Artificial Intelligence & Soft Computing

CSC 703

Subject In-charge

Safa Hamdare

Assistant Professor

Room No. 407

email: safahamdare@sfit.ac.in

Chapter 2
Problem Solving

Based on CO2: Choose an appropriate problem solving method for an agent to find a sequence of actions to reach the goal state.

Outline of Problem Solving

- **Solving Problems by Searching**
  - Problem Solving Agent
  - Formulating Problems
- **Search Strategies**
  1. **Uninformed Search Methods**
     - Breadth First Search
     - Depth First Search
     - Depth Limited Search
     - Uniform Cost Search
     - Iterative Search methods (Depth First iterative Deepening)
  2. **Informed Search Methods-** A*

Outline of Problem Solving

- **Local Search Problems**
  - Hill-climbing
  - Simulated annealing
  - Genetic Algorithms

Problem Solving Agent

Problem Solving Agent

- Four general steps in problem solving:
  - Goal formulation
    - ✓ What are the successful world states
  - Problem formulation
    - ✓ What actions and states to consider to give the goal
  - Search
    - ✓ Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
  - Execute
    - ✓ Give the solution perform the actions.

Problem Solving Agent

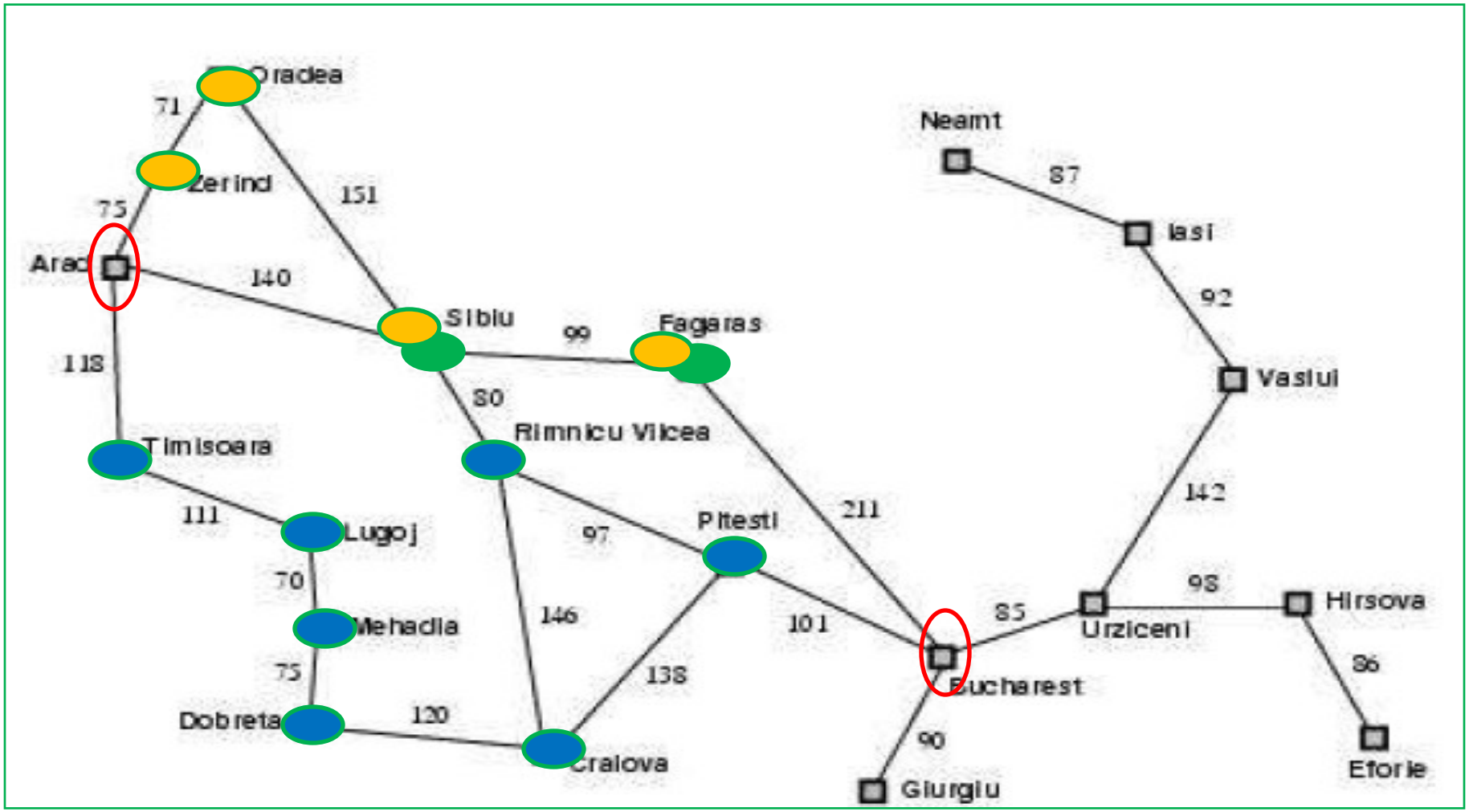
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) return an action
    static: seq, an action sequence
            state, some description of the current world state
            goal, a goal
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then
            goal ← FORMULATE-GOAL(state)
            problem ← FORMULATE-PROBLEM(state,goal)
            seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. Then it formulates next goal

