

# CS 1571 Introduction to AI

## Lecture 4

### Uninformed search methods I.

**Milos Hauskrecht**

[milos@cs.pitt.edu](mailto:milos@cs.pitt.edu)

5329 Sennott Square

### Announcements

#### Homework assignment 1 is out

- Due on Thursday, September 17, 2009 before the lecture
- Theoretical and programming part:
  - Programming part involves Puzzle 8 problem.

#### Course web page:

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

## 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

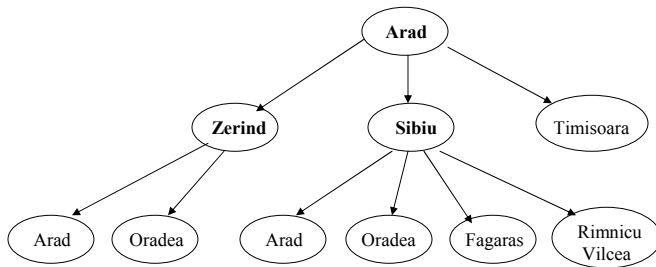
## Formulating a search problem

- 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

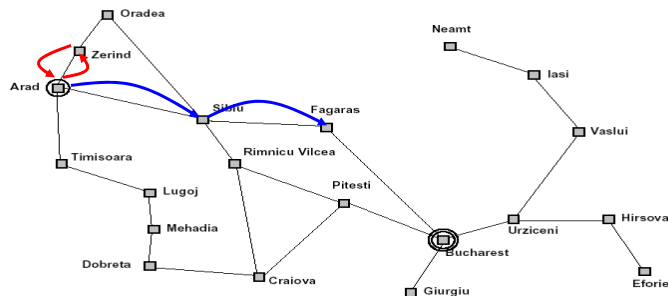


## Search process

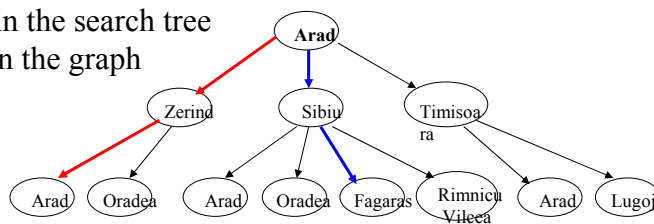
- Exploration of the state space through successive application of operators from the initial state
- A **search tree** = a kind of (search) exploration trace, branches corresponding to explored paths, and leaf nodes corresponding to exploration fringe



## Search tree

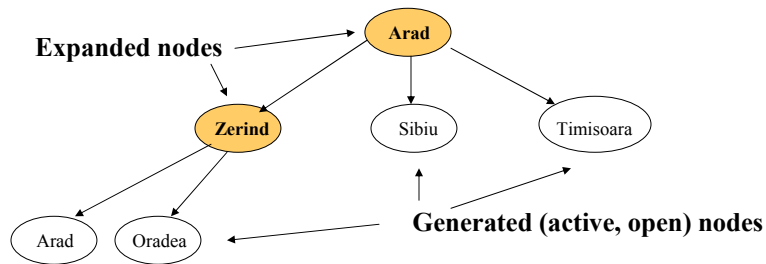


A branch in the search tree  
= path in the graph



## 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 next **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**



## 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

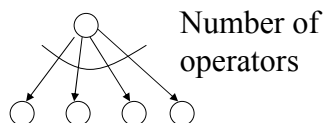
### Properties of search methods :

- **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.

