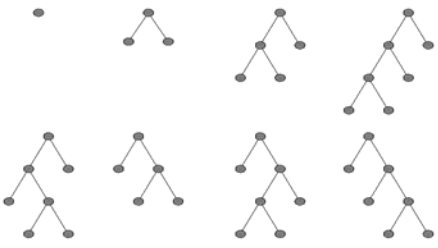
- **Memory (space) complexity:**

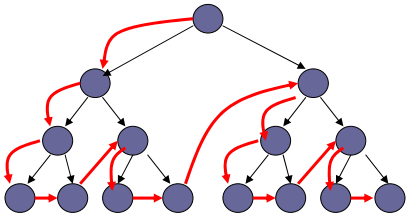
$$O(b^d)$$

same as time - every node is kept in the memory

Depth-first search (DFS)

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





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:**

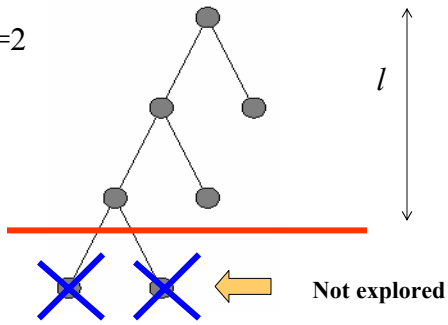
  $O(bm)$

  linear in the maximum depth of the search tree *m*

Limited-depth depth first search

- How to eliminate infinite depth first exploration?
- Put the limit *(l)* on the depth of the depth-first exploration

Limit l=2



- **Time complexity:** $O(b^l)$
- **Memory complexity**: $O(bl)$

l - is the given limit

Limited depth depth-first search

- Avoids pitfalls of depth first search
- Use cutoff on the maximum depth of the tree
- **Problem:** **How to pick the maximum depth?**

- **Assume:** we have a traveler problem with 20 cities
- How to pick the maximum tree depth?

Limited depth depth-first search

- Avoids pitfalls of depth first search
- Use cutoff on the maximum depth of the tree
- **Problem:** **How to pick the maximum depth?**

- **Assume:** we have a traveler problem with 20 cities
  - How to pick the maximum tree depth?
  - **We need to consider only paths of length < 20**

- Limited depth DFS
- **Time complexity:** $O(b^l)$ l - is the limit
- **Memory complexity**: $O(bl)$

Elimination of state repeats

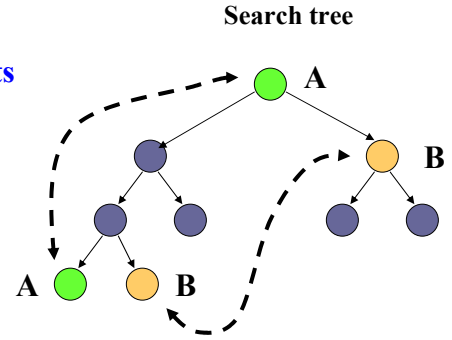
While searching the state space for the solution we can encounter the same state many times.

Question: Is it necessary to keep and expand all copies of states in the search tree?

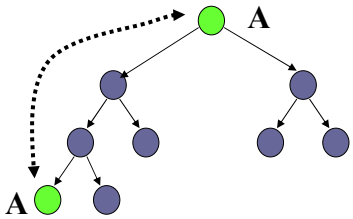
Two possible cases:

(A) Cyclic state repeats

(B) Non-cyclic state repeats



Elimination of cycles

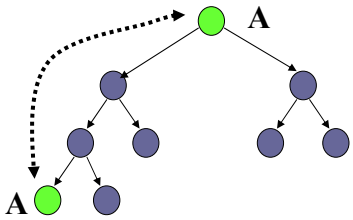


Case A: Corresponds to the path with a cycle

Can the branch (path) in which the same state is visited twice ever be a part of the optimal (shortest) path between the initial state and the goal?

???

Elimination of cycles

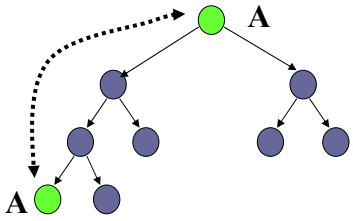


Case A: Corresponds to the path with a cycle

Can the branch (path) in which the same state is visited twice ever be a part of the optimal (shortest) path between the initial state and the goal? **No !!**

Branches representing cycles cannot be the part of the shortest solution and can be eliminated.