Problem Solving Agent- (4 Environment Types)

- The agent design assumes that the environment is **Static,** because formulating and solving the problem is done without paying attention to any changes that might be occurring in the environment.
- The agent design also assumes that the initial state is known; knowing it is easiest if the environment is **observable.**
- The idea of enumerating : "alternative courses of action" assumes that the environment can be viewed as **discrete.**
- Finally, and most importantly, the agent design assumes that the environment is **deterministic.**

Example : Romania



Example : Romania

- On holiday in Romania; currently in Arad → Current State
  - Flight leaves tomorrow from Bucharest
- Formulate goal
  - Be in Bucharest → Goal
- Formulate problem → Action , States
  - States: various cities
  - Actions: drive between cities
- Find solution → Takes problem as input & returns a solution in the form of action sequence
  - Sequence of cities; e.g. Arad, Sibiu, Fagaras, Bucharest, …

Problem Formulation

- The initial state that the agent starts in.

  E.g. In(Arad)
- Formulation uses a Successor Function: **S(X)=Set of Action-State pair.**

  E.g. S(Arad)={In(Arad)} -> {Go(sibiu),In(sibiu)}
- State Space is Set of all states reachable from the initial state.
  - Given as: initial state +successor function=State Space
  - S(Arad)={In(Arad)} -> {Go(sibiu),In(sibiu)}
  - S(Arad)={In(Arad)} -> {Go(Ti misoara),In(Ti misoara)}
  - S(Arad)={In(Arad)} -> {Go(Zerind),In(Zerind)}
- A Path in the state space is a sequence of states connected by a sequence of actions.
  - **S(Arad)**=**{**In(Arad)} -> {Go(sibiu),In(sibiu)},

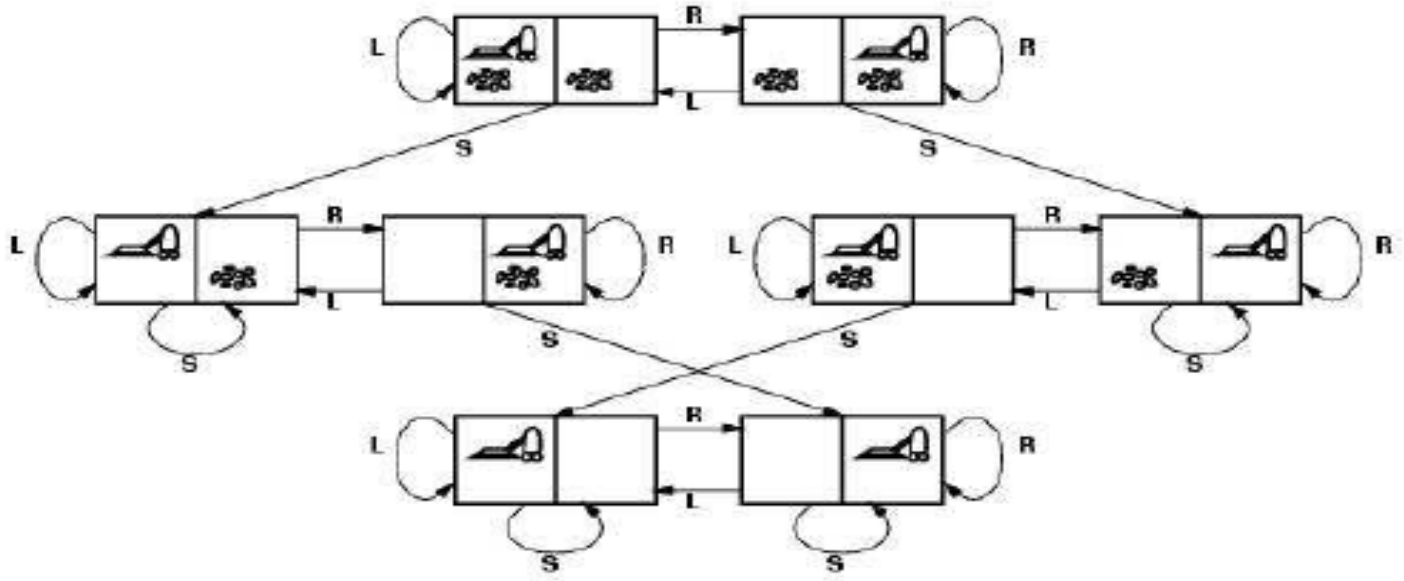
    {In(sibiu)} -> {Go(Fagaras),) In(Fagaras)}
- A Solution is a sequence of actions from initial to goal state. Optimal Solution has the lowest path cost.

PROBLEM FORMULATION

Problem in search space: A problem has four components initial state, goal test, set of actions, path cost.