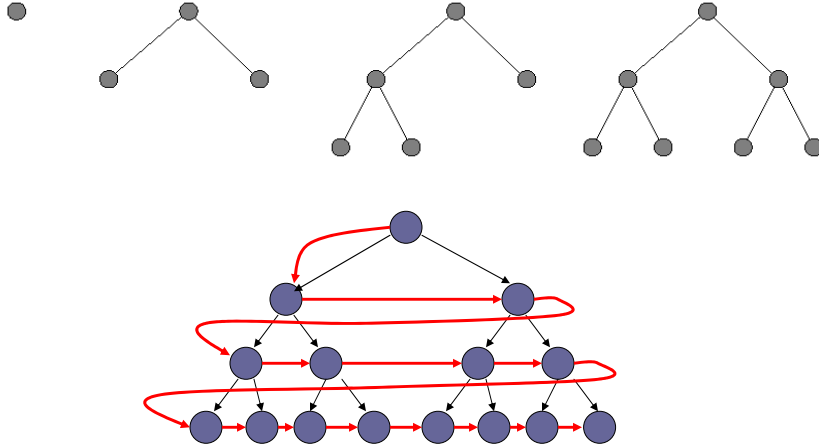
- **Space and time complexity.**
  - **Complexities** are measured in terms of parameters:
    - $b$  – maximum branching factor
    - $d$  – depth of the optimal solution
    - $m$  – maximum depth of the state space

