# 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**

- Due on Thursday before the lecture

**Course web page:**

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

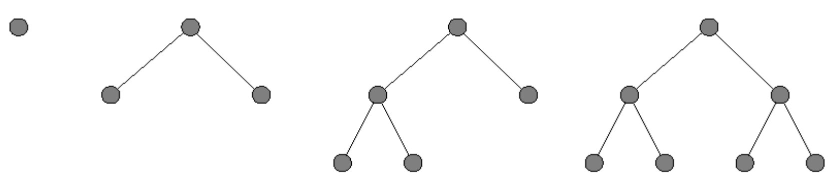
# Uninformed methods

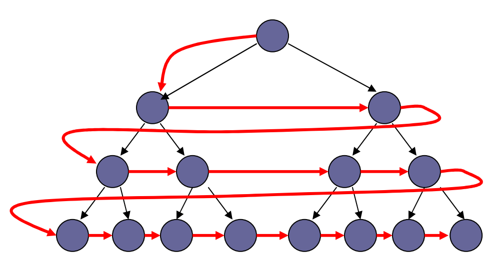
- Uninformed search methods use only information available in the problem definition
  - **Breadth-first search (BFS)**
  - **Depth-first search (DFS)**
  - **Iterative deepening (IDA)**
  - **Bi-directional search**
- **For the minimum cost path problem:**
  - **Uniform cost search**



# 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:**

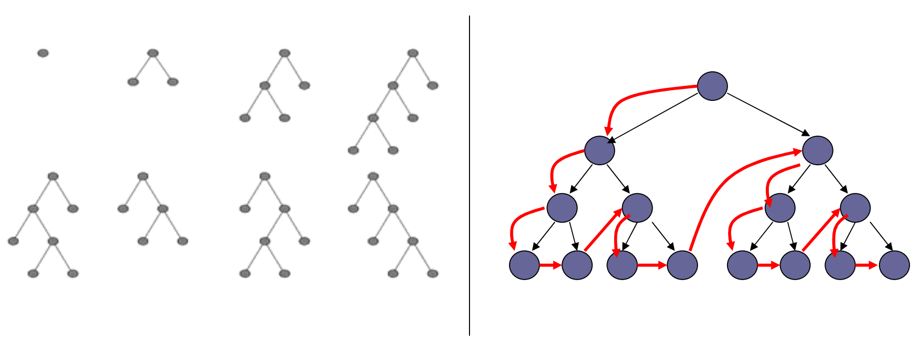
$$O(b^d)$$

**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:**

  $O(bm)$

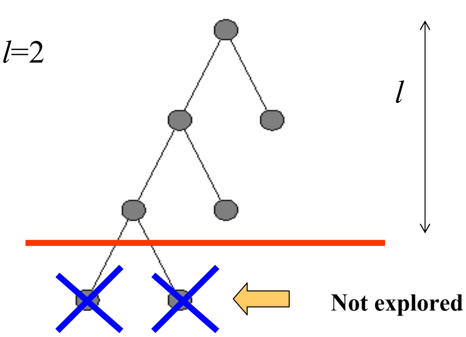
  **linear in the maximum depth of the search tree *m***



## 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

## Elimination of state repeats

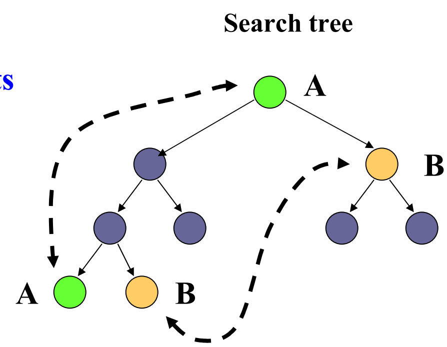
While searching the state space for the solution we can encounter the same state many times.

