CS 561 Artificial Intelligence
Lecture

SOLVING PROBLEMS BY SEARCHING

Rashmi Dutta Baruah

Department of Computer Science & Engineering

IIT Guwahati

OUTLINE

- Defining problems and solutions
- Searching for solutions
  - Uninformed Search
    - Breadth-first Search
    - Uniform-cost search
    - Depth-first Search
    - Depth-limited Search
    - Iterative Deepening Depth-first Search
    - Bidirectional Search

DEFINING PROBLEMS AND SOLUTIONS

- Given the Goal and Problem definition, a *problem-solving agent* (a kind of Goal-based agent) *searches* for a solution in terms of sequence of actions that would solve the problem, and then *executes* the actions one at a time.
- Problem can be defined by five components:

(1) Initial state: the state at which the agent starts.

(2) Actions: set of possible actions at each state.

`ACTION(s)` = set of actions that can be executed in `s`.

(3) Transition model: describes the result of each action.

`RESULT(s,a)` = state that results from doing action `a` in state `s`

The initial state, actions, and transition model *implicitly* define the state space of the problem. The set of all states reachable from the initial state by any sequence of actions.

DEFINING PROBLEMS AND SOLUTIONS

(4) Goal test: determines whether a given state is a goal state.

(5) Path cost function: gives a numeric cost to each path (solution), specifies the quality of an accepted solution

Path cost = sum of the costs of individual actions along the path

`c(s,a,s')` = step cost of taking action `a` in state `s` to reach state s', `c ≥ 0`.

- A solution is action sequence that leads from the initial state to goal state.
- Solution with lowest path cost is an optimal solution.

State space	=	directed graph (implicit)
States	=	nodes
Actions	=	links
Searching solution	=	searching a path from the start node to goal node

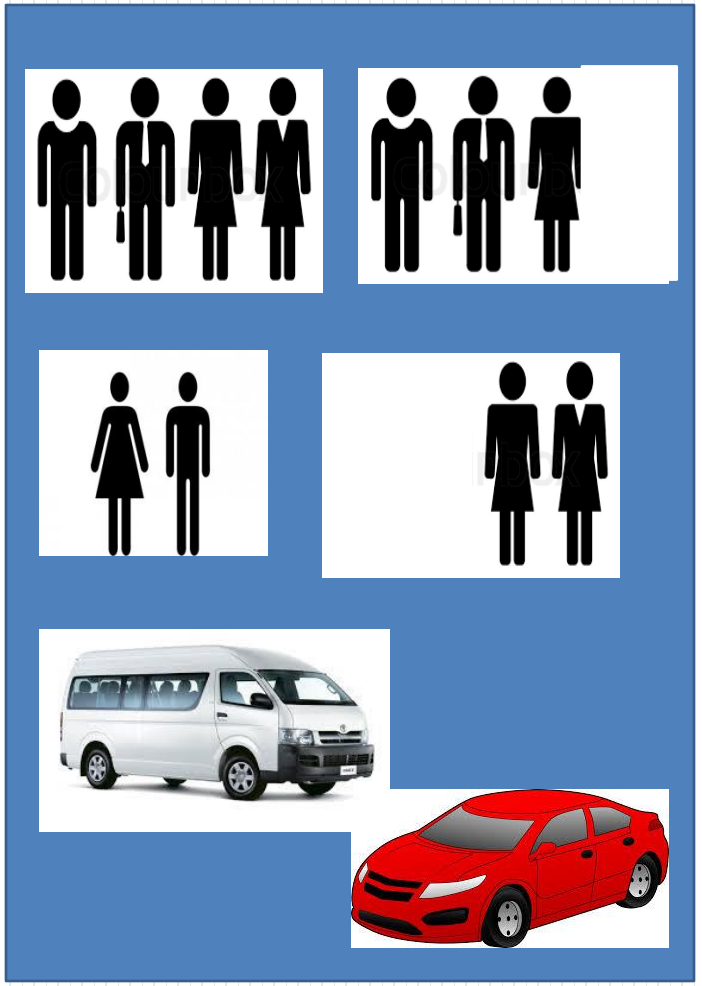
ENVIRONMENT PROPERTIES

- For a problem-solving agent the task environment is assumed to have following properties:
  - Fully observable
  - Deterministic
  - Sequential
  - Static
  - Discrete
  - Single agent

Example: Transportation Scheduling

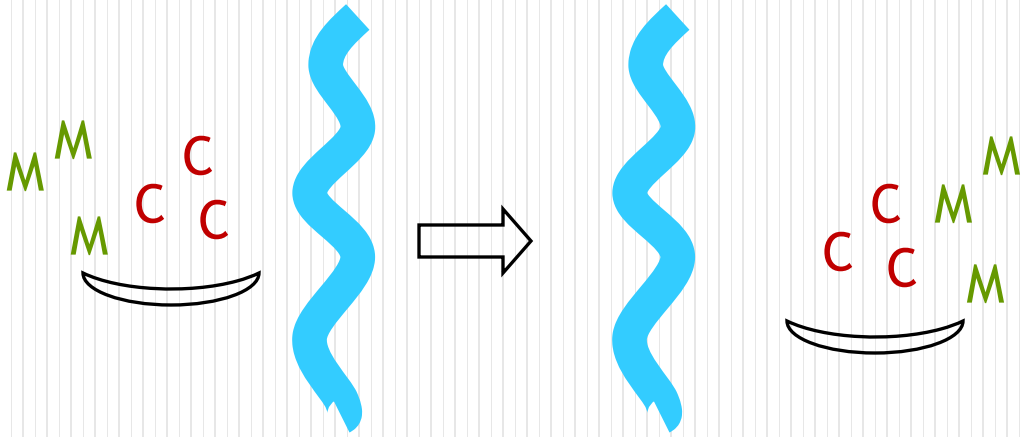
- Given:
  - a set of space points
  - an initial distribution of objects in the points
  - transportation facilities with given capacities
  - final distribution of objects in the points

Find an optimal sequence of transportations between the space points such that the final distribution of objects in these points can be attained without violating a set of given constraints.



EXAMPLE: MISSIONARIES AND CANNIBALS

- Subclass of transportation scheduling problem:
  - 'difficult crossing' problem : (M&C)
- The missionaries and cannibals wish to cross a river
- They have a boat that can hold two people
- It is unsafe to have cannibals outnumber missionaries



Initial state

Goal state

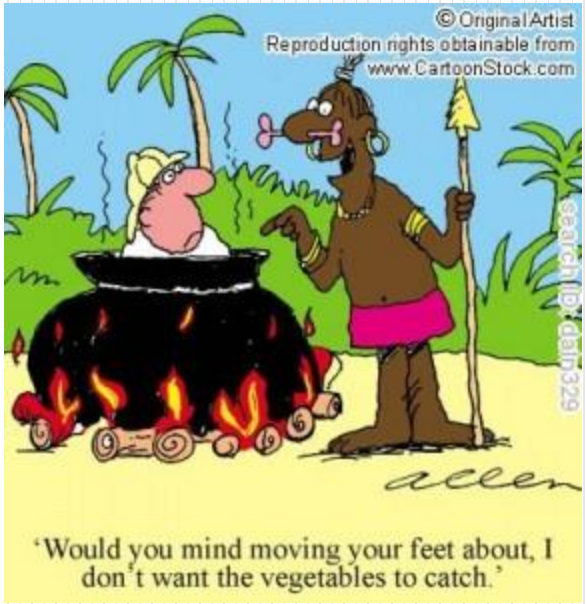


Image source:
http://www.overthinkingit.com/2009/10/16/best-fictional-food/3/

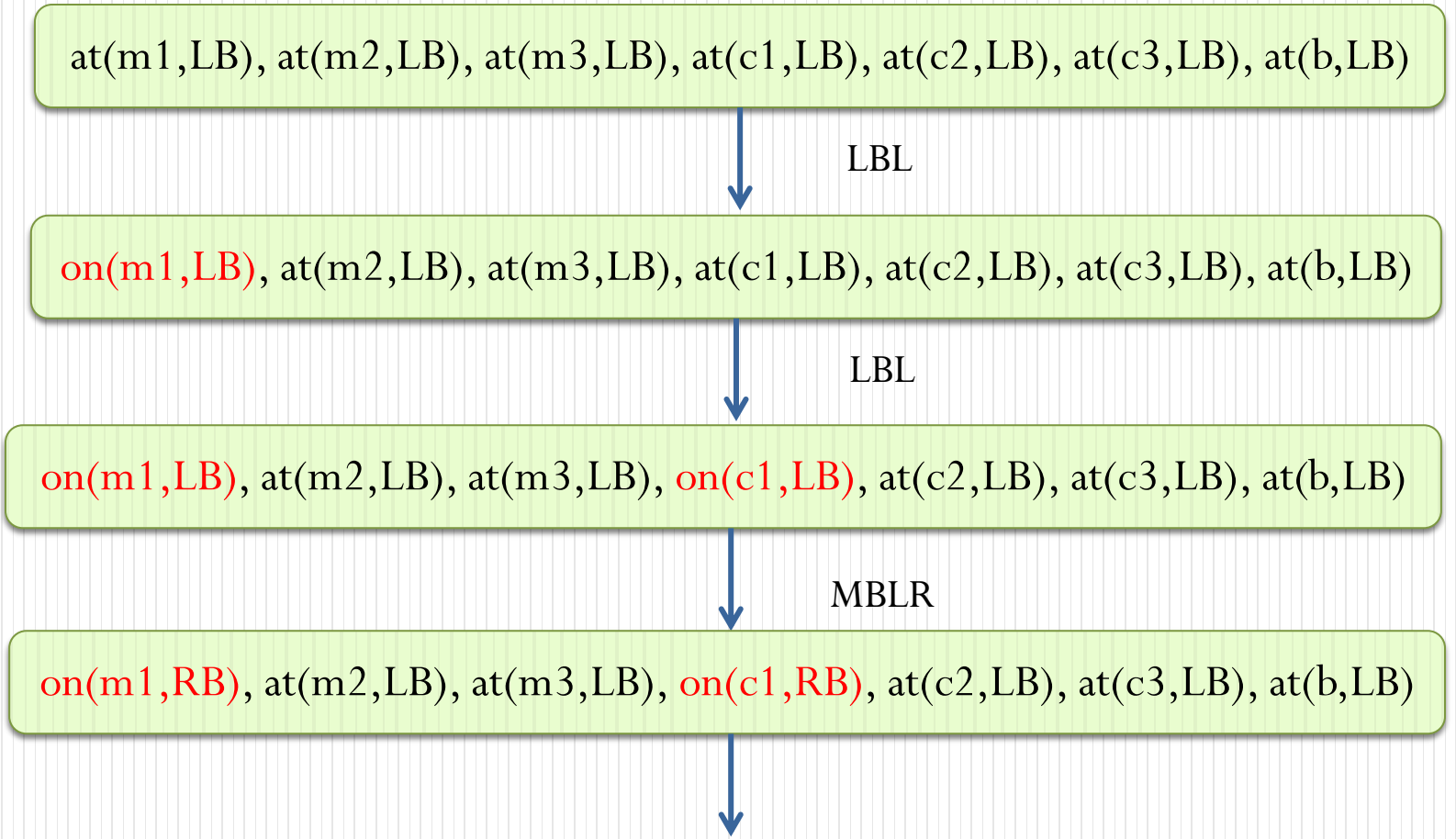
EXAMPLE: MISSIONARIES AND CANNIBALS

- Defining the problem: **Problem formulation 1**
- *m1*, *m2*, *m3* :Missionaries *c1*, *c2*, *c3* : Cannibals, *b* : Boat
- *LB*: left bank, *RB*: right bank
- States can be described by
  - *at(m1,LB)* misssionary m1 is at left bank, *at(b,LB)* boat *b* is at the left bank.
  - *on(c1, b)* cannibal c1 is on the boat
- Initial state: $s_0 = at(m1,LB), \ldots, at(c1,LB), \ldots at(b,LB)$
- Goal state: $s_G = at(m1,RB), \ldots, at(c1,RB), \ldots at(b,RB)$

EXAMPLE: MISSIONARIES AND CANNIBALS

- Actions:
  - Load boat at left, one individual at a time (LBL)
  - Move boat across the river from left to right (MBLR)
  - Unload boat at right, one individual at a time (UBR)
  - Similarly we have three more similar actions: LBR, MBRL, UBL
- These actions can be performed under certain constaints, for example:
  - LBL can be performed under the constraint $M_b+C_b \leq 2$ where M_b and C_b are the number of missionaries and number of cannibals on the boat
  - MBLR can be performed under the constraint $M_b+C_b > 0$