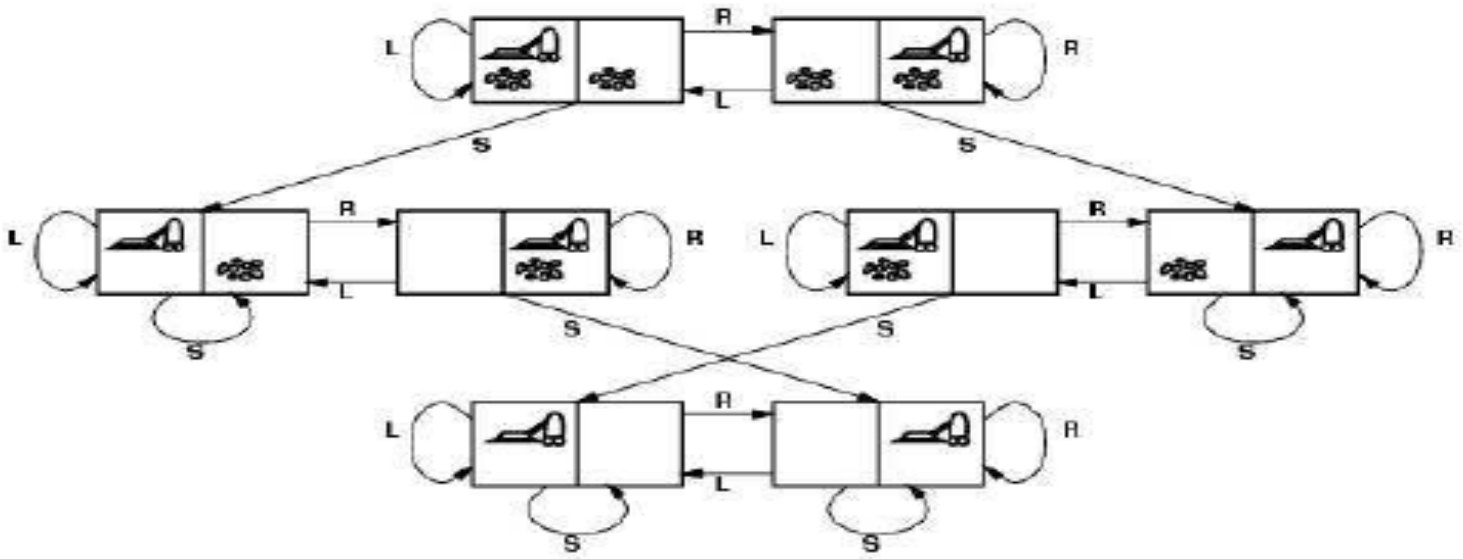
What are the components of problem formulation? Hence formulate for vacuum cleaner and 8 puzzle game. DEC 2010. (10 marks)

Example 1: Vacuum World



- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

Example 1: Vacuum World

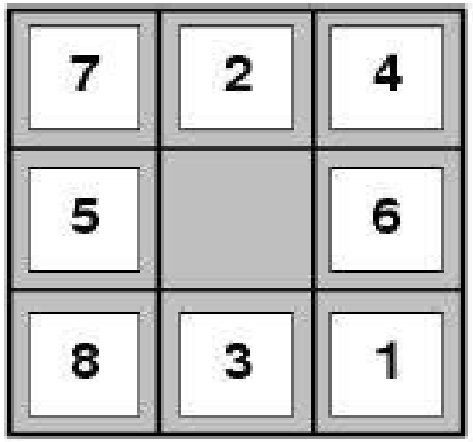


- States?? two locations with or without dirt: 2 x 2^2=8 states.
- Initial state?? Any state can be initial
- Actions?? {*Left*, *Right*, *Suck*}
- Goal test?? Check whether squares are clean.
- Path cost?? Number of actions to reach goal.

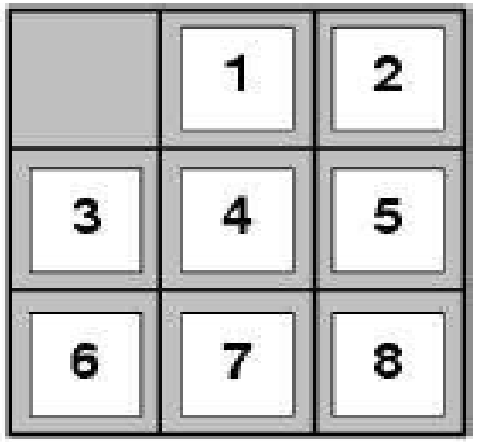
Suppose if Each step costs 1, so the path cost is the no. of steps in the path

Example 2: 8-Puzzle

An 8 Puzzle consists of 3x3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into space. The object is to reach specified goal state.