**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**






## 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.

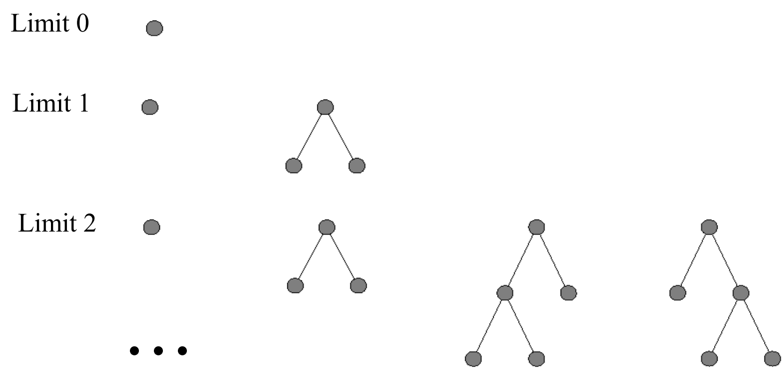
**Idea: try all depth limits in an increasing order.**

**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)

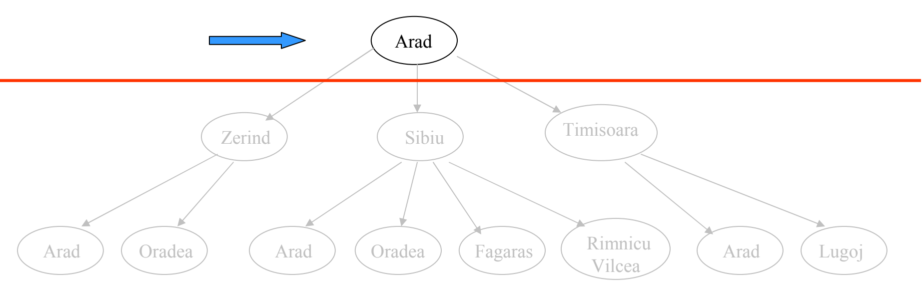
- Progressively increases the limit of the limited-depth depth-first search