Elimination of cycles

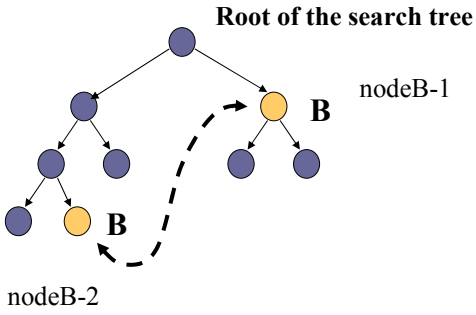


How to check for cyclic state repeats:

- Check ancestors in the tree structure
- Caveat: we need to keep the tree.

Do not expand the node with the state that is the same as the state in one of its ancestors.

Elimination of non-cyclic state repeats

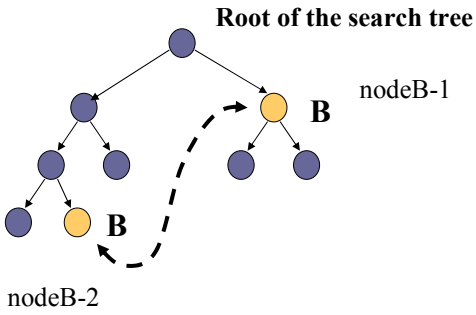


Case B: nodes with the same state are not on the same path from the initial state

Is one of the nodes nodeB-1, nodeB-2 better or preferable?

?

Elimination of non-cyclic state repeats

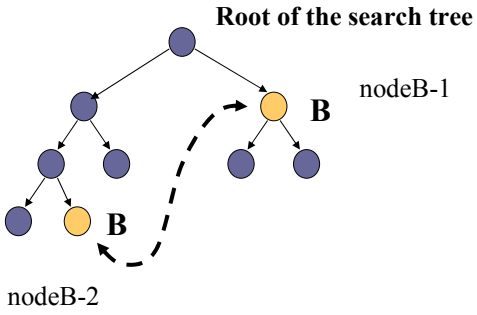


Case B: nodes with the same state are not on the same path from the initial state

Is one of the nodes nodeB-1, nodeB-2 better or preferable?

Yes. nodeB-1 represents the shorter path between the initial state and B

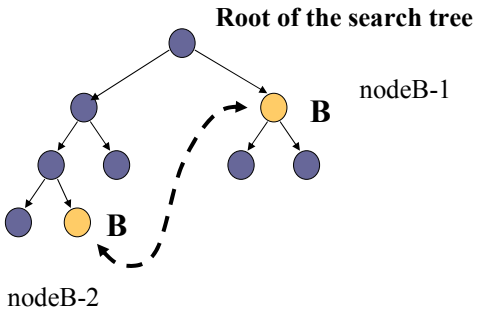
Elimination of non-cyclic state repeats



Since we are happy with the optimal solution nodeB-2 can be eliminated. It does not affect the optimality of the solution.

Problem: Nodes can be encountered in different order during different search strategies.

Elimination of non-cyclic state repeats with BFS



Breadth FS is well behaved with regard to non-cyclic state repeats: nodeB-1 is always expanded before nodeB-2

- Order of expansion determines the correct elimination strategy
- we can safely eliminate the node that is associated with the state that has been expanded before

Elimination of state repeats for the BFS

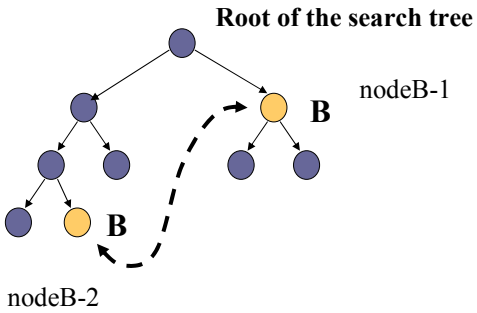
For **the breadth-first search (BFS)**

- we can safely eliminate all second, third, fourth, etc. occurrences of the same state
- this rule covers both cyclic and non-cyclic repeats !!!