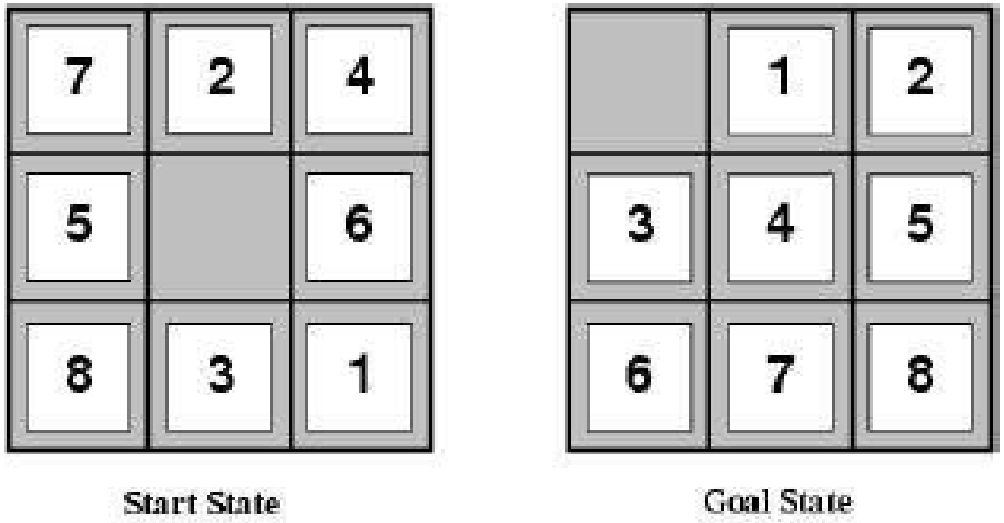
Start State



Goal State

- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

Example 2: 8-Puzzle



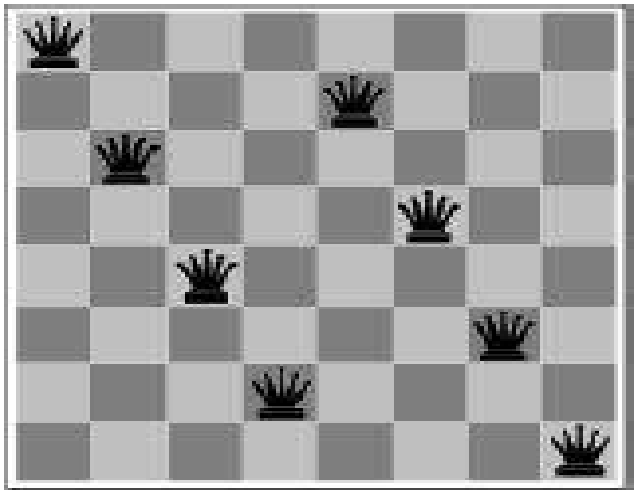
Location of each of the eight tiles and blank is one of nine squares

- States?? Integer location of each tile
- Initial state?? Any state can be initial
- Actions?? {*Left*, *Right*, *Up*, *Down*}
- Goal test?? Check whether goal configuration is reached
- Path cost?? Number of actions to reach goal

Suppose if Each step costs 1, so the path cost is the no. of steps in the path

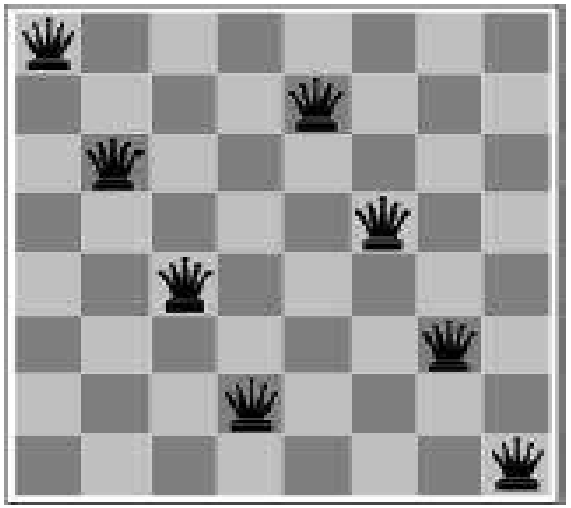
Example 3: 8 Queens Problem

8 Queens belongs to the family of Sliding-block puzzle. The goal of 8 queen is to place eight queens on a chessboard such that no queen attacks any other.(A queen attacks any piece in the same row, column or diagonal)



- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

Example 3: 8 Queens Problem



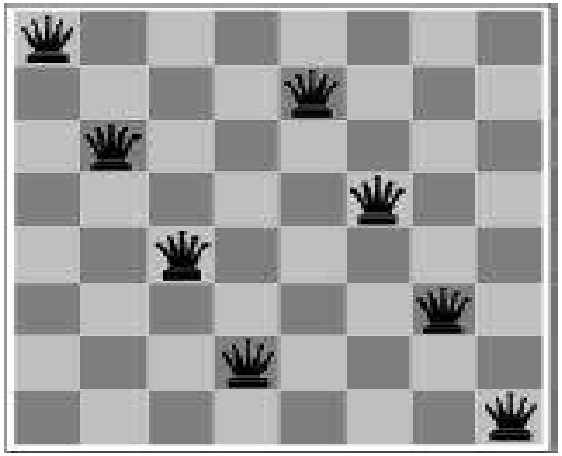
Incremental involves operators that augment the state description, starting with an empty state: each action adds a queen to the state.