### Branching factor

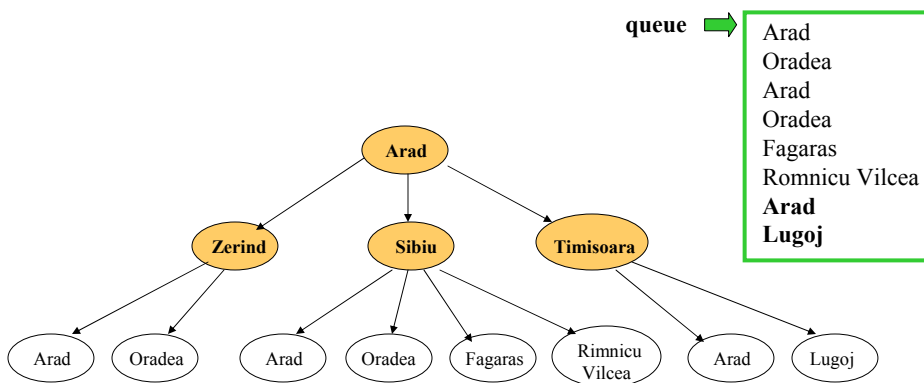


## Breadth first search (BFS)

- The shallowest node is expanded first



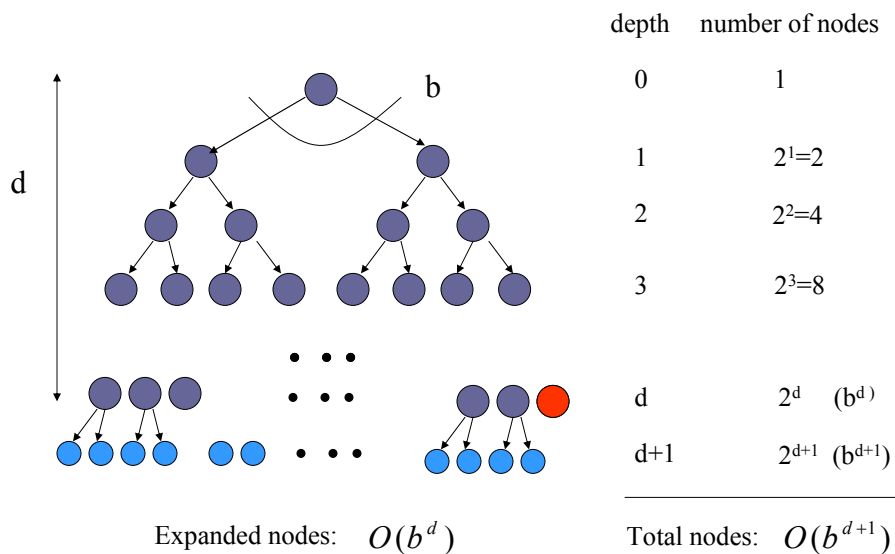
## Breadth-first search



## 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



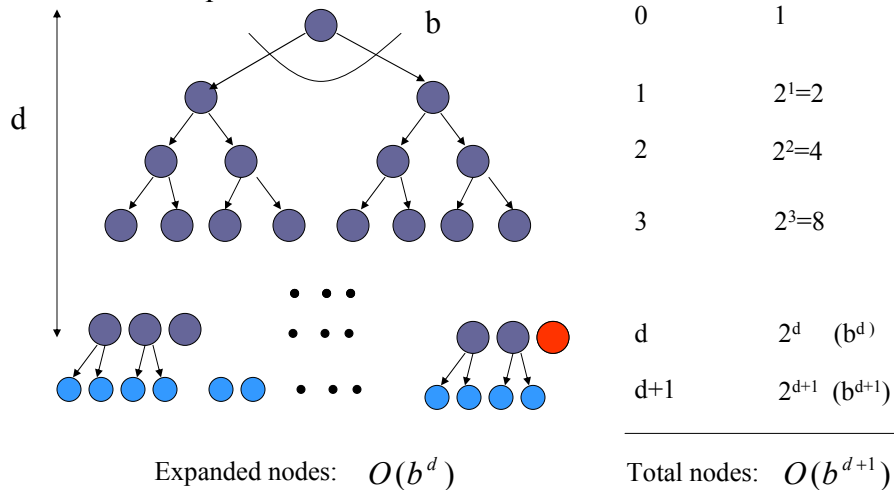
## Properties of breadth-first search

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

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





## Properties of breadth-first search

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

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

**exponential in the depth of the solution  $d$**

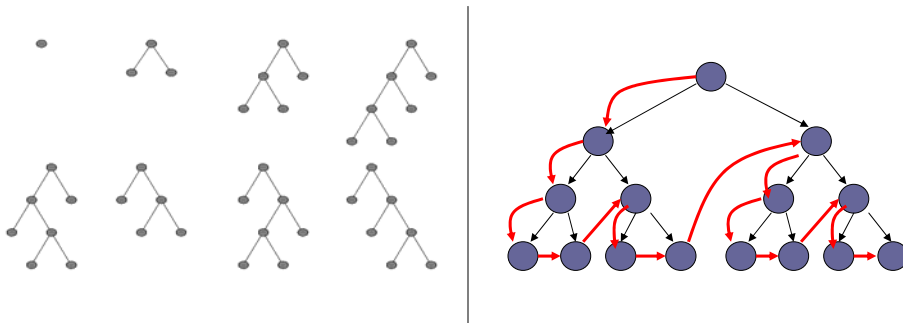
- **Memory (space) complexity:**

$$O(b^d)$$

**nodes are kept in the memory**

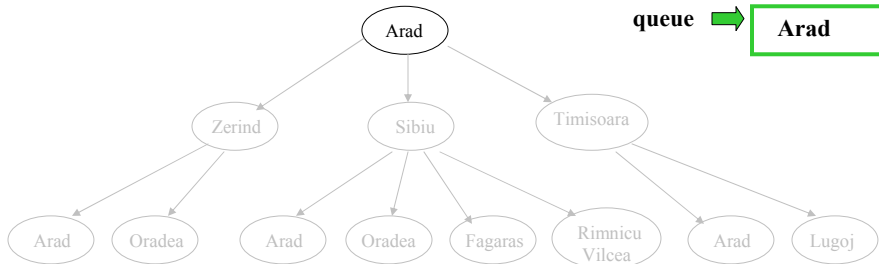
## Depth-first search (DFS)