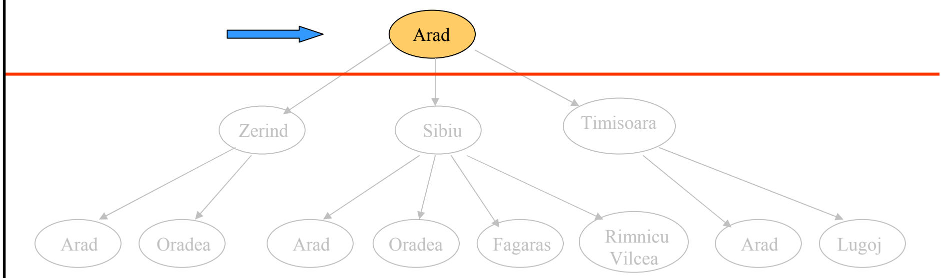

# Iterative deepening

**Cutoff depth = 0**



# Iterative deepening

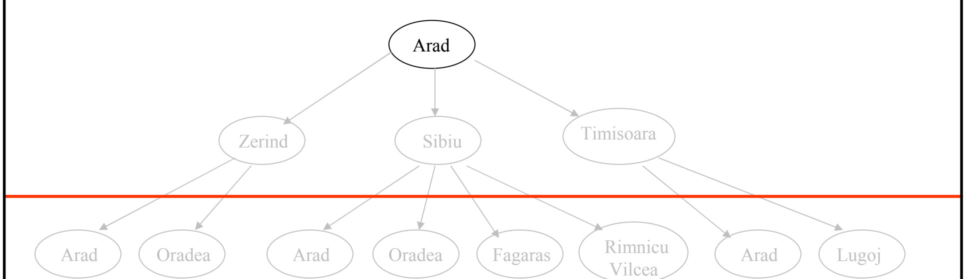
**Cutoff depth = 0**





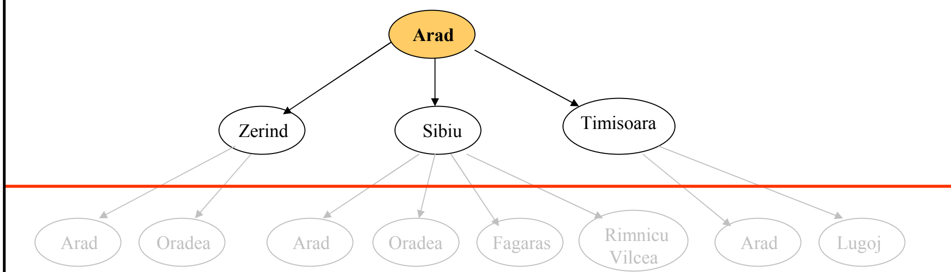

# Iterative deepening

**Cutoff depth = 1**



# Iterative deepening

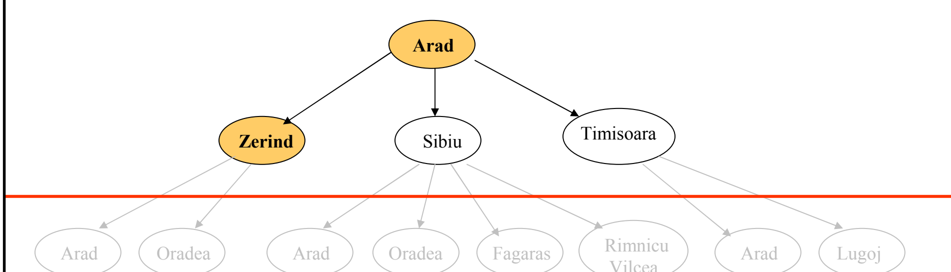
## Cutoff depth = 1





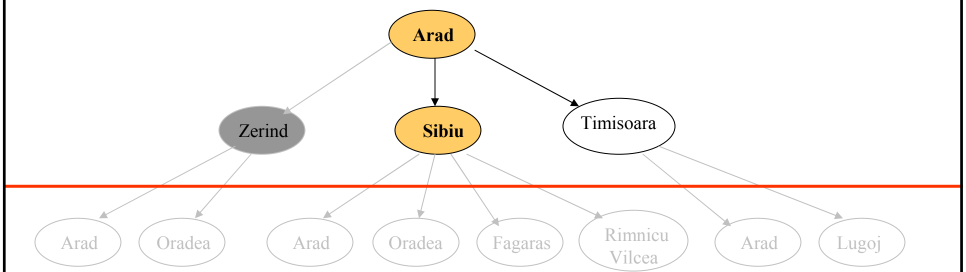

# Iterative deepening

## Cutoff depth = 1



# Iterative deepening

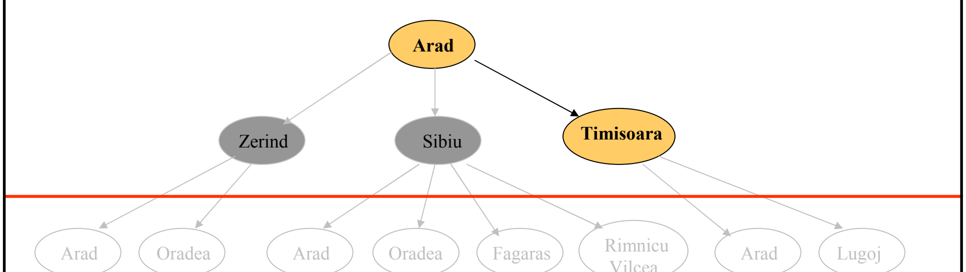
## Cutoff depth = 1





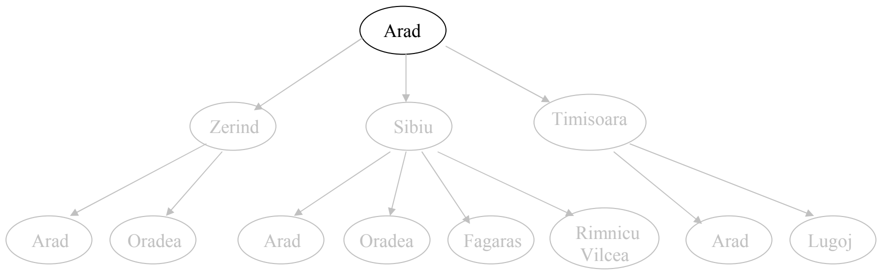