Incremental formulation vs. complete-state formulation

- States??
- Initial state??
- Actions??
- Goal test??
- Path cost??

Starts with all 8 queens on the board and moves them around.

Example 3: 8 Queens Problem



Incremental involves operators that augment the state description, starting with an empty state: each action adds a queen to the state.

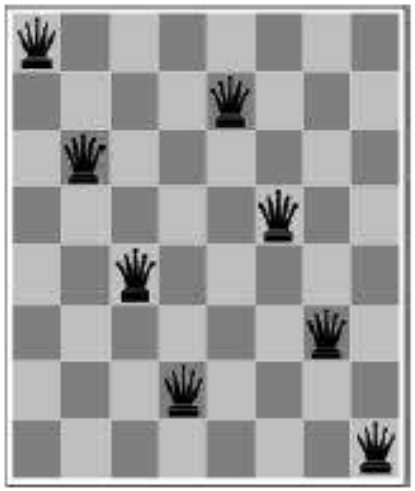
Incremental formulation

- States?? Any arrangement of 0 to 8 queens on the board
- Initial state?? No queens
- Actions?? Add queen in empty square
- Goal test?? 8 queens on board and none attacked
- Path cost?? None

Path cost is of no interest because only final state counts

3×10^{14} possible sequences to investigate

Example 3: 8 Queens Problem



Incremental formulation (alternative)

- States?? n ($0 \leq n \leq 8$) queens on the board, one per column in the n leftmost columns with no queen attacking another.
- Actions?? Add queen in leftmost empty column such that is not attacking other queens

2057 possible sequences to investigate; Yet makes no difference when n=100

Take a Quiz:

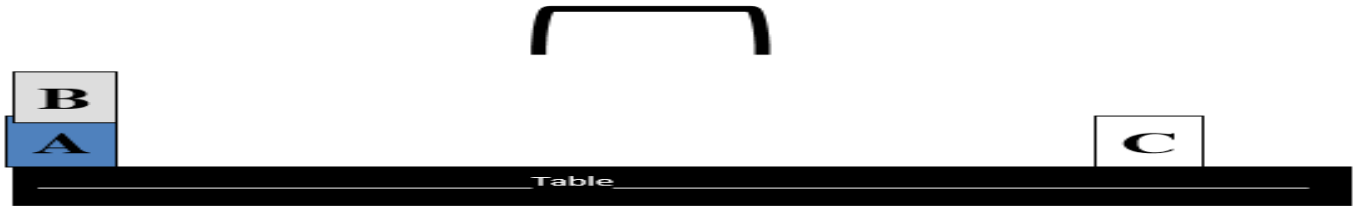
- https://docs.google.com/forms/d/e/1FAIpQLSdZnhzbRK6Hmy4-vT1oidA8rQe_CLJiUP6JHFudhyxvvREX0Q/viewform

May 2016

- Design a Planning Agent for a Block World Problem. Assume suitable initial State and Final State for the Problem.

Example 4: Block World Problem

- The blocks world is a micro-world that consists of a table, a set of blocks and a robot hand.
- **Sates:** Any configuration of Alphabet A,B,C
- **Initial State:** Any configuration like below



- **Goal State:** Blocks are one above the other as per alphabetical order.

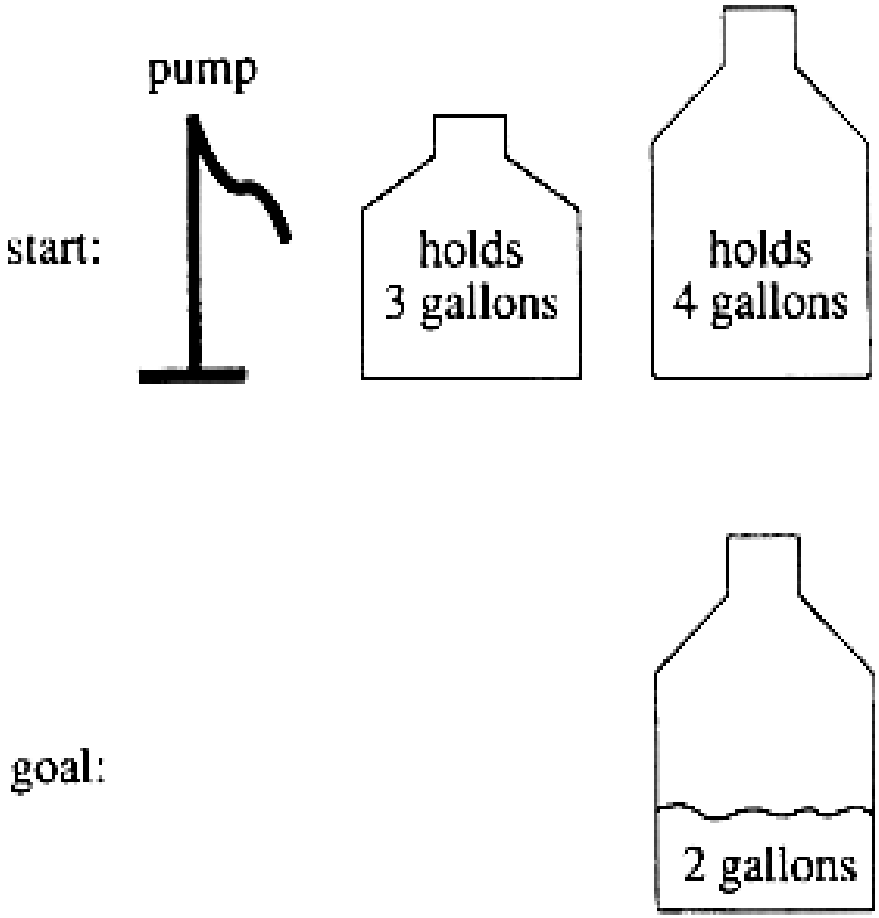
- **Action:** Move Block from one position to other by using robot hand.
- **Goal Test:** Check whether the goal position is reached or not
- **Path cost:** Number of action to reach goal state.