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

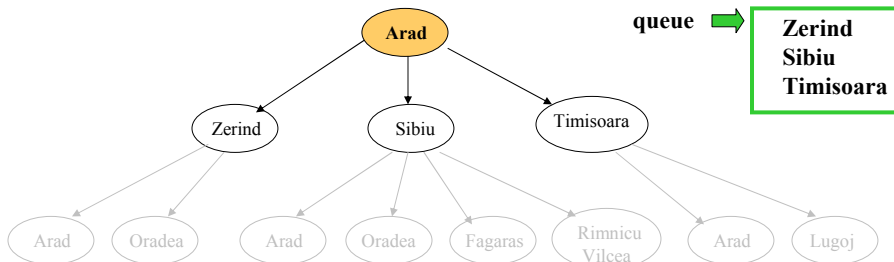


## Depth-first search

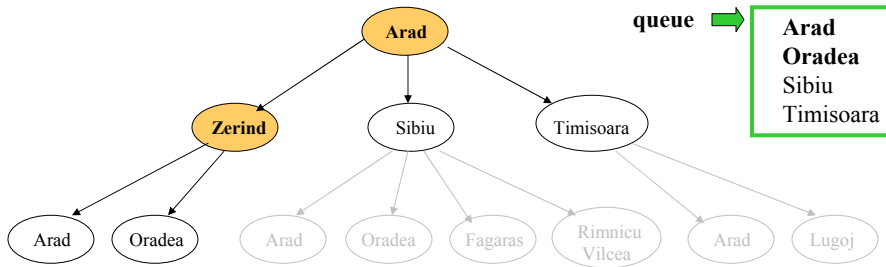
- The deepest node is expanded first
- Implementation: put successors to the beginning of the queue