# Iterative deepening

## Cutoff depth = 1



# Iterative deepening

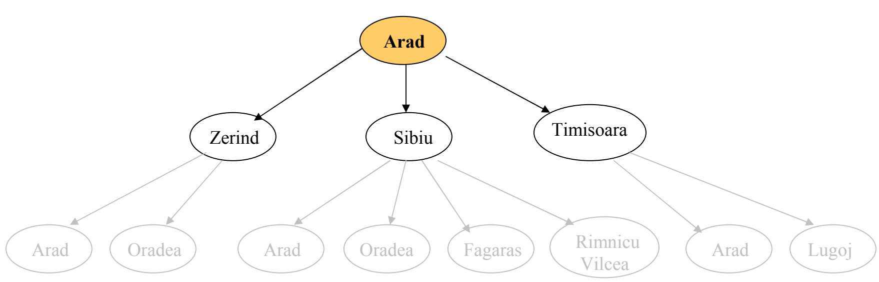
## Cutoff depth = 2





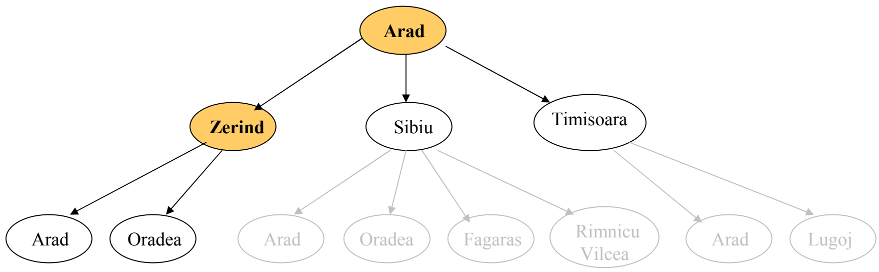

# Iterative deepening

## Cutoff depth = 2



# Iterative deepening

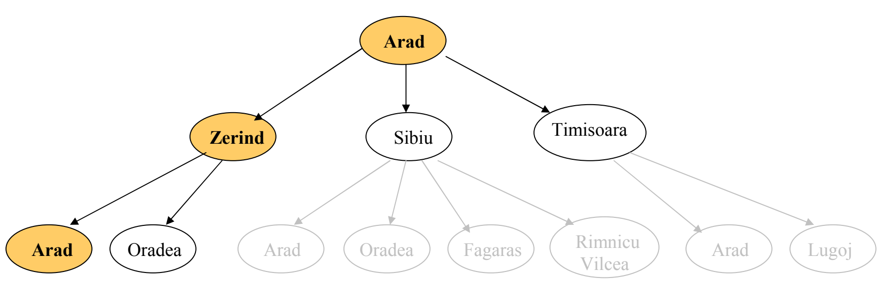
Cutoff depth = 2





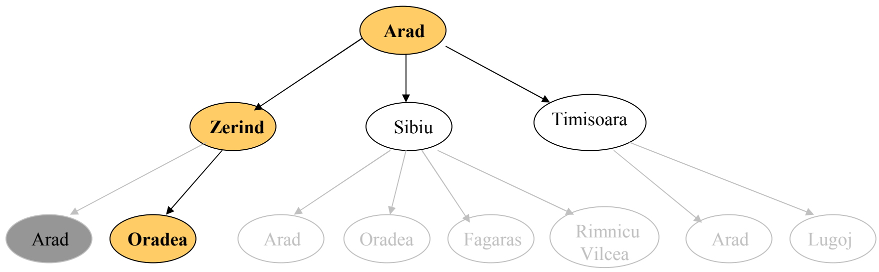

# Iterative deepening

Cutoff depth = 2



# Iterative deepening

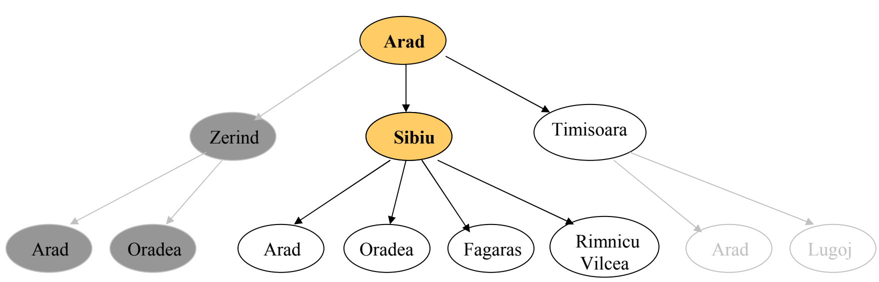
## Cutoff depth = 2






# Iterative deepening

## Cutoff depth = 2



## Properties of IDA

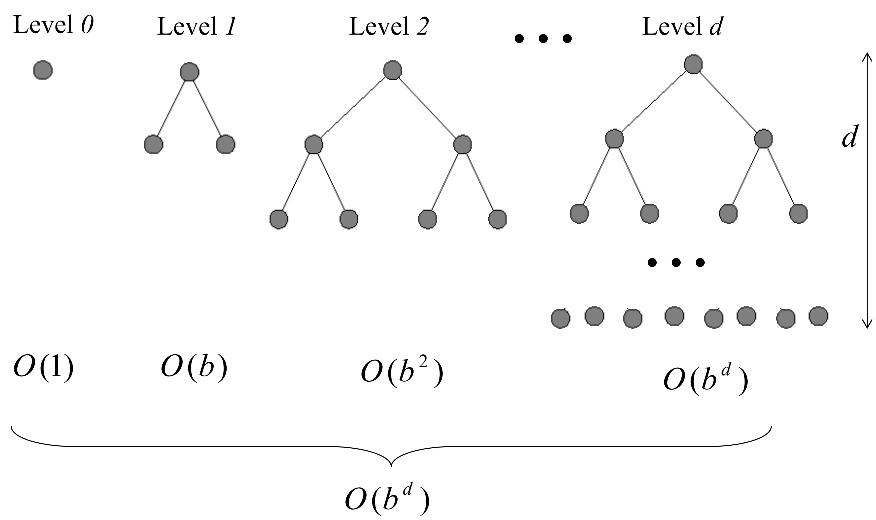
- **Completeness: ?**