Dec 2016 (10 Marks)

- Given a full 4 gallon Jug and an empty 3 Gallon Jug, the goal is to fill the 4 Gallon Jug with Exactly 2 Gallons of Water. Give State Space Representation.

Example 5: A Water Jug Problem

- You have a 4-gallon and a 3-gallon water jug
- You have a faucet with an unlimited amount of water
- You need to get exactly 2 gallons in 4-gallon jug



Example 5: A Water Jug Problem

- Puzzle Solving as search

- State representation: **(x, y)**
  - x: Contents of four gallon
  - y: Contents of three gallon
- Start state: **(0, 0)**
- Goal state **(2, n)**
- Operators
  - Fill 3-gallon from faucet, fill 4-gallon from faucet
  - Fill 3-gallon from 4-gallon , fill 4-gallon from 3-gallon
  - Empty 3-gallon into 4-gallon, empty 4-gallon into 3-gallon
  - Dump 3-gallon down drain, dump 4-gallon down drain

Example 5: A Water Jug Problem

- Production Rules for water jug Problem

1 $(x,y) \rightarrow (4,y)$ if $x < 4$	Fill the 4-gallon jug
2 $(x,y) \rightarrow (x,3)$ if $y < 3$	Fill the 3-gallon jug
3 $(x,y) \rightarrow (x - d,y)$ if $x > 0$	Pour some water out of the 4-gallon jug
4 $(x,y) \rightarrow (x,y - d)$ if $x > 0$	Pour some water out of the 3-gallon jug
5 $(x,y) \rightarrow (0,y)$ if $x > 0$	Empty the 4-gallon jug on the ground
6 $(x,y) \rightarrow (x,0)$ if $y > 0$	Empty the 3-gallon jug on the ground
7 $(x,y) \rightarrow (4,y - (4 - x))$ if $x + y \geq 4$ and $y > 0$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full

Example 5: A Water Jug Problem

- Production Rules for water jug Problem

8 $(x,y) \rightarrow (x - (3 - y),3)$ if $x + y \geq 3$ and $x > 0$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9 $(x,y) \rightarrow (x + y, 0)$ if $x + y \leq 4$ and $y > 0$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10 $(x,y) \rightarrow (0, x + y)$ if $x + y \leq 3$ and $x > 0$	Pour all the water from the 4-gallon jug into the 3-gallon jug

Example 5: A Water Jug Problem

- One Solution for water jug Problem

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5
0	2	9
2	0	

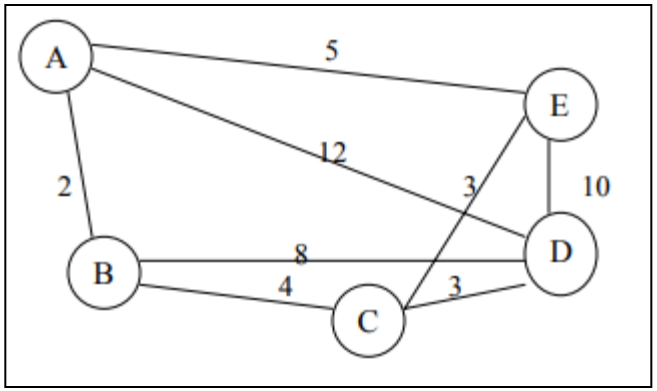
Dec 2016 (6 Marks)

- Formulate Problem for Travelling Salesman Problem (TSP). There is a map involving N cities connected by roads. The aim is to find the shortest tour that starts from a city, Visits all cities exactly once and comes back to the starting city.