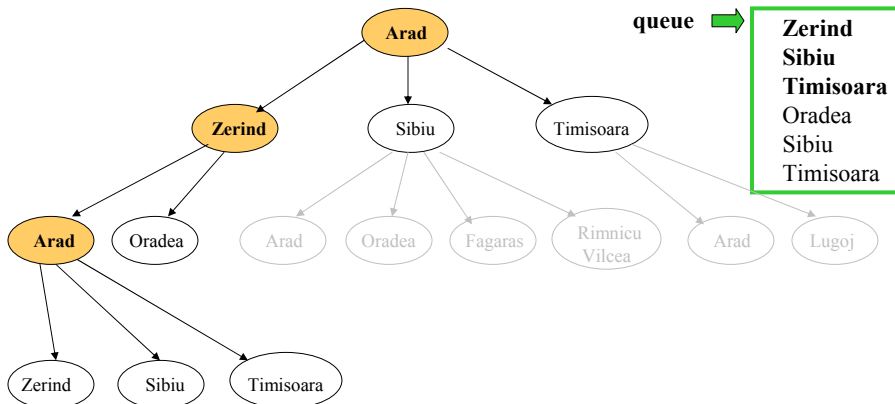
## Depth-first search



## Depth-first search



## Depth-first search



**Note:** Arad – Zerind – Arad cycle

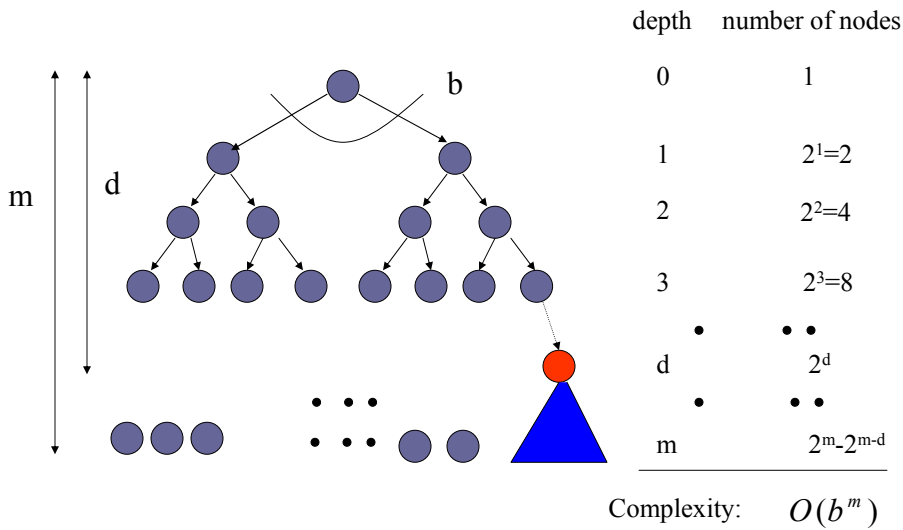
## Properties of depth-first search

- **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



CS 1571 Intro to AI

M. Hauskrecht


## 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:** ?

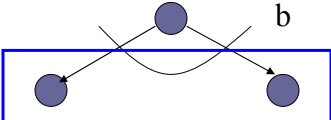
CS 1571 Intro to AI

M. Hauskrecht

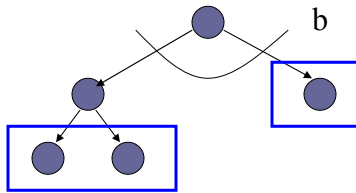
## DFS – memory complexity

		depth	number of nodes kept
	b	0	1

## DFS – memory complexity

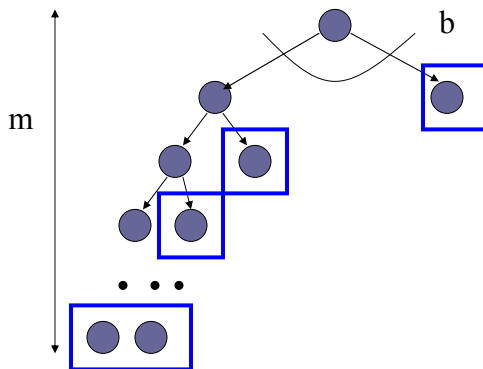
		depth	number of nodes kept
	b	0	0
		1	$2 = b$

## DFS – memory complexity



depth	number of nodes kept
0	0
1	$1 = (b-1)$
2	$2 = b$

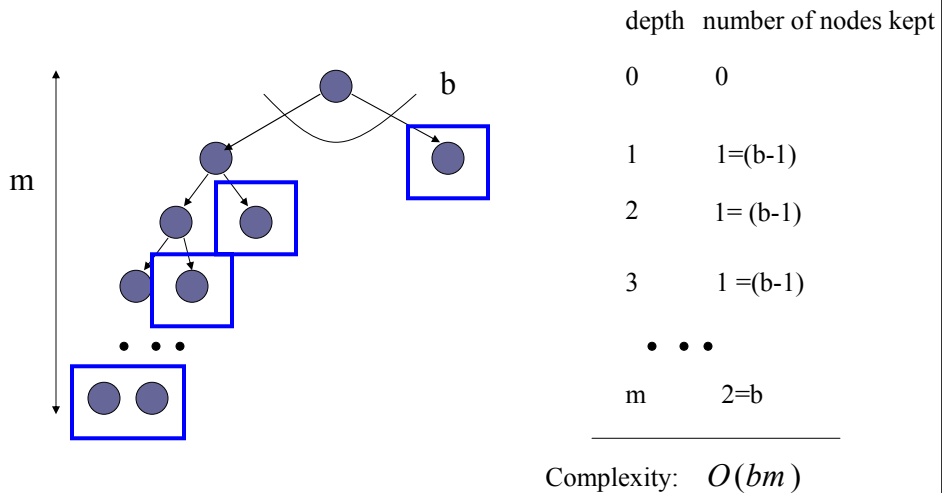
## DFS – memory complexity



depth	number of nodes kept
0	0
1	1
2	1
3	1
...	...
m	$2=b$

Complexity:

## DFS – memory complexity



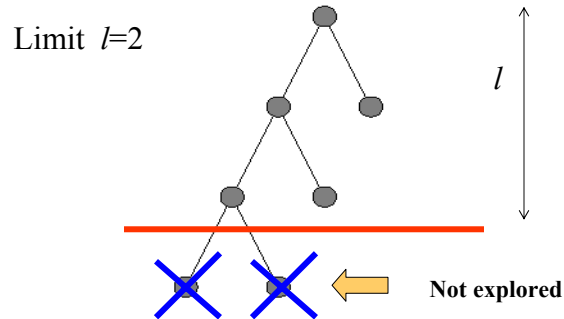
## Properties of depth-first search

- **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:**  
 $O(bm)$   
**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



- **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?
  - **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

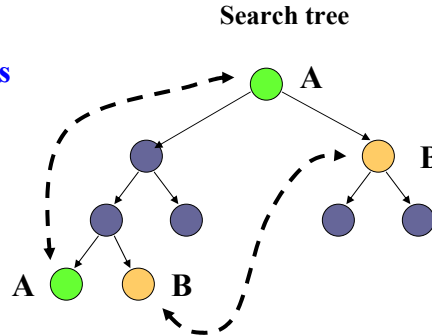
While searching the state space for the solution we can encounter the same state many times.