- **Optimality: ?**
- **Time complexity:**
  **?**
- **Memory (space) complexity:**
  **?**



## 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

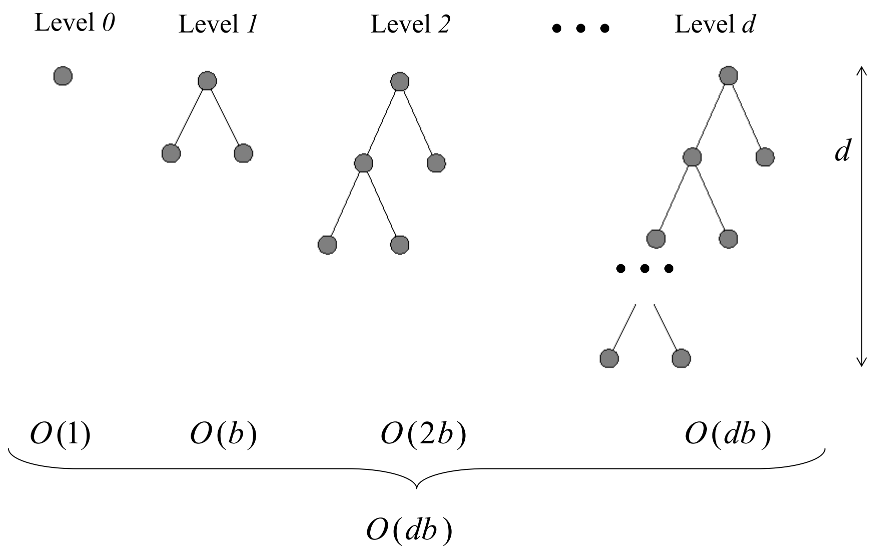





## Properties of IDA

- **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**

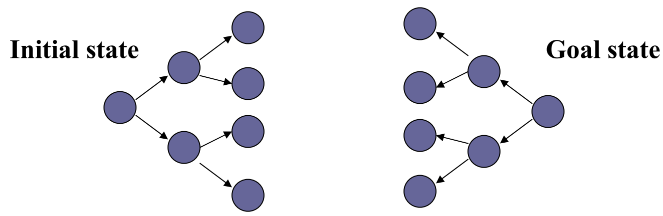
- **Memory (space) complexity:**

$$O(db)$$

  **much better than BFS**

# Bi-directional search

- In some search problems we want to find the path from the initial state to the **unique goal state** (e.g. traveler problem)
- **Bi-directional search idea:**