EXAMPLE: MISSIONARIES AND CANNIBALS

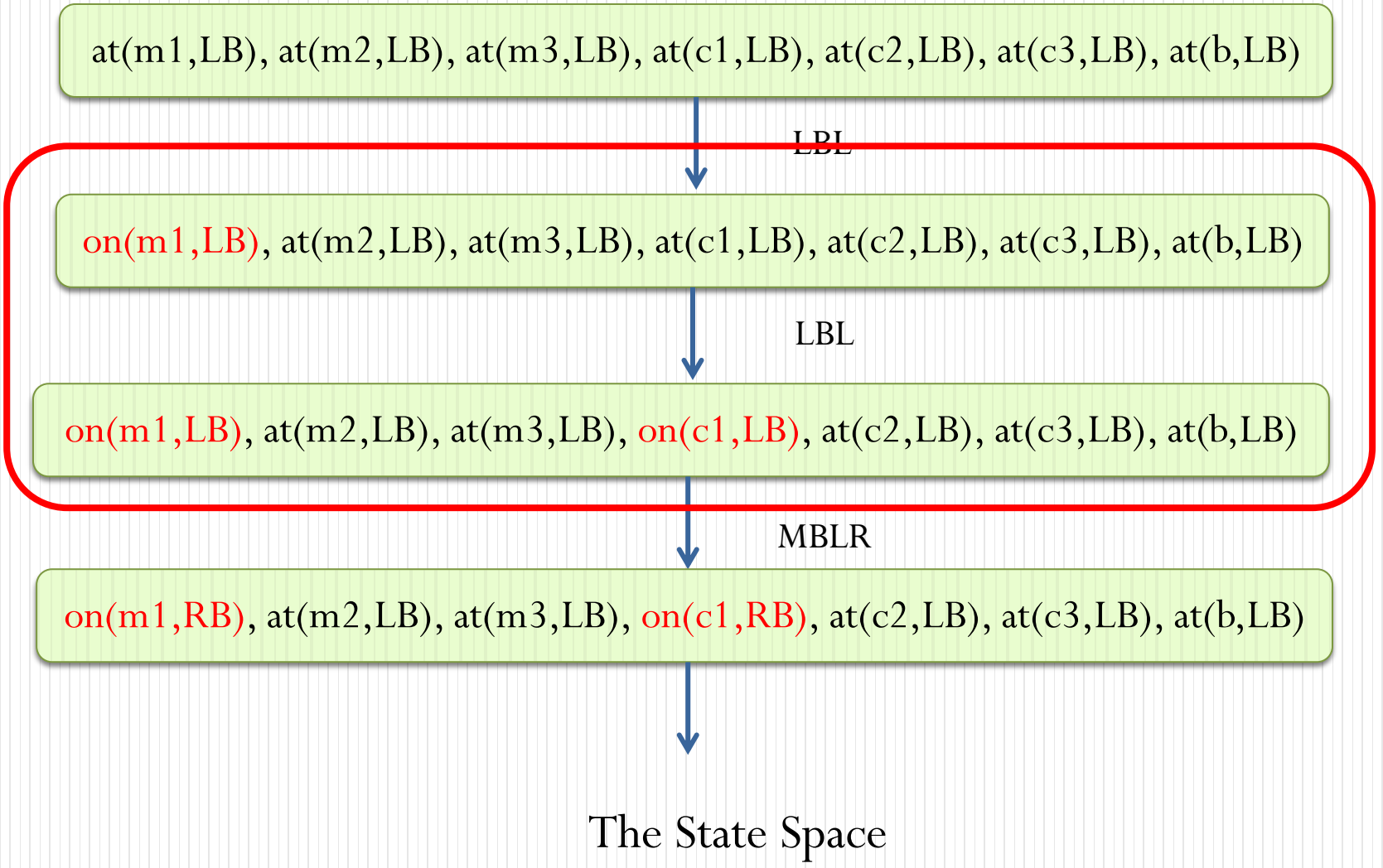


The State Space

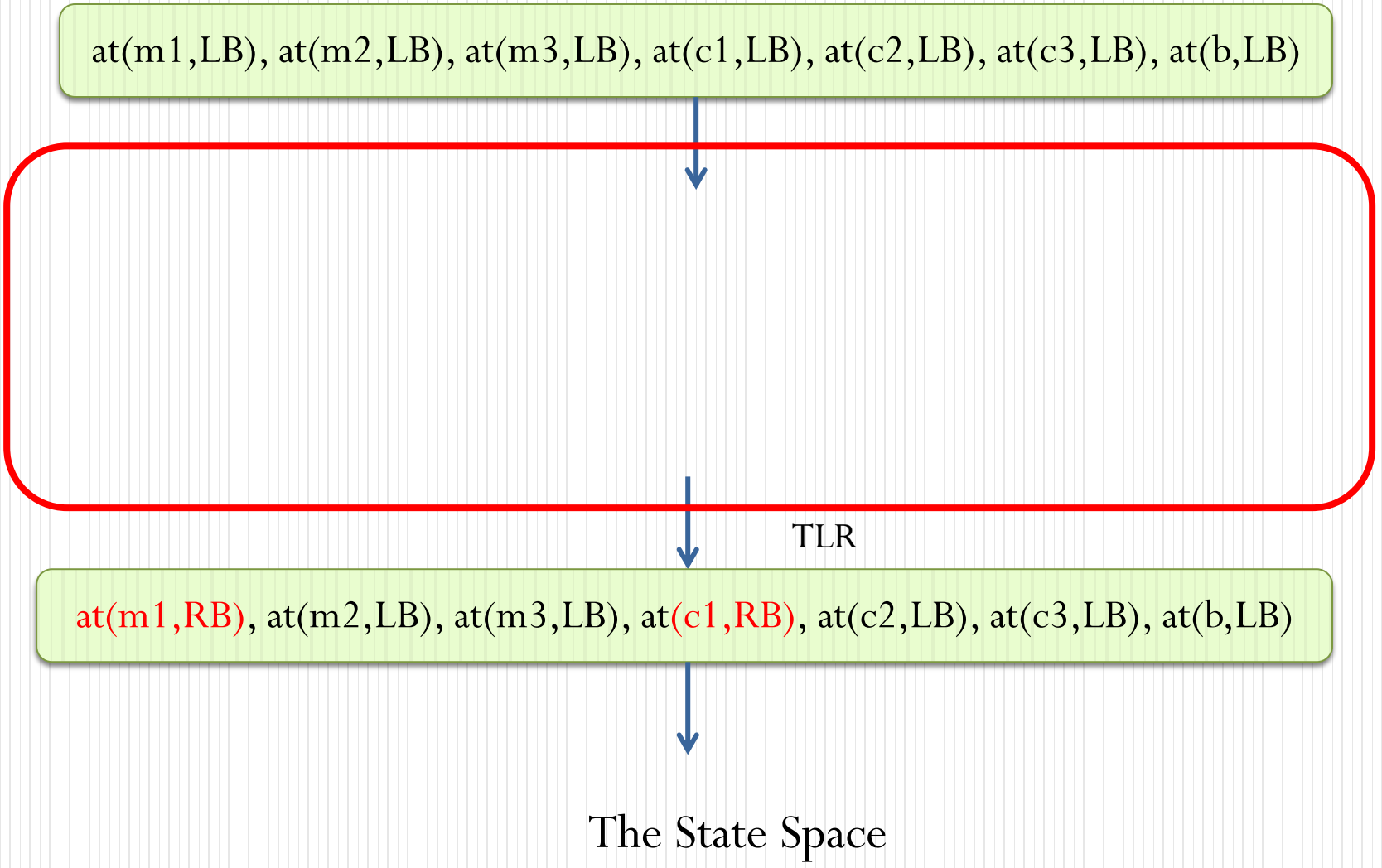
EXAMPLE: MISSIONARIES AND CANNIBALS

- Defining the problem: **Problem formulation 2**
- *m1*, *m2*, *m3* :Missionaries *c1*, *c2*, *c3* : Cannibals, *b* : Boat
- *LB*: left bank, *RB*: right bank
- States can be described by
  - *at(m1,LB)* misssionary m1 is at left bank, *at(b,LB)* boat *b* is at the left bank.
  - Initial state: s_0 = *at(m1,LB), …,at(c1,LB), …at(b,LB)*
  - Goal state: s_G = *at(m1,RB), …,at(c1,RB), …at(b,RB)*
- Actions:
  - Transfer a set of individuals from left to right (TLR)
  - Transfer a set of individuals from right to left (TRL)

EXAMPLE: MISSIONARIES AND CANNIBALS



EXAMPLE: MISSIONARIES AND CANNIBALS



EXAMPLE: MISSIONARIES AND CANNIBALS

- Defining the problem: **Problem formulation 3**
- The use of distinct use of individuals in the formulation is not used.
- Only the entities M_L M_R M_b, C_L, C_R, C_b are sufficient to use. (idea is to use those elements that are necessary for expressing the rules of action, that define the permissible transitions between states)
- It is sufficient to consider either M_L, M_b, C_L, C_b or M_R , M_b C_R, C_b
- Also, at(b, LB) and at (b, RB) can be represented by a variable B = 1(boat at left bank) B = 0 (boat at right bank).

EXAMPLE: MISSIONARIES AND CANNIBALS

- 3 missionaries, 3 cannibals, 1 boat
- State: (m c b) number of missionaries, cannibals, and boats on the initial bank of river.
  - Initial state is (3 3 1)
  - Goal state is (0 0 0)
- Actions are addition or subtraction of the vectors
  (1 0 1), (2 0 1), (0 1 1), (0 2 1), (1 1 1)
  - Such that the result is between (0 0 0) and (3 3 1)
- A possible solution

  (331) → (310) → (321) → (300) → (311) →
  (110)→(221)→(020) → (031) → (010) → (021)→000

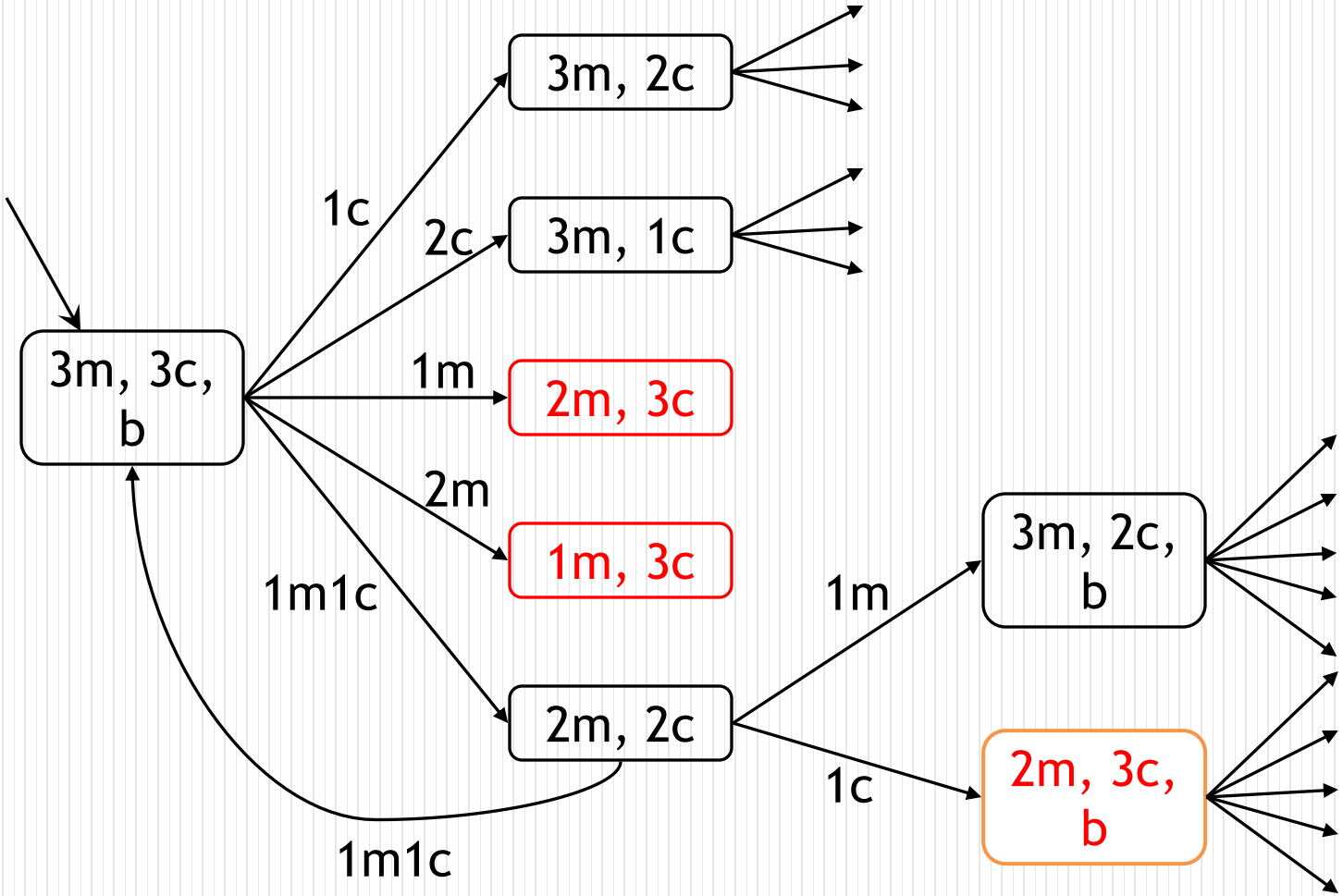
EXAMPLE: MISSIONARIES AND CANNIBALS

Another Representation

- Initial state: 3m,3c,b
- Goal state: 0m,0c

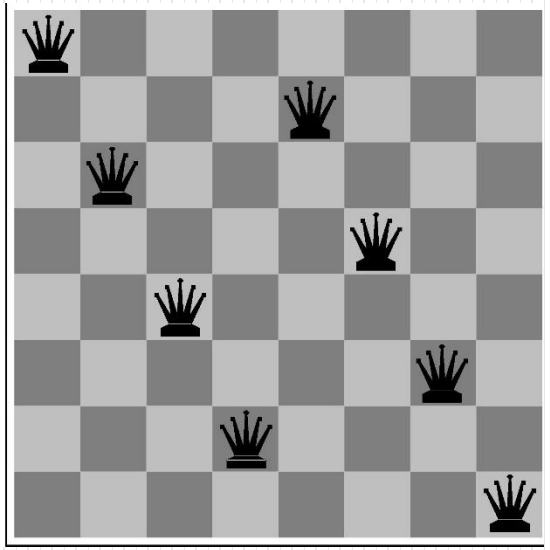
- Five possible actions are:
  - boat takes 1 missionary across river (1m)
  - boat takes 1 cannibal across river (1c)
  - boat takes 2 missionaries across river (2m)
  - boat takes 2 cannibals across river (2c)
  - boat takes 1 missionary and 1 cannibal across river (1m1c)
- We don't have to specify direction as only one of these will be possible at any given time