Implementation of all state repeat elimination through **marking**:

- All expanded states are marked
- All marked states are stored in a hash table
- Checking if the node has ever been expanded corresponds to the mark structure lookup

Elimination of non-cyclic state repeats with DFS



Depth FS: nodeB-2 is expanded before nodeB-1

- The order of node expansion does not imply correct elimination strategy
- we need to remember the length of the path between nodes to safely eliminate them

Elimination of all state redundancies

- **General strategy:** A node is redundant if there is another node with exactly the same state and a shorter path from the initial state
  - Works for any search method
  - Uses additional path length information

Implementation: marking with the minimum path value:

- The new node is redundant and can be eliminated if
  - it is in the hash table (it is marked), and
  - its path is longer or equal to the value stored.
- Otherwise the new node cannot be eliminated and it is entered together with its value into the hash table. (if the state was in the hash table the new path value is better and needs to be overwritten.)

Iterative deepening algorithm (IDA)

- Based on the idea of the limited-depth search, but
- It resolves the difficulty of knowing the depth limit ahead of time.

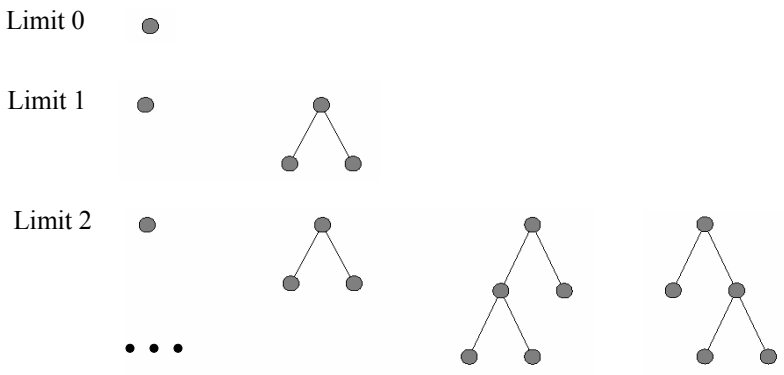
Idea: try all depth limits in an increasing order.

That is, search first with the depth limit $l=0$, then $l=1$, $l=2$, and so on until the solution is reached

Iterative deepening combines advantages of the depth-first and breadth-first search with only moderate computational overhead

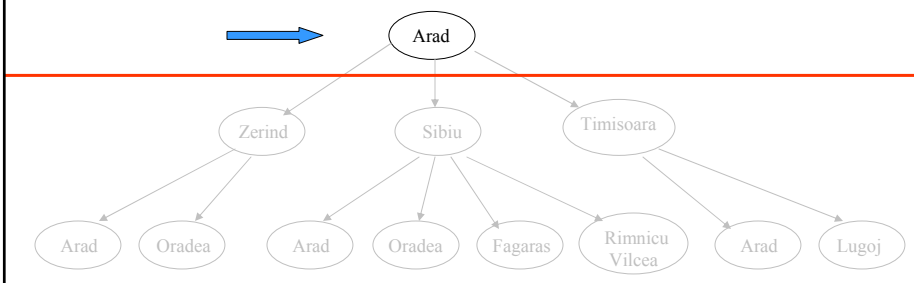
Iterative deepening algorithm (IDA)