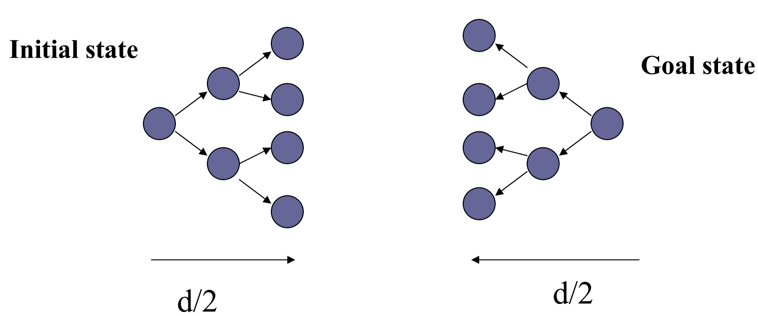

  - Search both from the initial state and the goal state;
  - Use inverse operators for the goal-initiated search.



# Bi-directional search

Why bidirectional search? What is the benefit? Assume BFS.

- Cut the depth of the search space by half

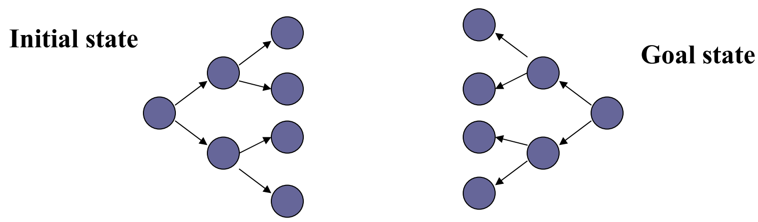



$O(b^{d/2})$ Time and memory complexity

# Bi-directional search

Why bidirectional search? What is the benefit? Assume BFS

- It cuts the depth of the search tree by half.





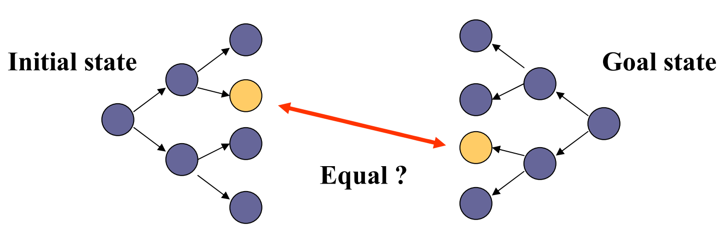

# Bi-directional search

Why bidirectional search? Assume BFS.

- It cuts the depth of the search tree by half.

What is necessary?

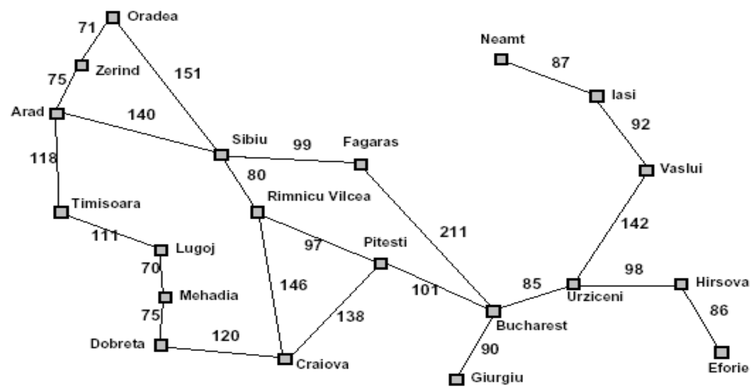
- Merge the solutions.




- How? The hash structure remembers the side of the tree the state was expanded first time. If the same state is reached from other side we have a solution.