The state space



EXAMPLE: 8-QUEENS PROBLEM

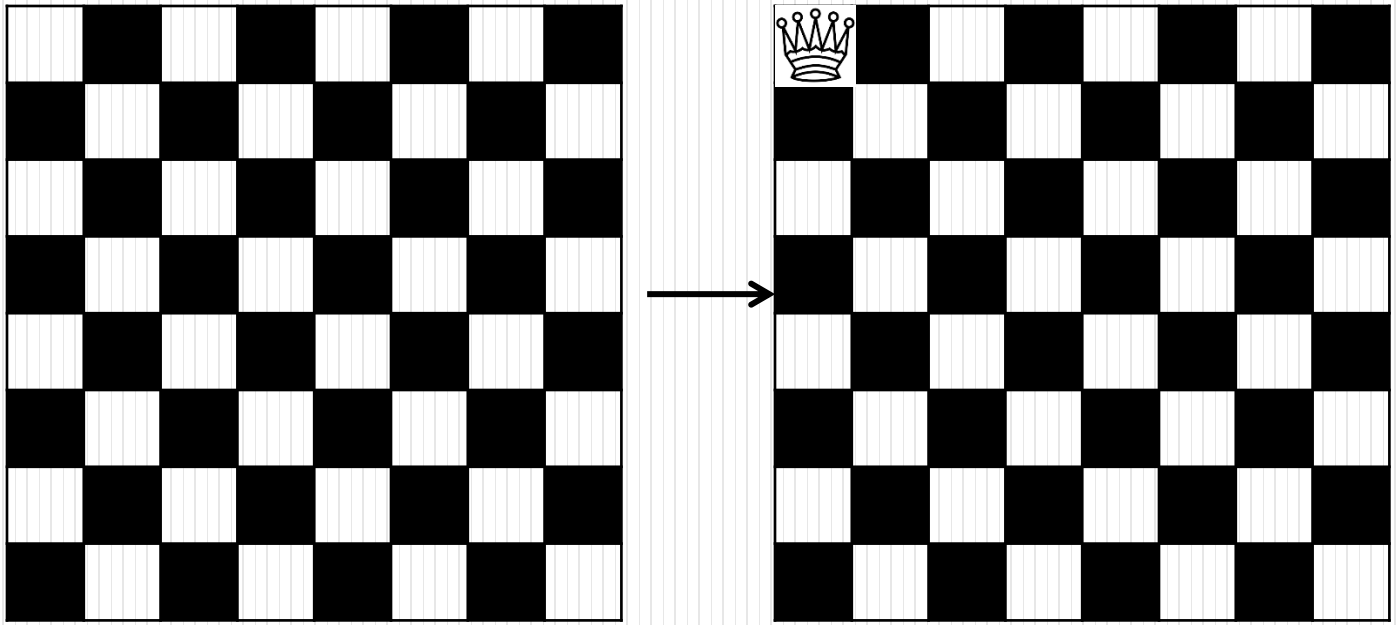
- Place 8 queens on a chessboard such that no queens attacks any other. A queen attacks any piece in the same row, column, or diagonal.
- States
  - any arrangement of `n<=8` queens
  - OR arrangement of `n<=8` queens in leftmost n columns, 1 per column such that no queen attacks any other.
- Initial state
  - No queen on the board.
- Actions
  - Add queen to any empty square
  - OR add queen to left most empty square such that it is not attacked



EXAMPLE: 8-QUEENS PROBLEM

- Transition model
  - returns the board with a queen added to specified square.
- Goal test
  - 8 queens on the board, none attacked.
- Path cost
  - 1 per move, however its of no interest because only the final state counts

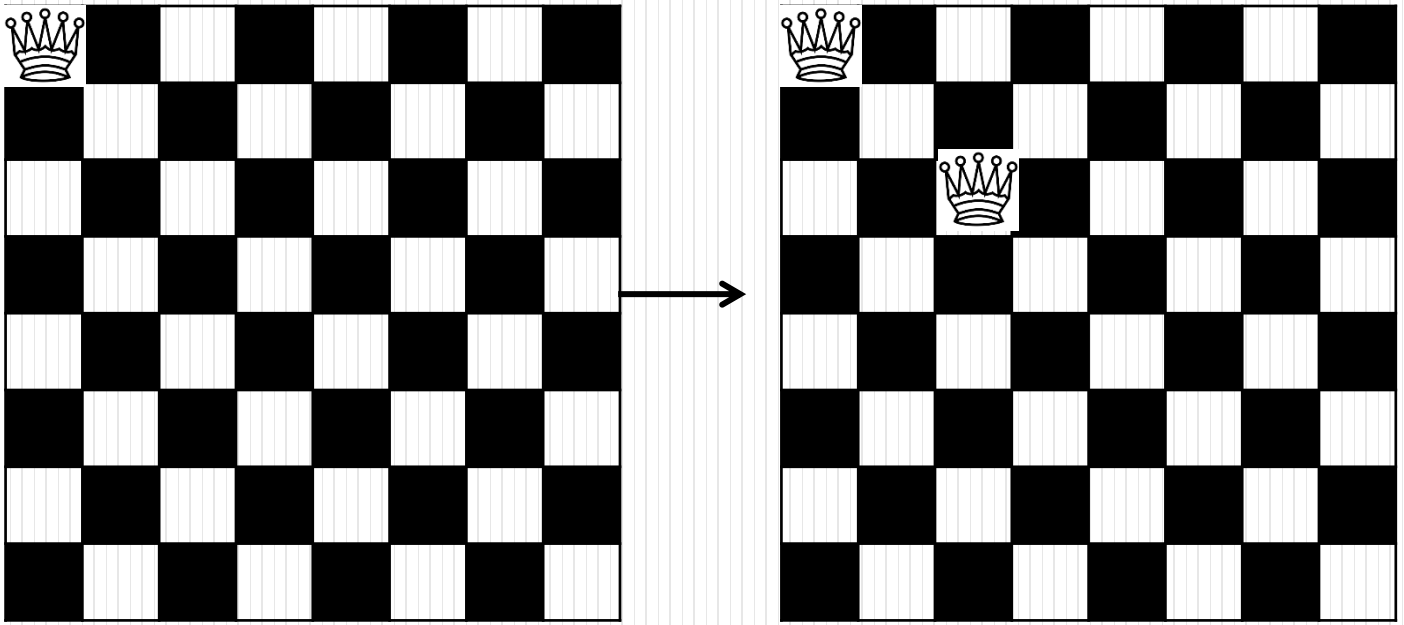
EXAMPLE: 8-QUEENS PROBLEM



Initial state

State after adding a queen
64 such states will be there

EXAMPLE: 8-QUEENS PROBLEM

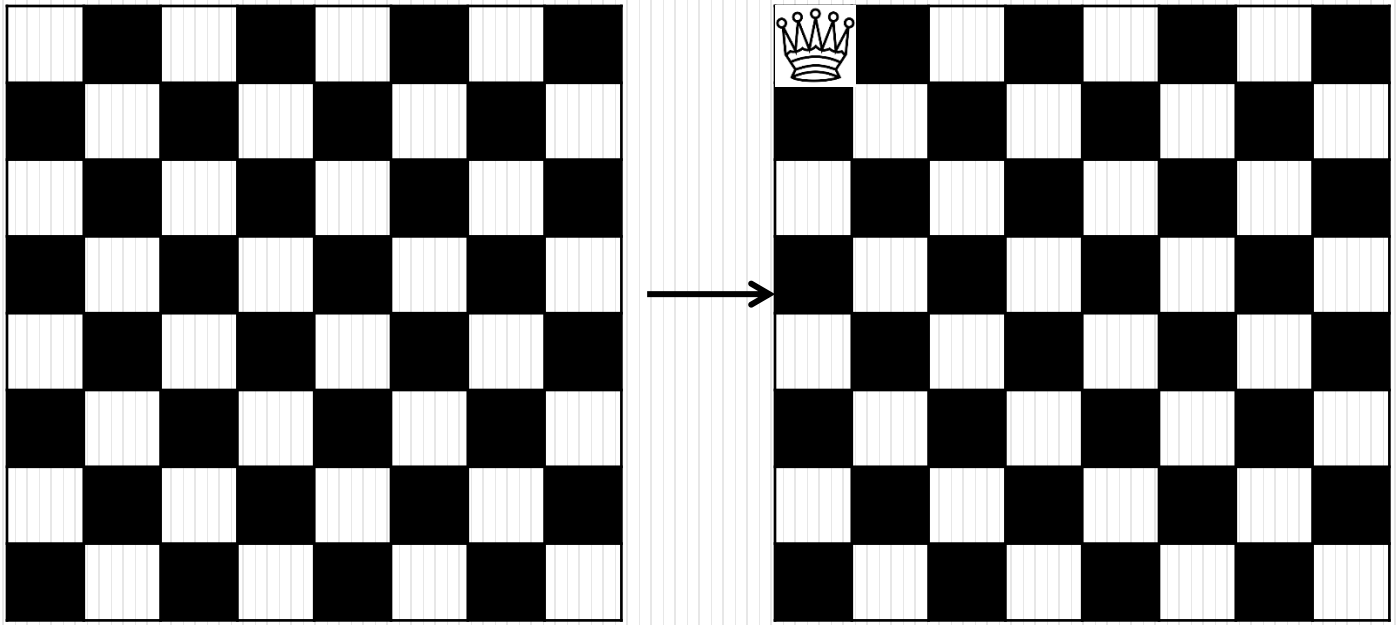


State after adding a queen
64 such states will be there

State after adding another queen
63 such states will be there

$64 \cdot 63 \cdot \ldots \cdot 57 \approx 1.8 \times 10^{14}$ configurations

EXAMPLE: 8-QUEENS PROBLEM

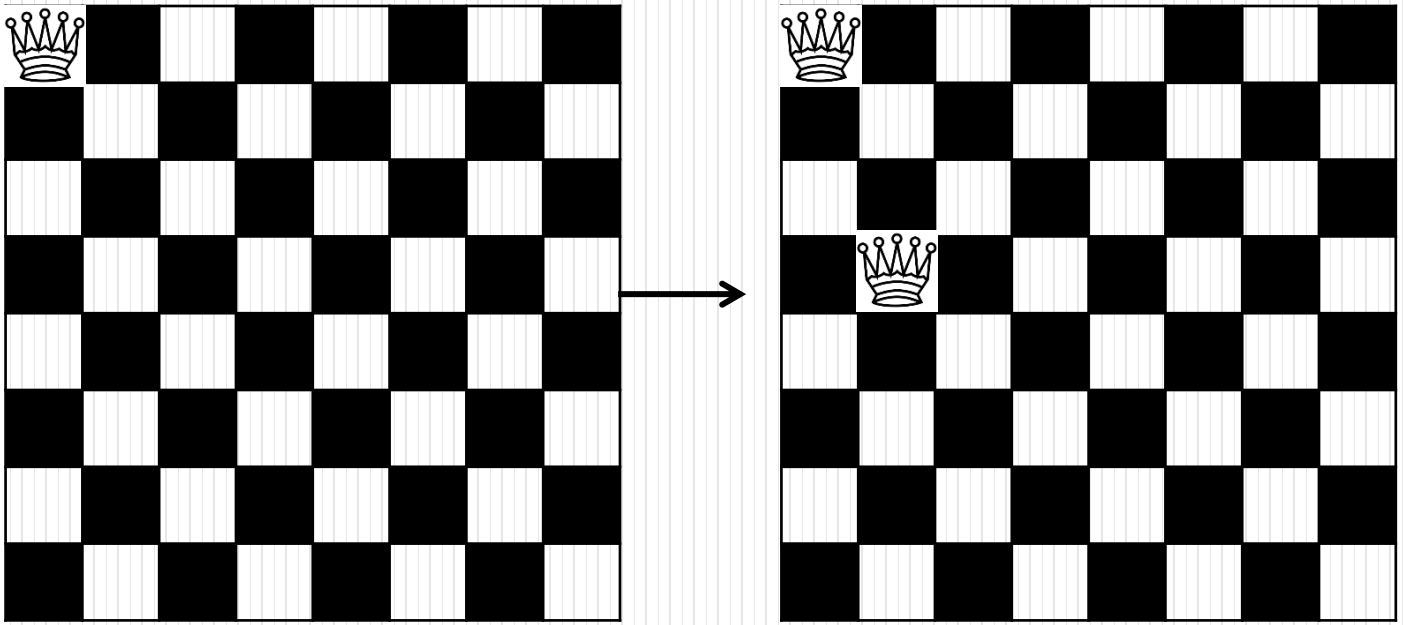


Initial state

State after adding a queen

arrangement of `n<=8` queens in leftmost n columns, 1 per column such that no queen attacks any other.