Example 6: The Travelling Salesman Problem

The Travelling Salesman Problem- *Is a touring problem in which each city must be visited exactly once. The aim to find the shortest tour.*

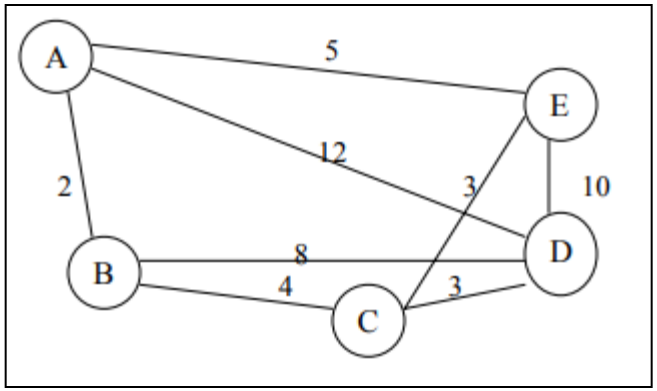
"A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the shortest possible route the salesman should follow"



Example 6: The Travelling Salesman Problem

Nearest neighbour heuristic:

1. Arbitrarily Select a starting city.
2. To Select the next city, look at all the cities not yet visited and select the one closest to the current city. Go to that city.
3. Repeat step 2 until all cities have been visited.



TSP Solution

Consider the following set of cities:

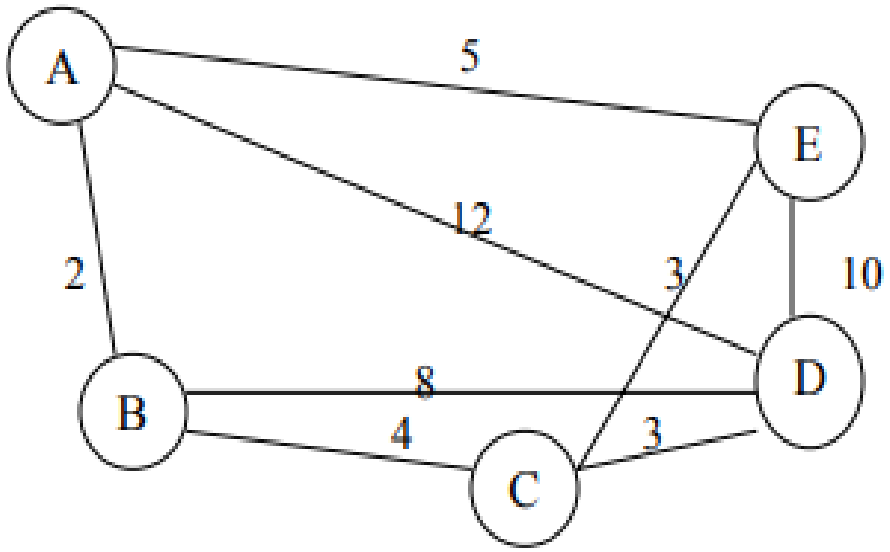


Figure 10.1 A graph with weights on its edges.

The problem lies in finding a minimal path passing from all vertices once. For example the path Path1 {A, B, C, D, E, A} and the path Path2 {A, B, C, E, D, A} pass all the vertices but Path1 has a total length of 24 and Path2 has a total length of 31.

Example 6: The Travelling Salesman Problem

- Problem Formulation for Travelling Salesman Problem:
  - ❖**State:** Any City which is not traversed before
  - ❖**Initial State:** Any City can be the starting city
  - ❖**Action:** Go to city S(A)={In(A)} -> {Go(D),In(D)}
  - ❖**Goal Test:** Shortest Path
  - ❖**Path Cost:** No. of actions to reach the goal
    - ❖Note: Suppose if Each city visiting costs 1, so the path cost is the no. of cities visited in the path

Search Strategies

Search Strategies

Requirements of a good search strategy:

1. It causes motion

   Otherwise, it will never lead to a solution.

2. It is systematic

   Otherwise, it may use more steps than necessary.

3. It is efficient

   Find a good, but not necessarily the best, answer.

SEARCH STRATEGIES

1. **Uninformed/ blind Search-**
   - Search strategies use only the information available in the problem definition.
   - Having No Information About The Number Of Steps From The Current State To The Goal.
2. **Informed/ heuristic search -**
   - When Strategies Can Determine Whether One Non-goal State Is Better Than Another
   - More efficient than uninformed search

UNINFORMED SEARCH ALGORITM

Uninformed search strategies

Compare different Uninformed Search Techniques.