- Progressively increases the limit of the limited-depth depth-first search



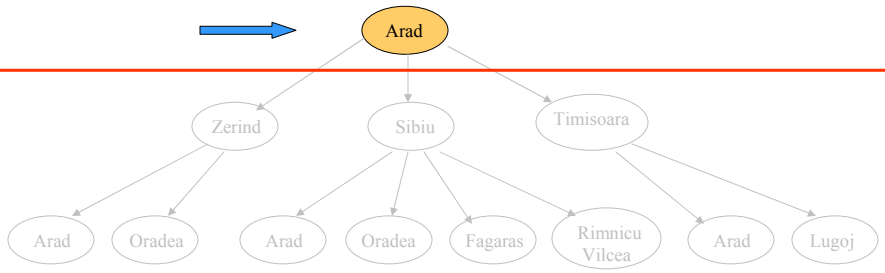
Iterative deepening

Cutoff depth = 0



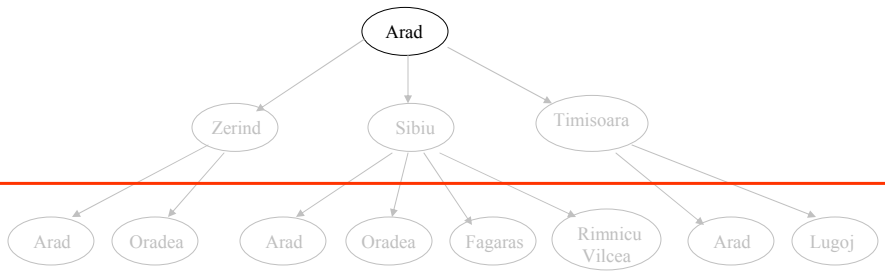
Iterative deepening

Cutoff depth = 0



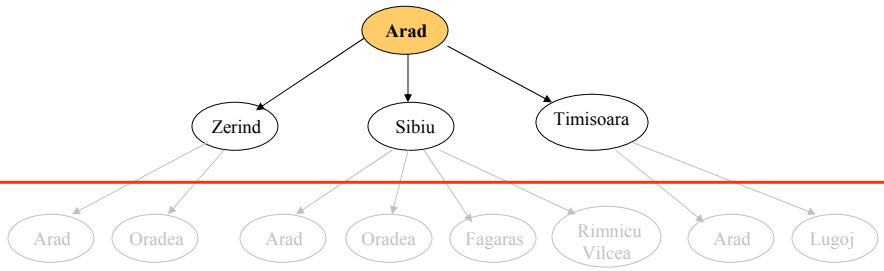
Iterative deepening

Cutoff depth = 1



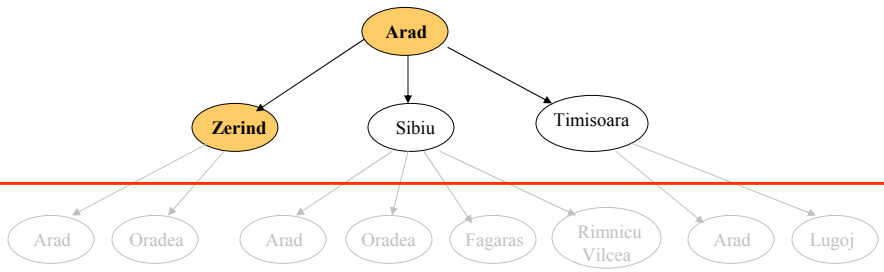
Iterative deepening

Cutoff depth = 1



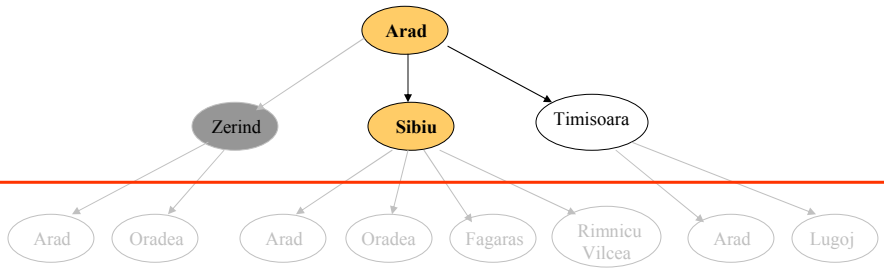
Iterative deepening

Cutoff depth = 1



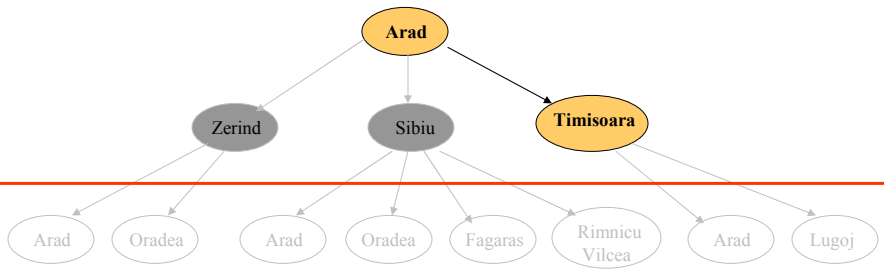
Iterative deepening

Cutoff depth = 1



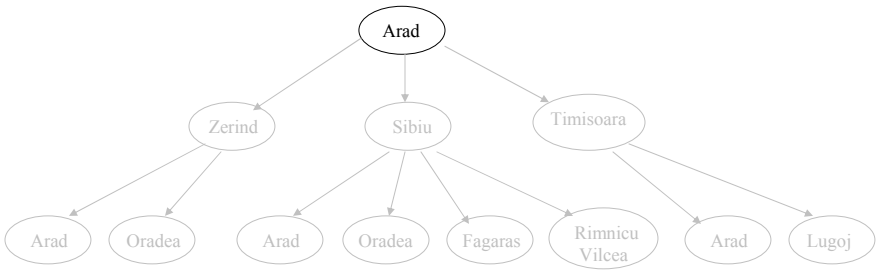
Iterative deepening

Cutoff depth = 1



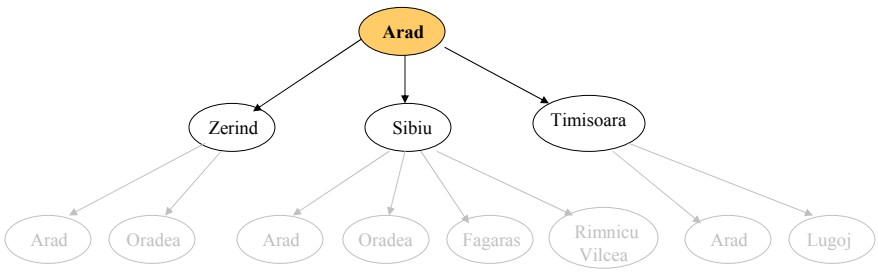
Iterative deepening

Cutoff depth = 2



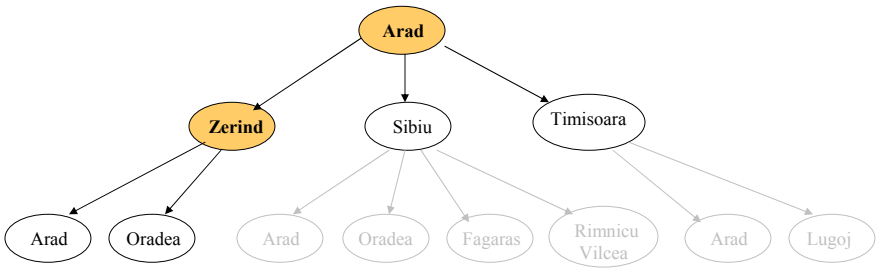
Iterative deepening

Cutoff depth = 2