EXAMPLE: 8-QUEENS PROBLEM



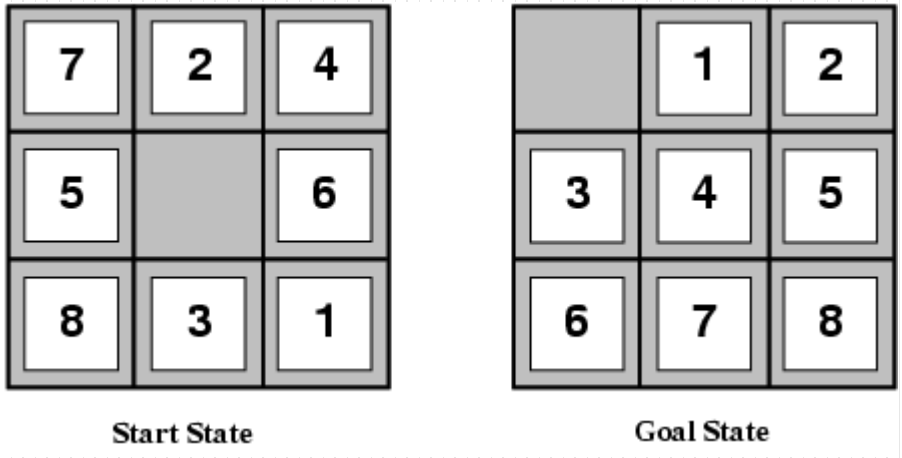
State after adding a queen

State after adding another queen

Just 2,057 configurations

EXAMPLE: THE 8-PUZZLE PROBLEM

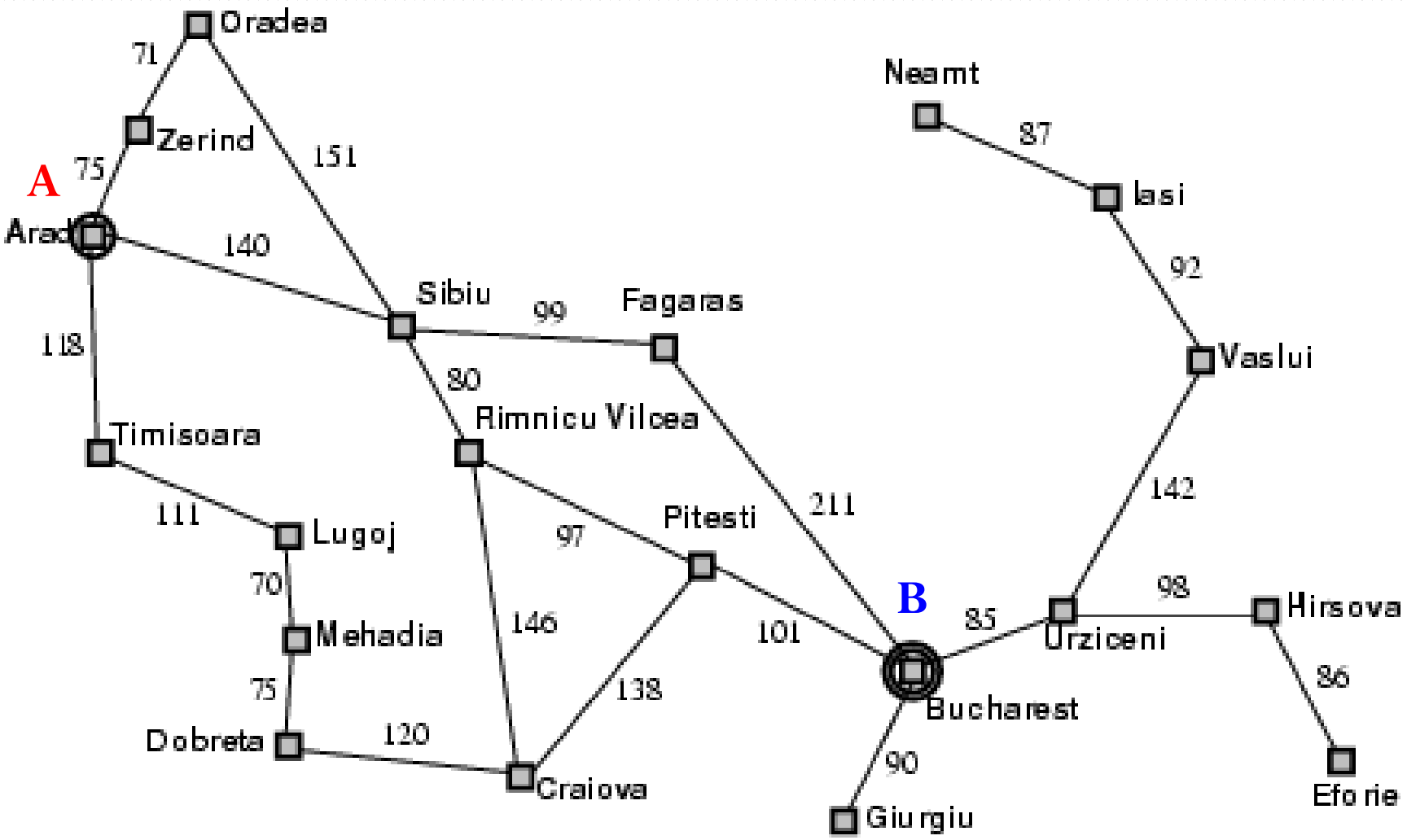
- Problem consists of a 3x3 board with eight numbered tiles and a blank space, tile adjacent to the blank space can slide into the space, the aim is to reach a specified goal such as the one shown in the figure.



EXAMPLE: THE 8-PUZZLE PROBLEM

- States
  - location of each tiles and blank
- Initial state
  - any state (as shown in figure)
- Actions
  - move blank left, right, up, down
- Transition model
  - Given a state and action, this returns the resulting state
- Goal test
  - checks if the given state is same as the goal state.
- Path cost
  - 1 per step (move)

EXAMPLE: ROUTE FINDING PROBLEM-ROMNIA

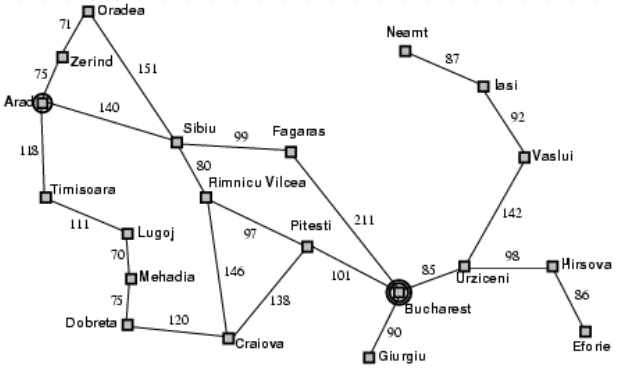


EXAMPLE: ROUTE FINDING PROBLEM-ROMNIA

- States
  - locations at which agent is in (various cities)
- Initial state
  - `In(A)` : in city Arad
- Actions
  - drive between cities

    `ACTION(IN(A)) = {Go(S), Go(T), Go(Z)}`
- Transition model

  `RESULT(In(A), Go(Z)) = In(Z)`
- Goal test
  - Check if agent is in state `{In(B)}`
- Path cost
  - Step cost as given in figure

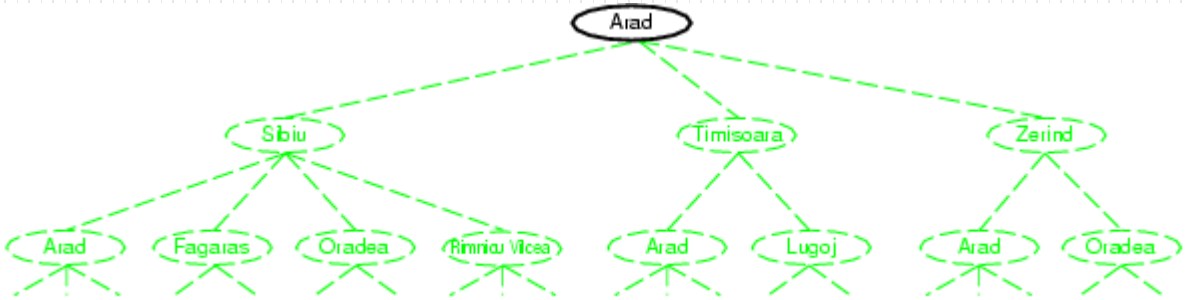


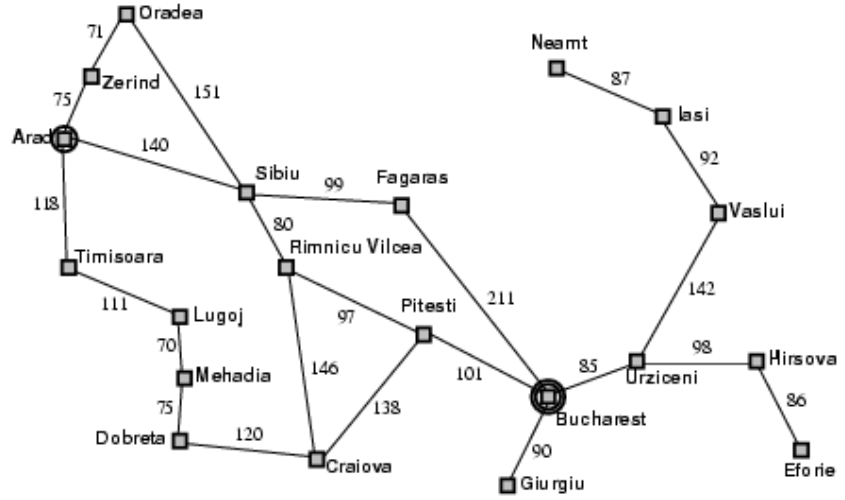
Solution: sequence of actions: in this case sequence of cities: Example: A, S, F, B.

SEARCHING

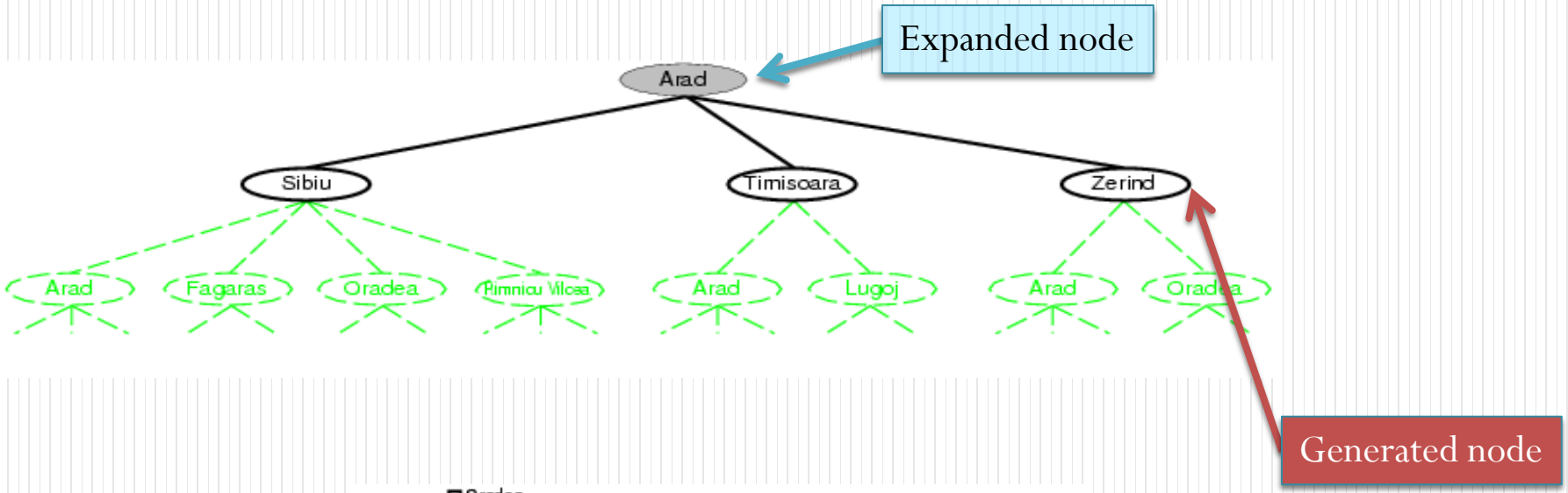
- Basic idea
  - performed by *expanding* current state and *generating* new set of states
  - every state is tested if it is a goal state before expanding it
  - the process of expanding the generated nodes continues until either a solution is found or there are no more states to expand.
- Search strategy
  - Basic idea is same various strategies vary mainly according to how they chose which state to expand next.