# Minimum cost path search

## Traveler example with distances [km]




**Optimal path:** the shortest distance path from Arad to Bucharest



# Searching for the minimum cost path

- **General minimum cost path-search problem:**
  - **adds weights or costs** to operators (links)

"Intelligent" expansion of the search tree should be driven by the cost of the current (partially) built path

**Path cost function** $g(n)$; path cost from the initial state to $n$

**Search strategy:**

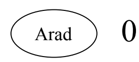
- Expand the leaf node with the minimum $g(n)$ first.
  - When operator costs are all equal to 1 it is equivalent to BFS
- The basic algorithm for finding the minimum cost path:
  - **Dijkstra's shortest path**
- In AI, the strategy goes under the name
  - **Uniform cost search**

# Uniform cost search

- Expand the node with the minimum path cost first
- **Implementation: a priority queue**

**queue** ⇒

| | ***g(n)*** |
|---|---|
| **Arad** | **0** |

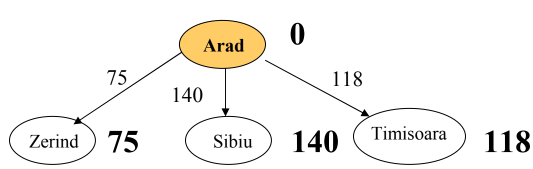





# Uniform cost search

**queue** ⇒

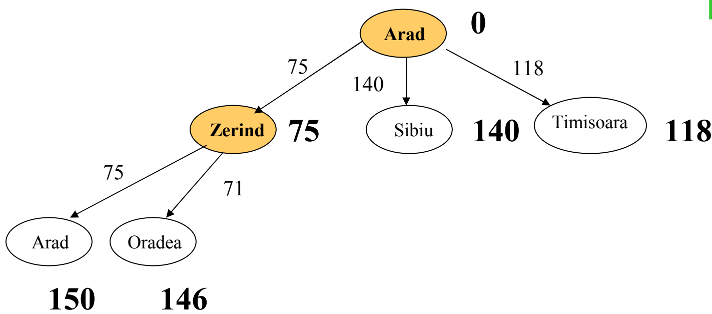
| | ***g(n)*** |
|---|---|
| **Zerind** | **75** |
| **Timisoara** | **118** |
| **Sibiu** | **140** |



# Uniform cost search

queue

| | *g(n)* |
|---|---|
| Timisoara | 118 |
| Sibiu | 140 |
| **Oradea** | **146** |
| **Arad** | **150** |

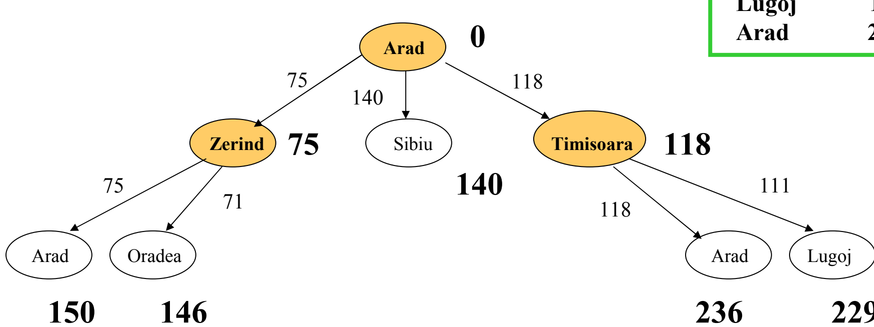





# Uniform cost search

queue

| | *g(n)* |
|---|---|
| Sibiu | 140 |
| Oradea | 146 |
| Arad | 150 |
| **Lugoj** | **129** |
| **Arad** | **236** |



## Properties of the uniform cost search

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



## Properties of the uniform cost search

- **Completeness: Yes**, assuming that operator costs are non-negative (the cost of path never decreases)
  $$g(n) \leq g(\text{successor }(n))$$
- **Optimality: Yes.** Returns the least-cost path.
- **Time complexity:**
  **number of nodes with the cost *g(n)* smaller than the optimal cost**
- **Memory (space) complexity:**
  **number of nodes with the cost *g(n)* smaller than the optimal cost**

## Elimination of state repeats

**Idea:**

- A node is redundant and can be eliminated if there is another node with exactly the same state and a shorter path from the initial state

**Assuming positive costs:**

- If the state has already been expanded, is there a shorter path to that node ?



## Elimination of state repeats

**Idea:**

- A node is redundant and can be eliminated if there is another node with exactly the same state and a shorter path from the initial state

**Assuming positive costs:**

- If the state was already expanded, is there a a shorter path to that node ?
- **No !**

**Implementation:**

- Marking with the hash table