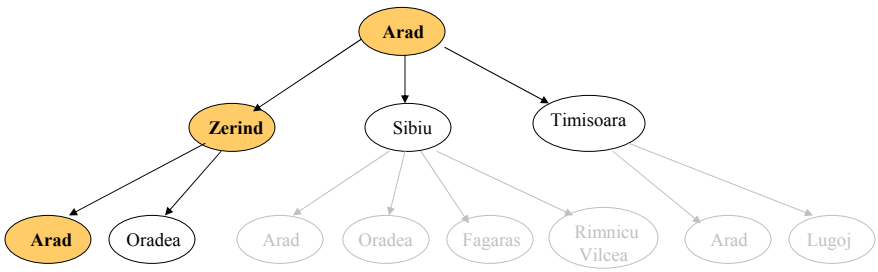
Iterative deepening

Cutoff depth = 2



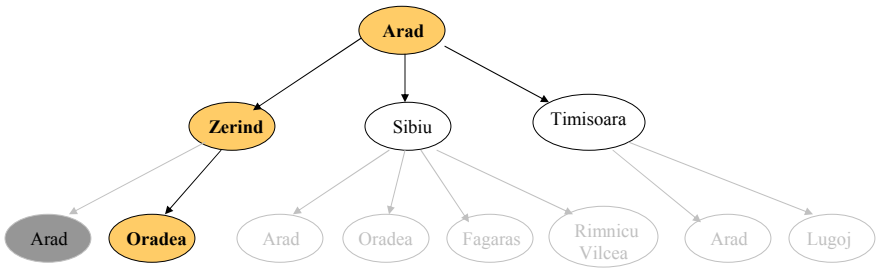
Iterative deepening

Cutoff depth = 2



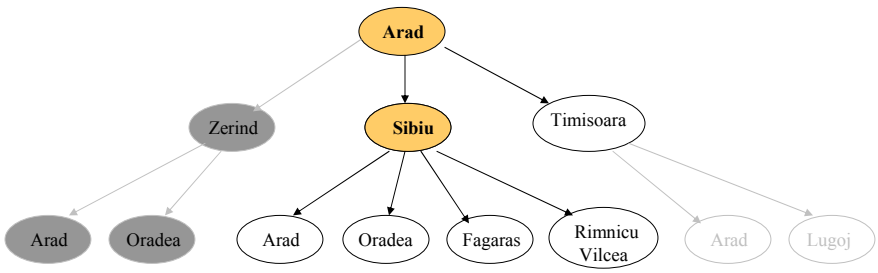
Iterative deepening

Cutoff depth = 2



Iterative deepening

Cutoff depth = 2



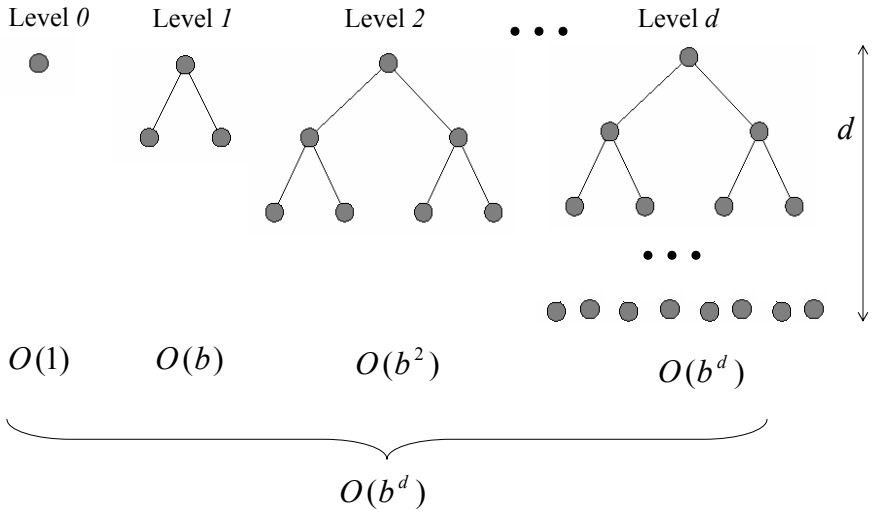
Properties of IDA

- **Completeness: ?**
- **Optimality: ?**
- **Time complexity:**
  ?
- **Memory (space) complexity:**
  ?

Properties of IDA

- **Completeness:** **Yes.** The solution is reached if it exists.
  (the same as BFS when limit is always increased by 1)
- **Optimality:** **Yes**, for the shortest path.
  (the same as BFS)
- **Time complexity:**
  ?
- **Memory (space) complexity:**
  ?

IDA – time complexity



Properties of IDA

- **Completeness:** **Yes.** The solution is reached if it exists.