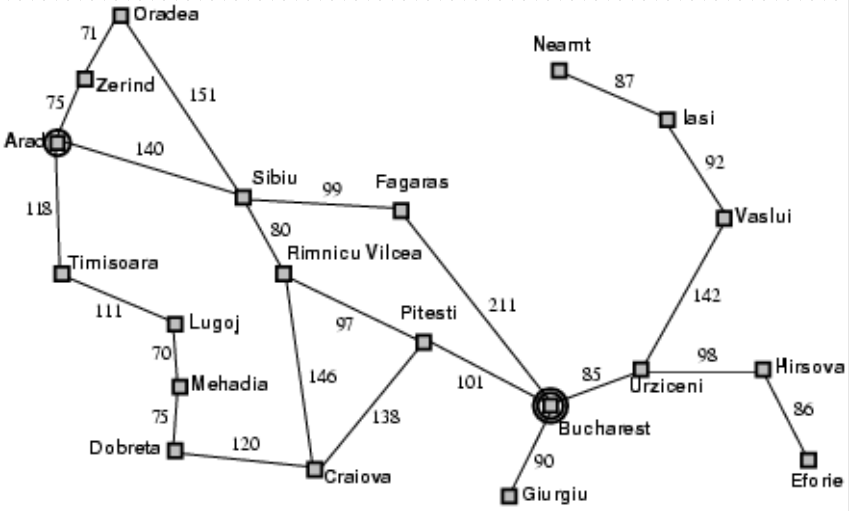
SEARCH EXAMPLE: ROMANIA



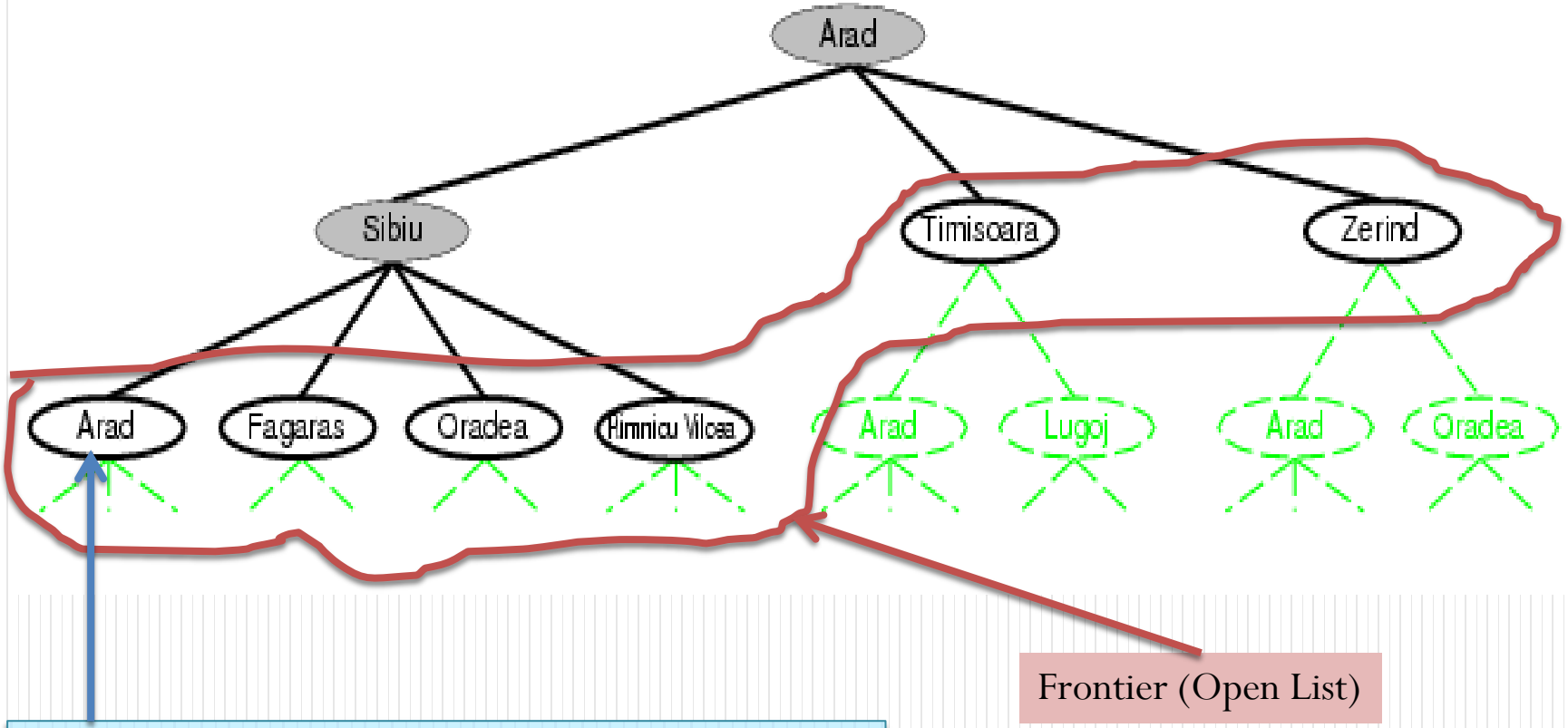


SEARCH EXAMPLE: ROMANIA





SEARCH EXAMPLE: ROMANIA



To avoid repeated states, maintain Explored Set (Closed List) that includes every expanded node.

PERFORMANCE MEASURE

- Search strategies are evaluated using the following parameters:
  - Completeness
    - Does it always find a solution when there is one?
  - Time complexity
    - How long does it take to find a solution?
  - Space complexity
    - How much memory is needed to perform the search?
  - Optimality
    - Does it always find a least cost solution?

PERFORMANCE MEASURE

- As state space is represented implicitly, Time and Space complexity is measured in terms of:
  - **b**: branching factor or maximum number of successors of any node
  - **d**: the depth of the shallowest goal node (depth of least cost solution)
  - **m**: maximum length of any path in the state space

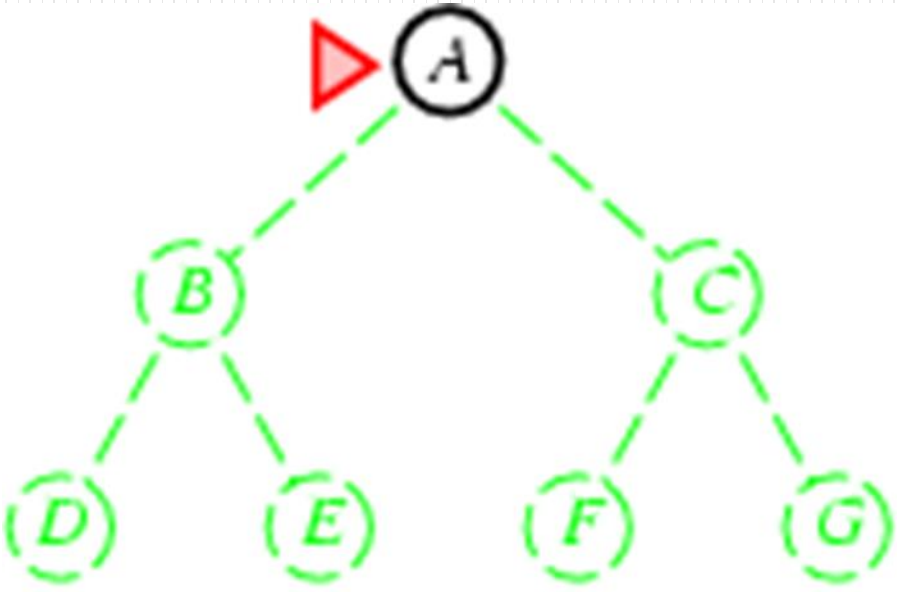
UNINFORMED SEARCH STRATEGIES

- Uninformed (blind search strategies)
  - no additional information about the states other than problem definition provided.
  - no clue whether one non-goal state is better than any other.
- Various blind strategies:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

BREADTH-FIRST SEARCH

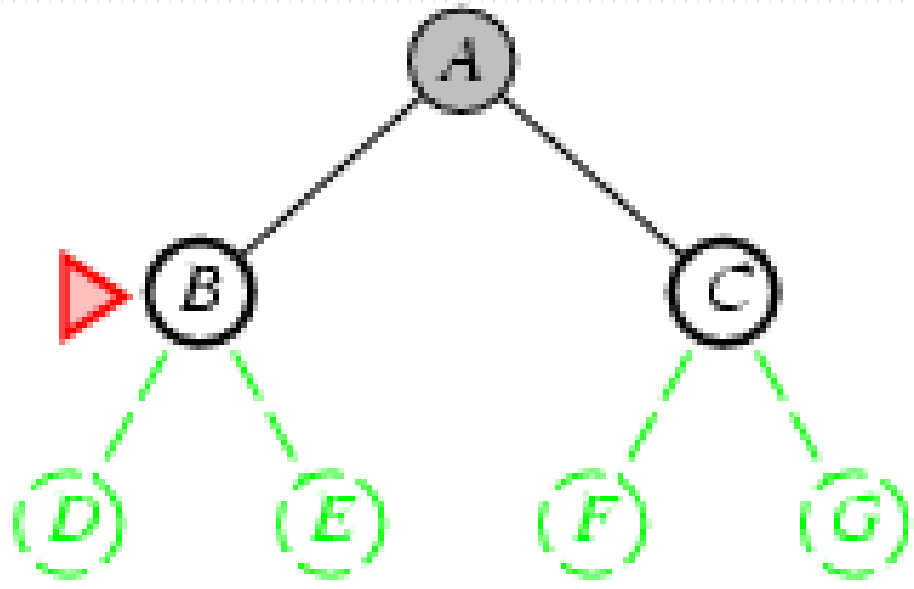
- the shallowest unexpanded node is chosen for expansion
- FIFO queue is used to store the frontier
- *goal test is applied to each node when it is generated rather then when it is selected for expansion.*

Is A a goal state?



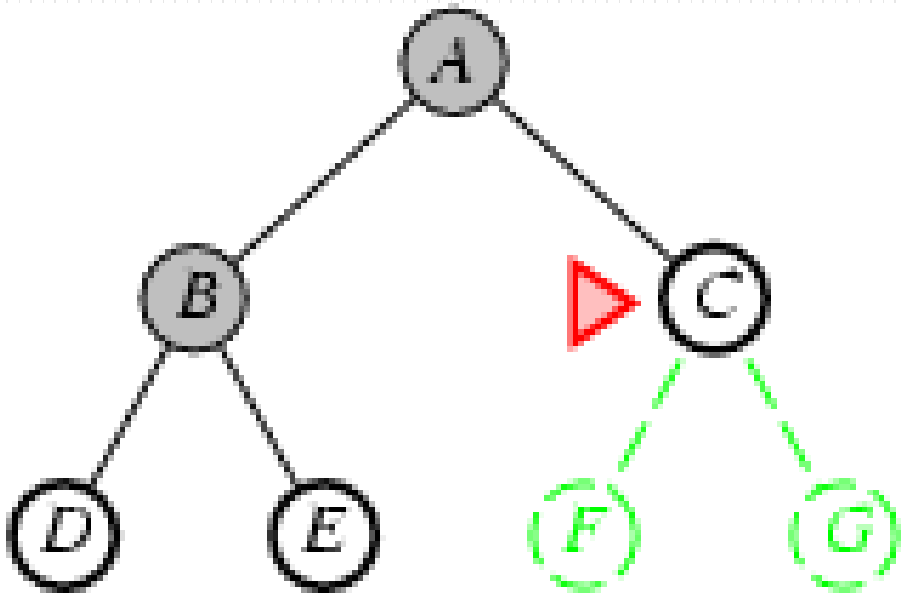
BREADTH-FIRST SEARCH

- Expand A
- Is B a goal state?
- Frontier/Open List = [B]
- Is C a goal state?
- Frontier = [B,C]



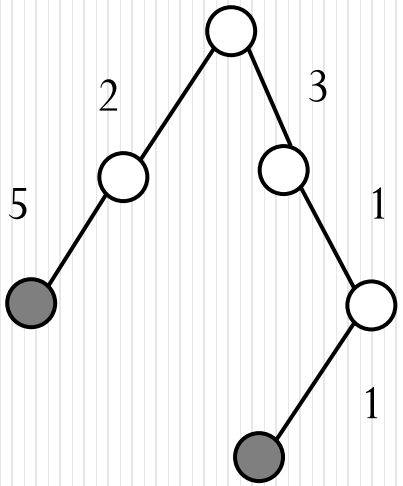
BREADTH-FIRST SEARCH

- Expand B
- Is D a goal state?
- frontier=[C,D]
- Is E a goal state?
- frontier = [C,D,E]



PROPERTIES OF BREADTH-FIRST SEARCH

- Complete?
  - Yes it always reaches goal (if *b* is finite)
- Optimal?
  - Yes (if path cost is nondecreasing funtion of the depth of node, e.g. step-cost=1).



Example: Path cost is not increasing with depth.

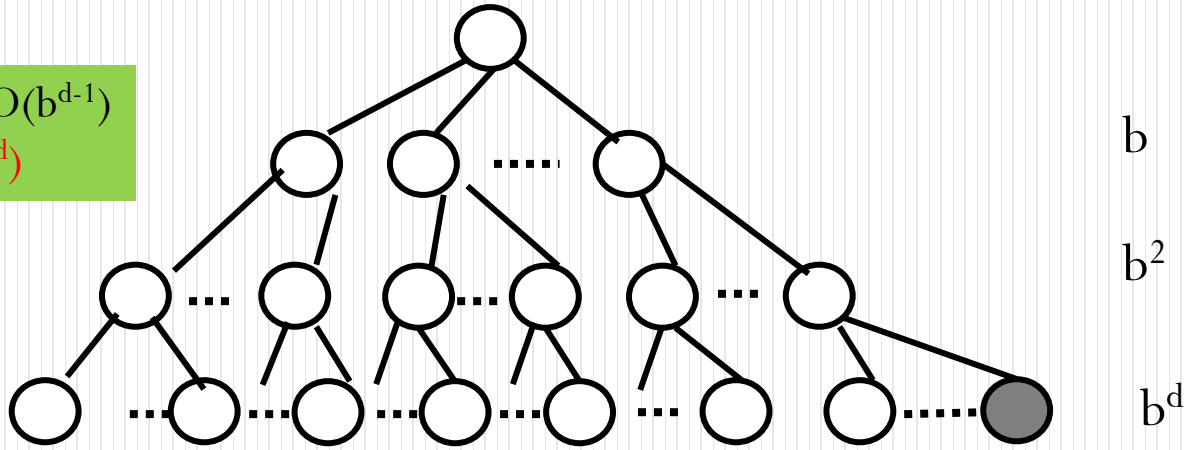
UCS

PROPERTIES OF BREADTH-FIRST SEARCH

- Time?
  - $1+b+b^2+b^3+\ldots+b^d = O(b^d)$

    (this is the number of nodes we generate)
- Space?
  - $O(b^d)$ (keeps every node in memory, expanded node as well as frontier.)