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

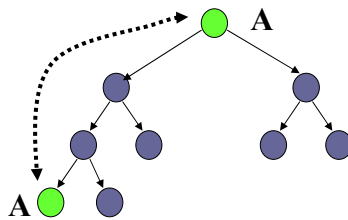
**Two possible cases:**

**(A) Cyclic state repeats**

**(B) Non-cyclic state repeats**



## Elimination of cycles

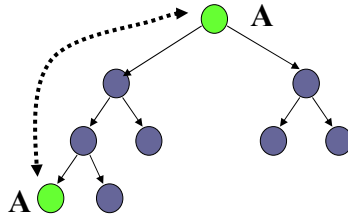


**Case A:** Corresponds to the path with a cycle

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? **No !!**

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

## Elimination of cycles

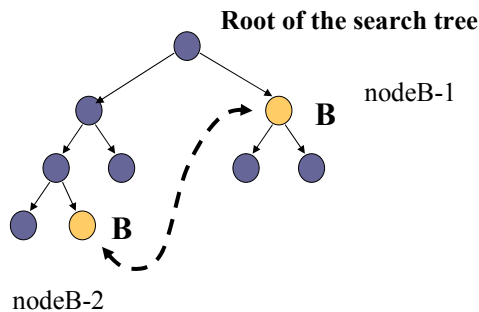


### How to check for cyclic state repeats:

- 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

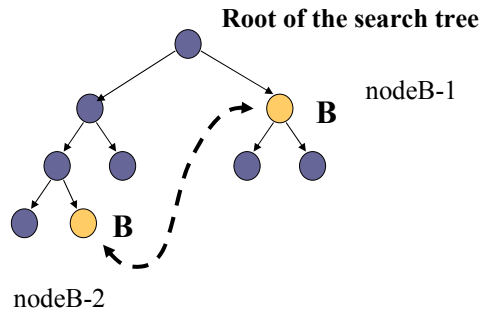


**Case B:** nodes with the same state are not on the same path from the initial state

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

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

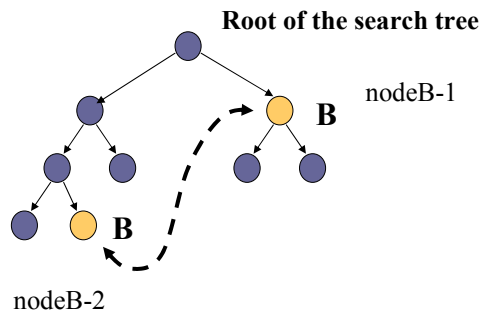
## Elimination of non-cyclic state repeats



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)**

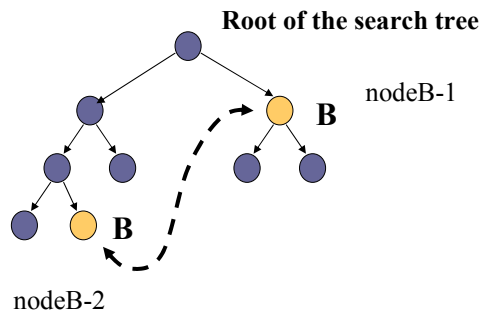
- we can safely eliminate all second, third, fourth, etc. occurrences of the same state
- this rule covers both cyclic and non-cyclic repeats !!!

**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

**Use hash table to implement marking**

## Elimination of non-cyclic state repeats



**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.)