  (the same as BFS)
- **Optimality:** **Yes**, for the shortest path.
  (the same as BFS)
- **Time complexity:**

$$O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d)$$

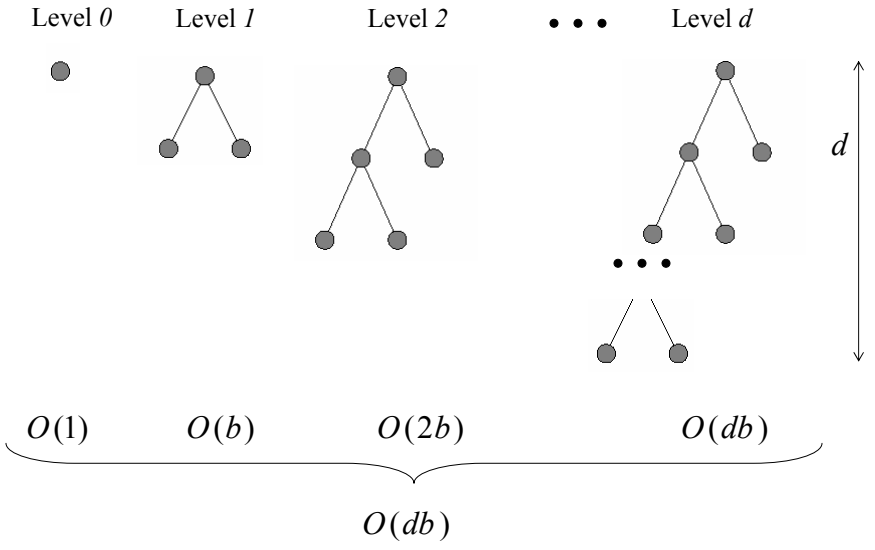
exponential in the depth of the solution *d*

worse than BFS, but asymptotically the same

- **Memory (space) complexity:**

?

IDA – memory complexity



Properties of IDA

- **Completeness:** **Yes.** The solution is reached if it exists.
  (the same as BFS)
- **Optimality:** **Yes**, for the shortest path.
  (the same as BFS)
- **Time complexity:**

$$O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d)$$

  exponential in the depth of the solution *d*
  worse than BFS, but asymptotically the same
- **Memory (space) complexity:**

$$O(db)$$

  much better than BFS