Explored set : $O(b^{d-1})$
Frontier : $O(b^d)$



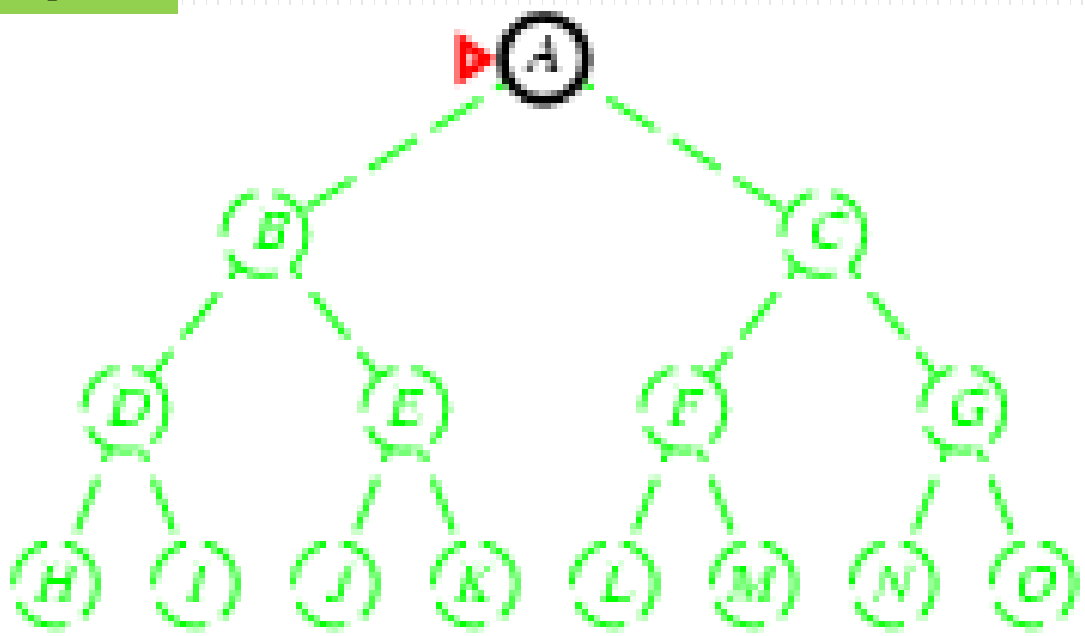
DEPTH-FIRST SEARCH

- the *deepest* unexpanded node is chosen for expansion
- Frontier is implemented using stack (LIFO).

Goal test
Expand

OPEN = [A]

CLOSED = []

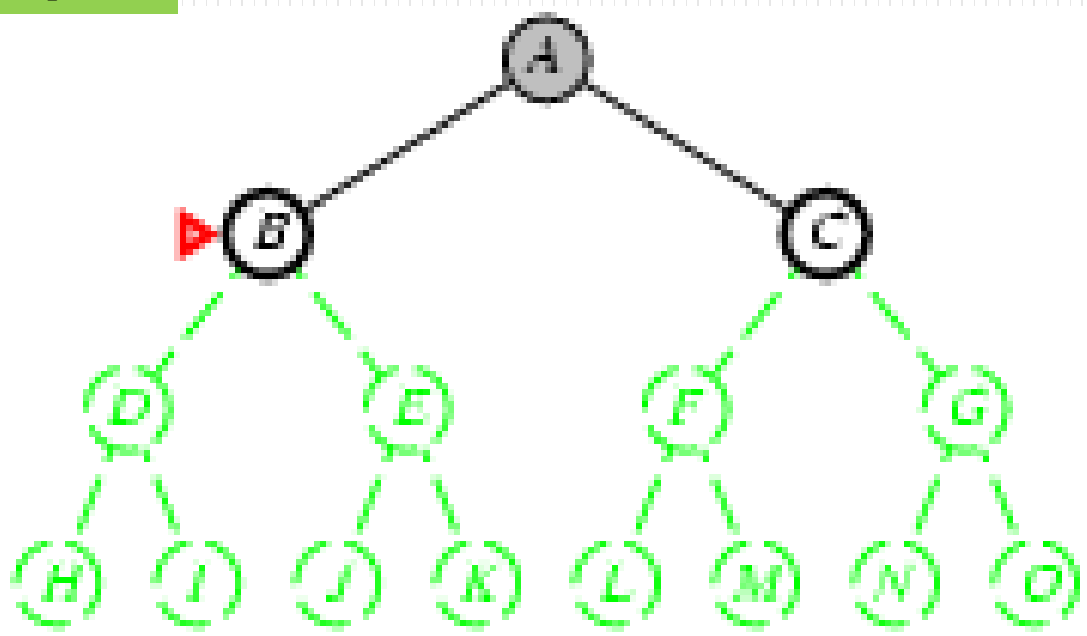


DEPTH-FIRST SEARCH

Goal test
Expand

OPEN = [C B]

CLOSED = [A]

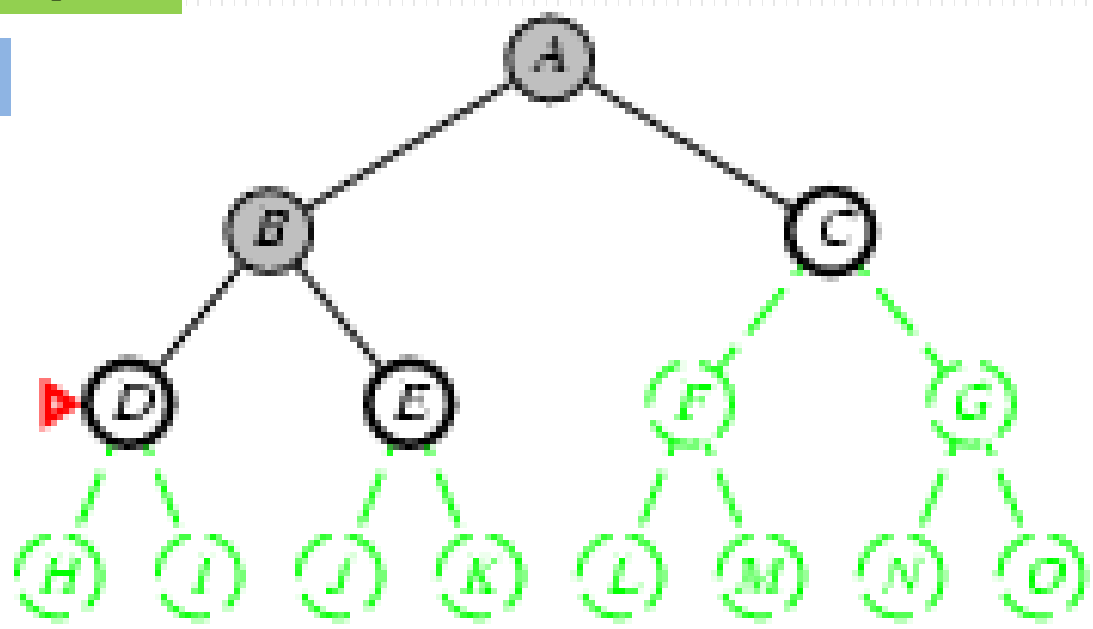


DEPTH-FIRST SEARCH

Goal test
Expand

OPEN = [C E D]

CLOSED = [A B]

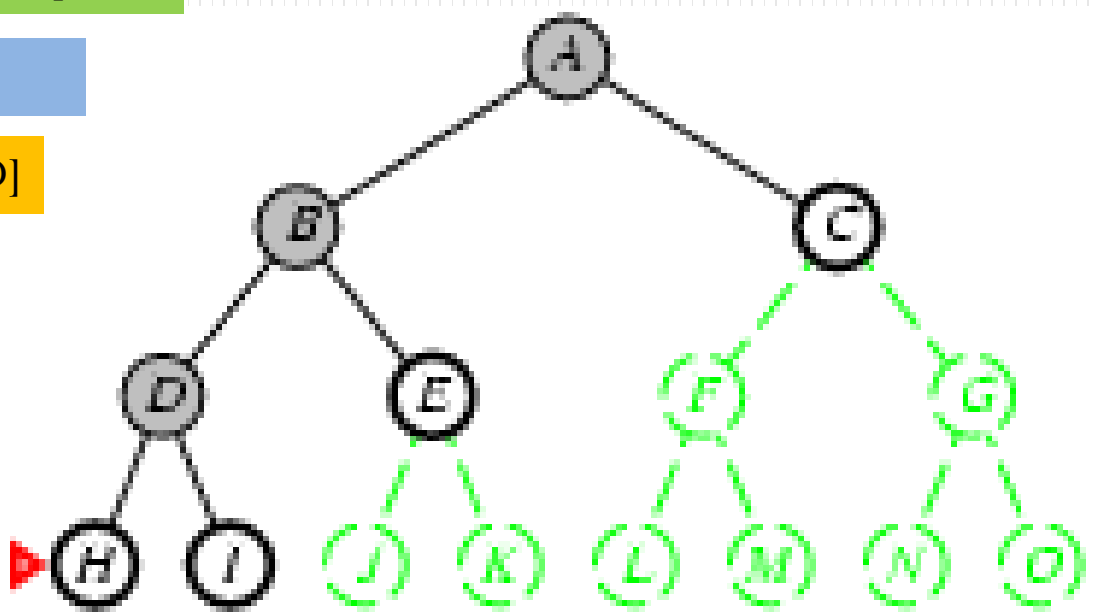


Depth-first search

Goal test
Expand

OPEN = [C E I H]

CLOSED = [A B D]



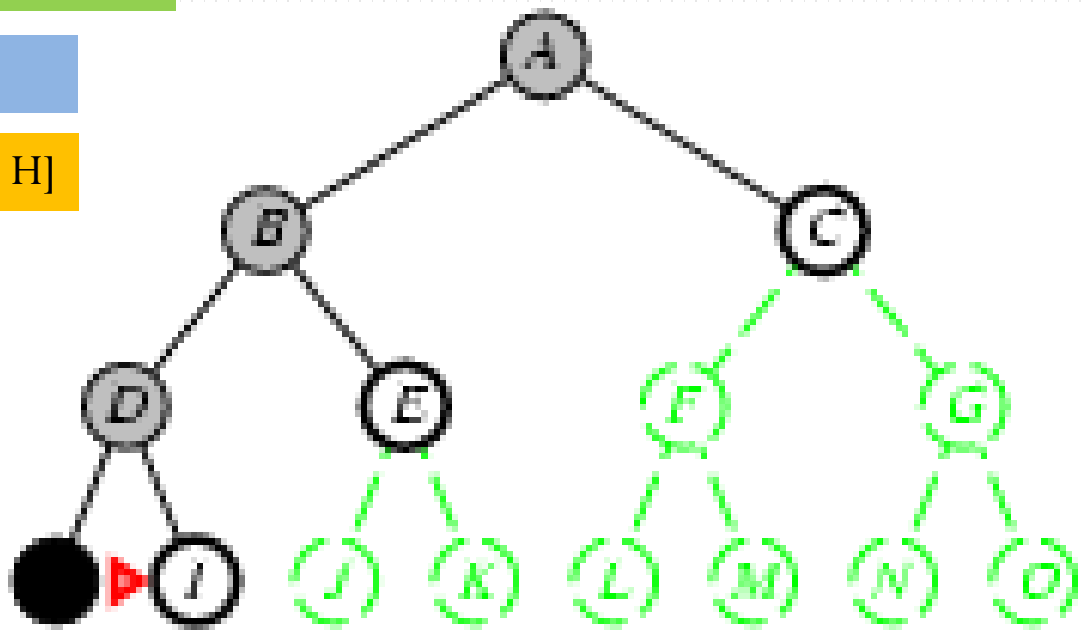
Depth-first search

Black nodes with no descendants in frontier can be removed.

Goal test
Expand

OPEN = [C E I]

CLOSED = [A B D H]

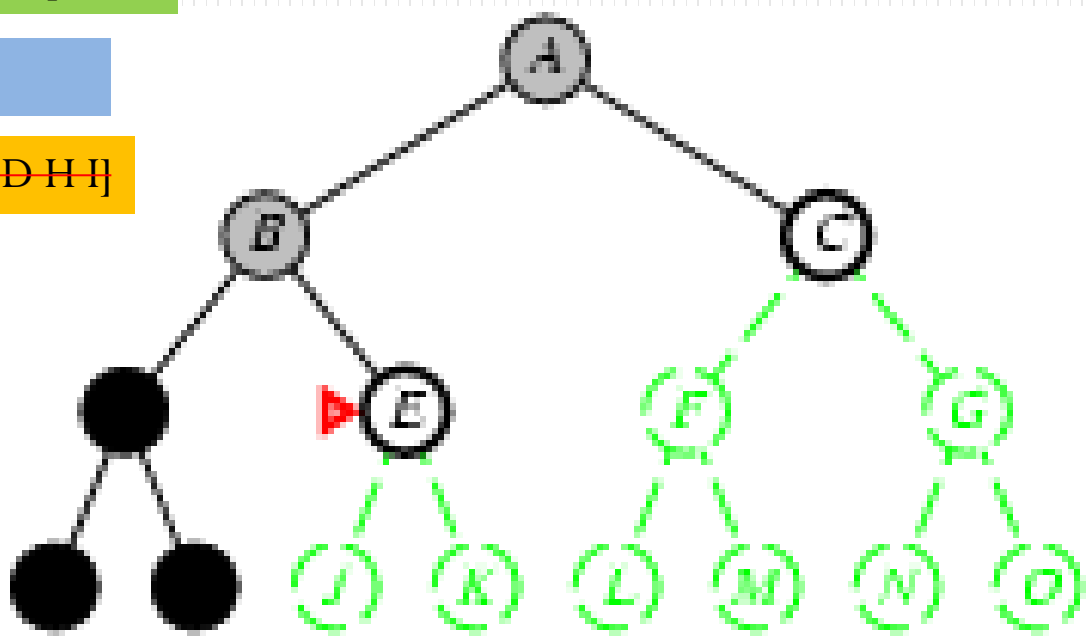


Depth-first search

Goal test
Expand

OPEN = [C E]

CLOSED = [A B ~~D H I~~]

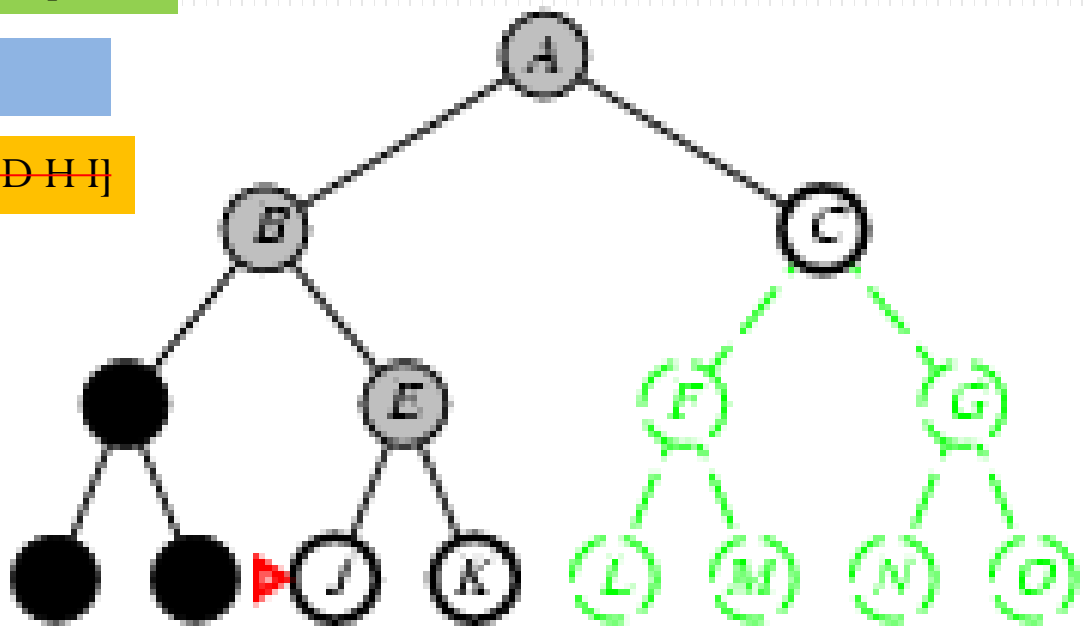


Depth-first search

Goal test
Expand

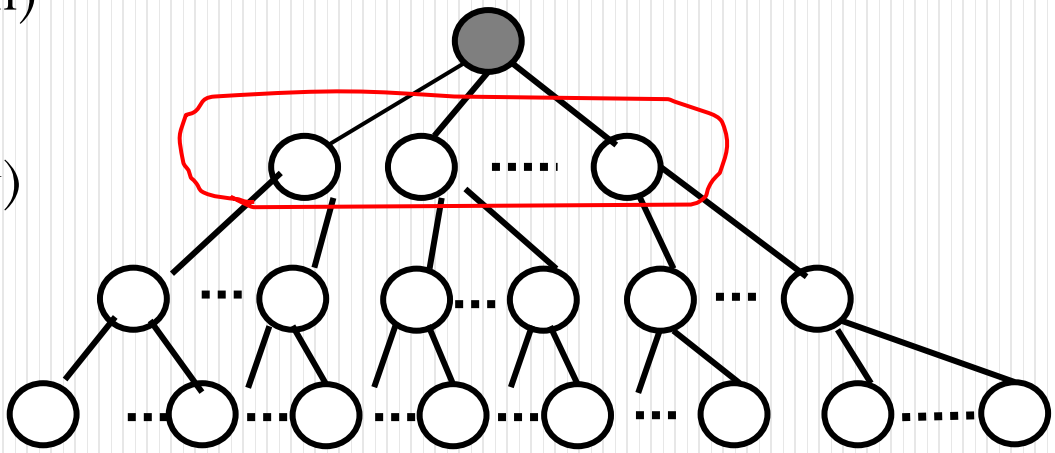
OPEN = [C K J]

CLOSED = [A B ~~D H I~~]



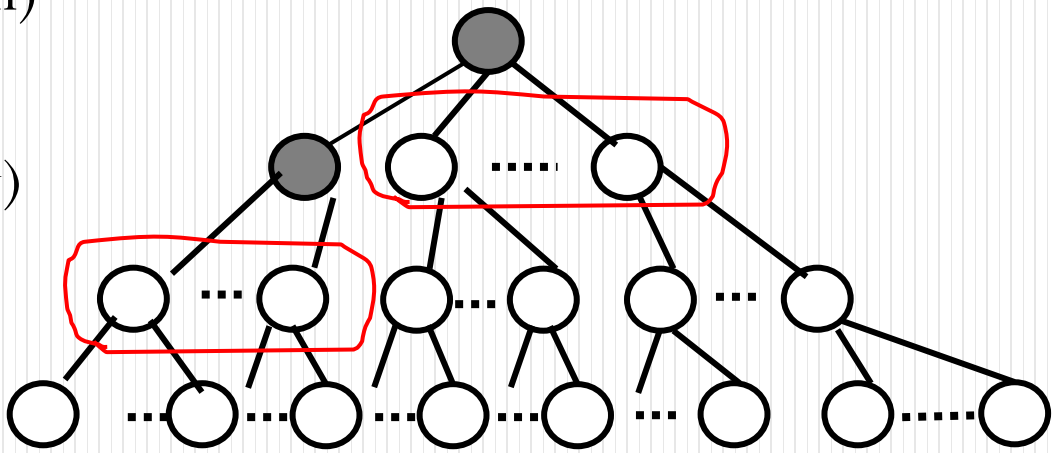
PROPERTIES OF DEPTH-FIRST SEARCH

- Complete?
  - Yes (if the state space is finite)
- Optimal?
  - No (it does not search level by level)
- Space?
  - O(bm)
- Time?
  - $O(b^m)$



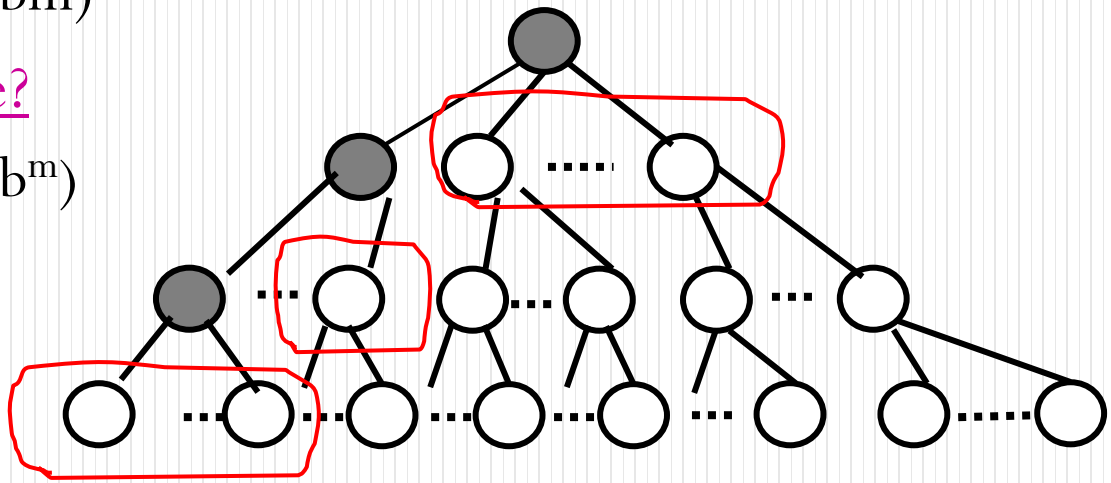
PROPERTIES OF DEPTH-FIRST SEARCH

- Complete?
  - Yes (if the state space is finite)
- Optimal?
  - No (it does not search level by level)
- Space?
  - O(bm)
- Time?
  - $O(b^m)$



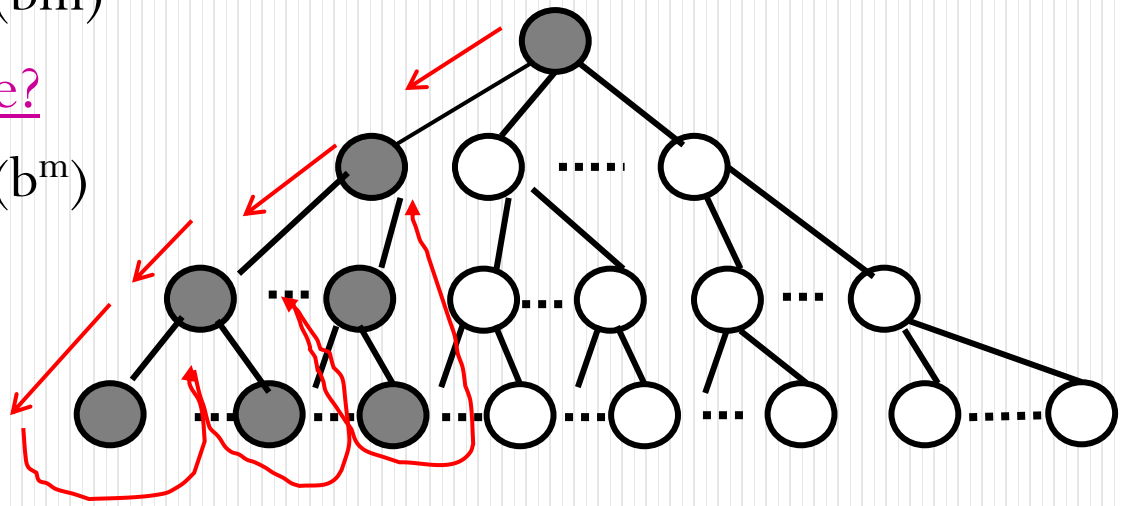
PROPERTIES OF DEPTH-FIRST SEARCH

- Complete?
  - Yes (if the state space is finite)
- Optimal?
  - No (it does not search level by level)
- Space?
  - O(bm)
- Time?
  - $O(b^m)$



PROPERTIES OF DEPTH-FIRST SEARCH

- Complete?
  - Yes (for finite state-spaces it is complete)
- Optimal?
  - No (it does not search level by level)
- Space?
  - O(bm)
- Time?
  - $O(b^m)$



DEPTH-LIMITED SEARCH

- depth-first search with an imposed limit (l) on the depth of exploration, solves the problem of infinite path
- How to determine the value of l?
- If **l<d** then it is **incomplete** and if **l > d** then it **is not optimal.**
- Space?
  - O(bl)
- Time?
  - $O(b^l)$

ITERATIVE DEEPENING SEARCH

- strategy to find the best depth limit.
- *increase limit Iteratively* – first 0, then 1, then 2, …until a goal is found (this will occur when the depth limit reaches d, the depth of the shallowest goal node).
- Combines the benefits of DFS and BFS
  - memory requirements like DFS
  - complete and optimal like BFS

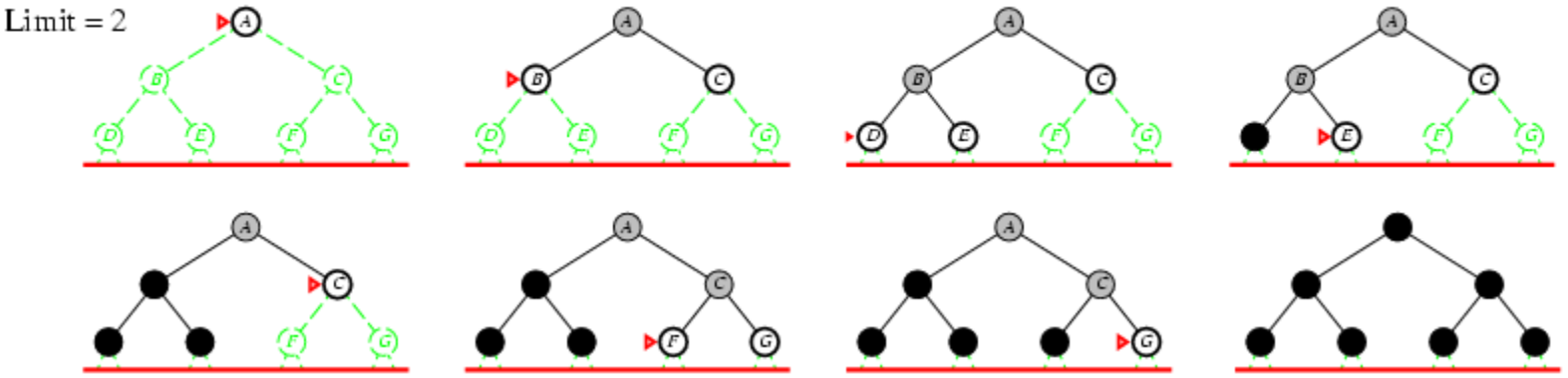
It is often used in combination with depth-first tree search that does not take into account repeated states.

ITERATIVE DEEPENING SEARCH *L*=0

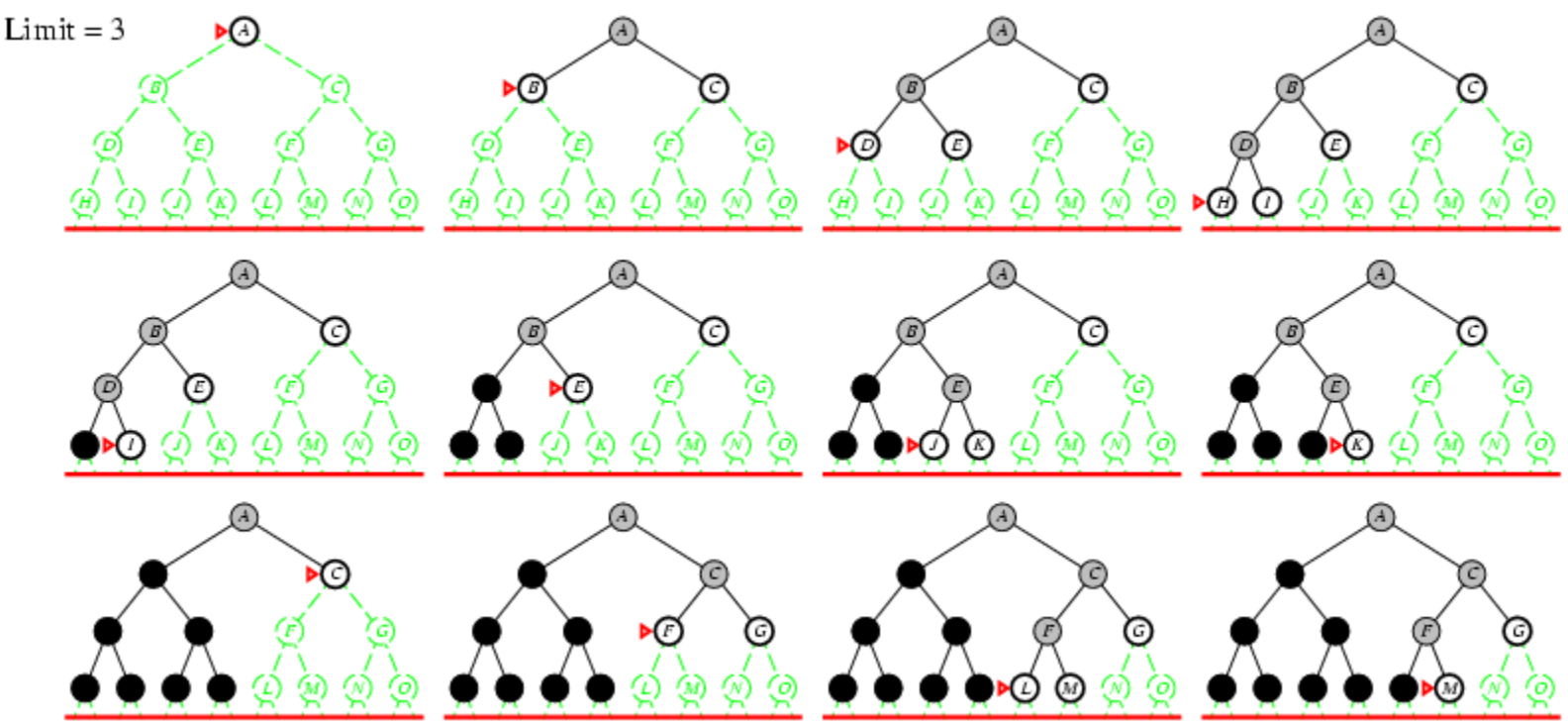
Limit = 0

ITERATIVE DEEPENING SEARCH *L*=1

ITERATIVE DEEPENING SEARCH *L*=2



ITERATIVE DEEPENING SEARCH *L*=3



ITERATIVE DEEPENING SEARCH

- Nodes are generated multiple times.
- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

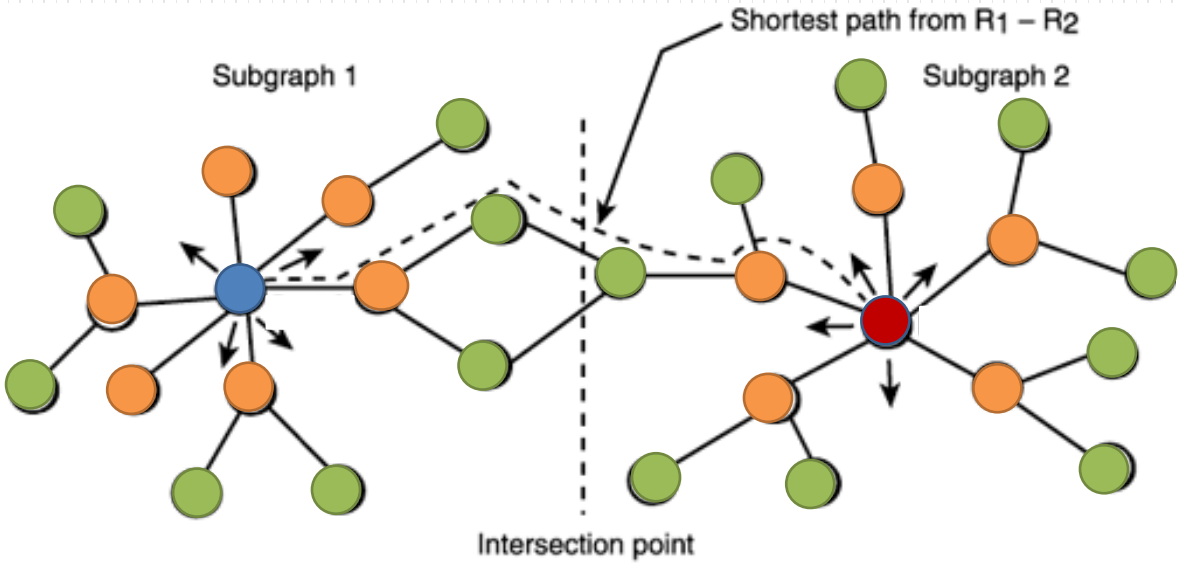
$$N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$$

PROPERTIES OF ITERATIVE DEEPENING SEARCH

- Complete? Yes
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

BIDIRECTIONAL SEARCH

- simultaneously search forward from S and backwards from G
- stop when both "meet in the middle"
- need to keep track of the intersection of both frontiers



PROPERTIES OF BIDRECTIONAL SEARCH

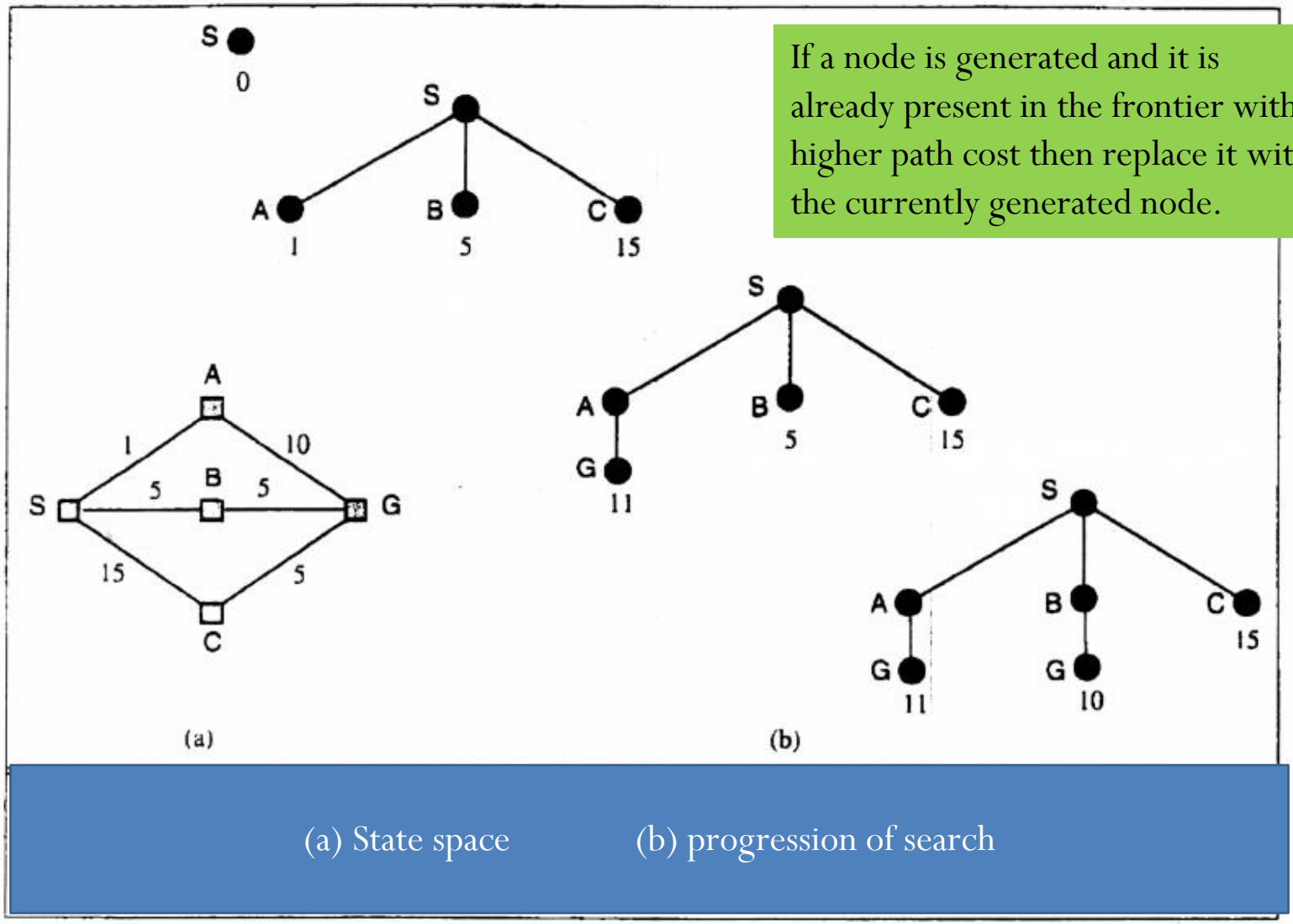
- Complete? Yes (if both directions use BFS)
- Time? $O(b^{d/2})$
- Space? $O(b^{d/2})$
- Optimal? Yes(if both direction uses BFS and if step cost = 1 or increasing function of depth)

UNIFORM COST SEARCH

- For problems that have cost associated with actions (step cost and path cost).
- Expand node with smallest path cost g(n).
- frontier stored in a priority queue ordered by g(n)
- *goal test is applied to a node when it is selected for expansion rather than when it is first generated.*

(Reason is that first goal node that is generated may be on a suboptimal path)

UNIFORM COST SEARCH



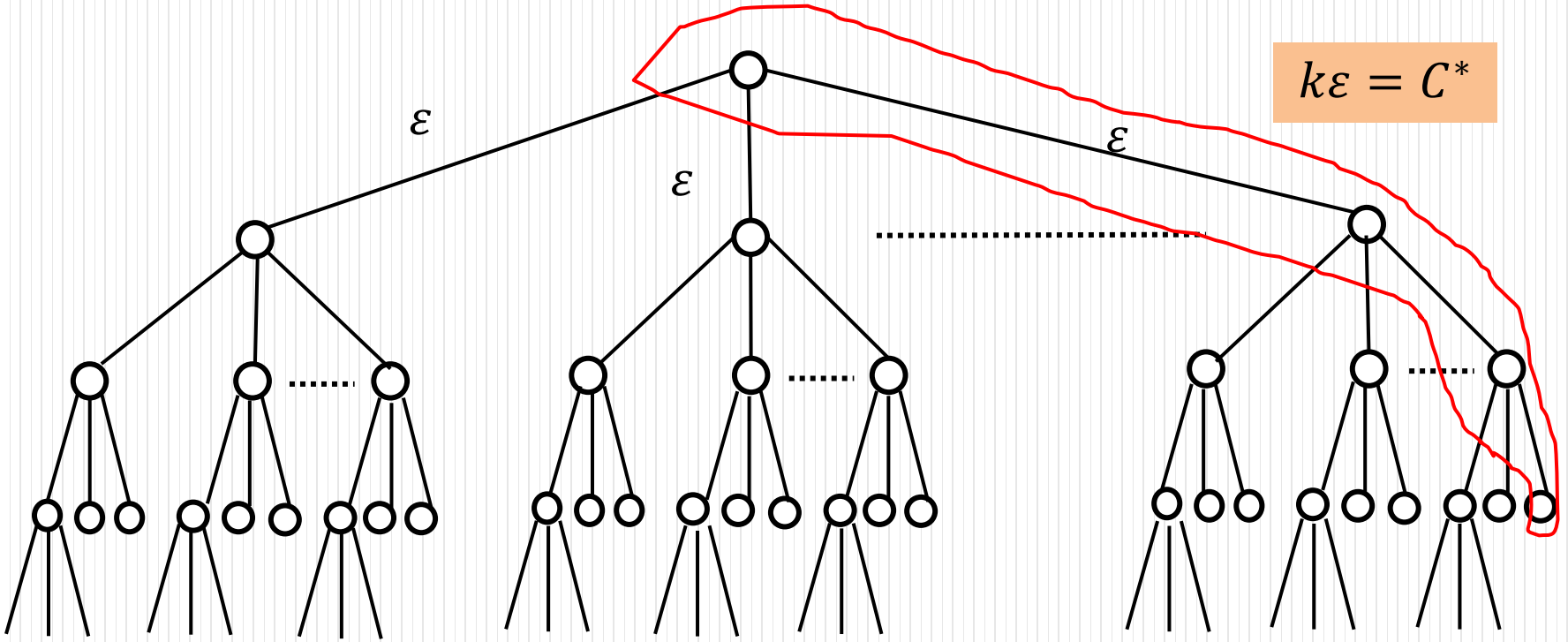
(a) State space (b) progression of search

BFS

PROPOERTIES OF UNIFORM COST SEARCH

- Complete?
  - Yes, if step cost $\geq \varepsilon > 0$
- Optimal?
  - Yes – nodes expanded in increasing order of $g(n)$
- Time?
  - let C* be the cost of optimal solution and every action costs at least ε
  - number of nodes with $g \leq$ cost of optimal solution
  - $O(b^{1+C^*/\varepsilon})$
- Space?
  - $O(b^{1+C^*/\varepsilon})$

PROPOERTIES OF UNIFORM COST SEARCH



worst case: # nodes generated $= 1 + b^{\varepsilon/\varepsilon} + b^{2\varepsilon/\varepsilon} + \ldots + b^{\frac{k\varepsilon}{\varepsilon}} + b^{1+\frac{k\varepsilon}{\varepsilon}} - b$

What did we learn?

- What is a state space (search space)?
- What is a search tree?
- Problem Formulation : How to define problems and solutions in terms of initial state, actions, goal test, transition model, and path cost?
- Various Uninformed search strategies: BFS, DFS, Depth-limited, IDDFS, Bidirectional search, Uniform cost search.
- Performance of uninformed search strategies
- The problem of repeated states and how to handle it.