(JUNE 2011, JUNE 2013)

- **Uninformed:** While searching you have no clue whether one non-goal state is better than any other. Your search is blind.
- **Various blind strategies:**
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth Limited Search
  5. Iterative deepening search

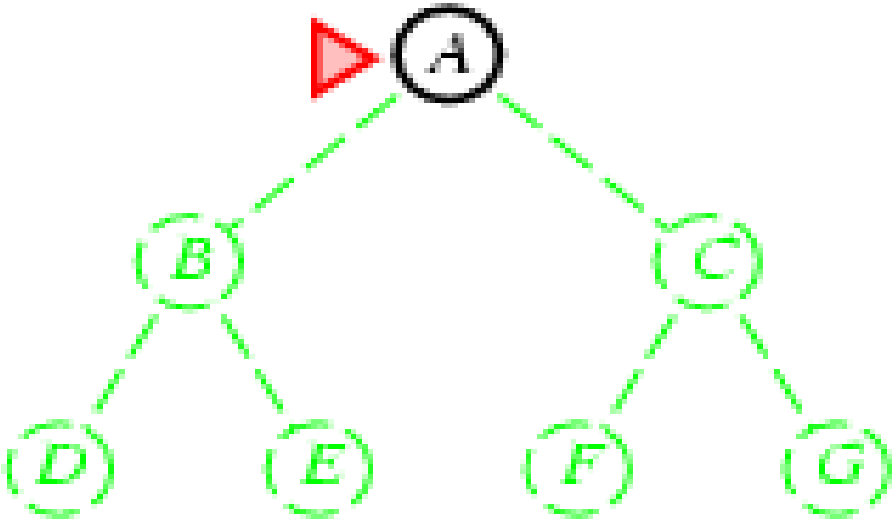
1. Breadth-first search

Compare and contrast BFS and DFS. And explain the search strategy developed to overcome the drawback of both.

(DEC 2010)

- Expand shallowest unexpanded node
- Implementation:
  - *Fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?

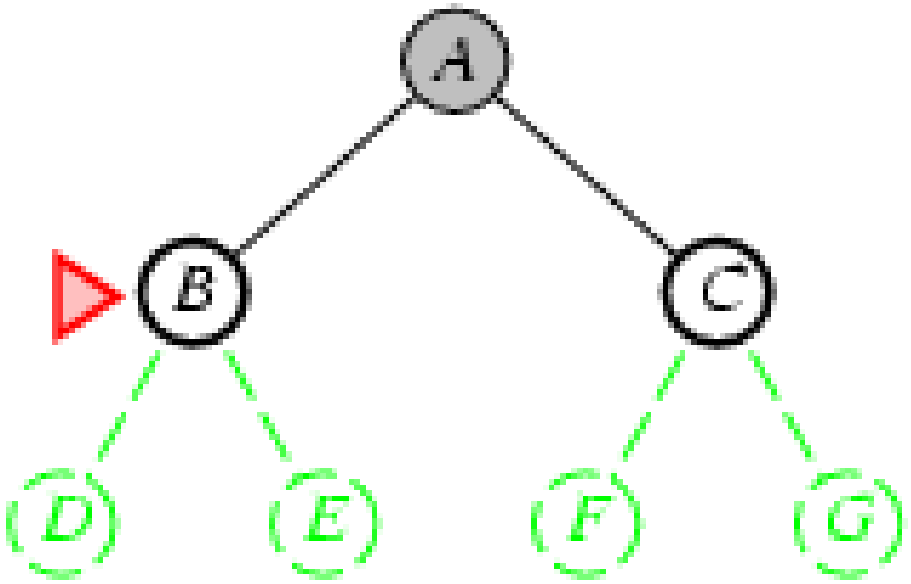


1. Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *Fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe = [B,C]

Is B a goal state?

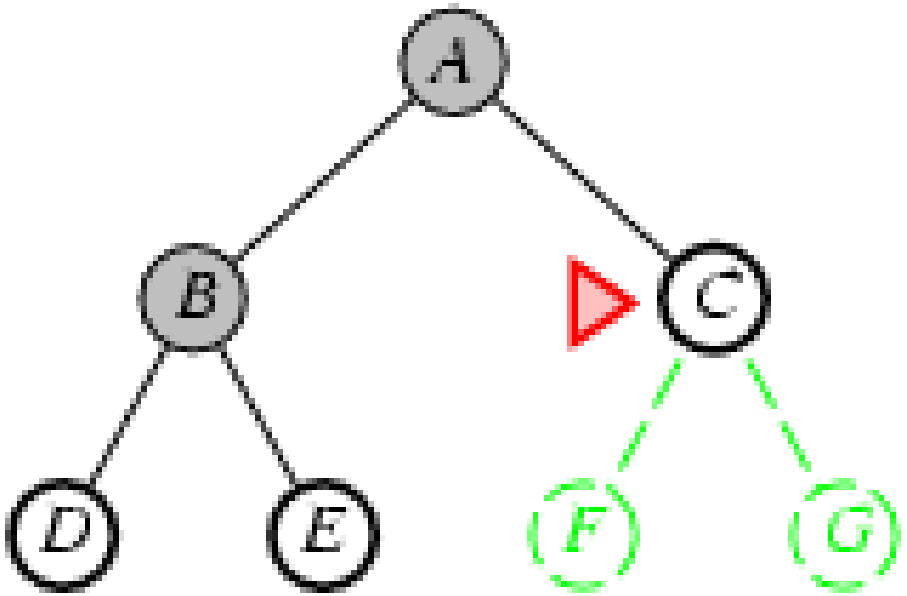


1. Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[C,D,E]

Is C a goal state?



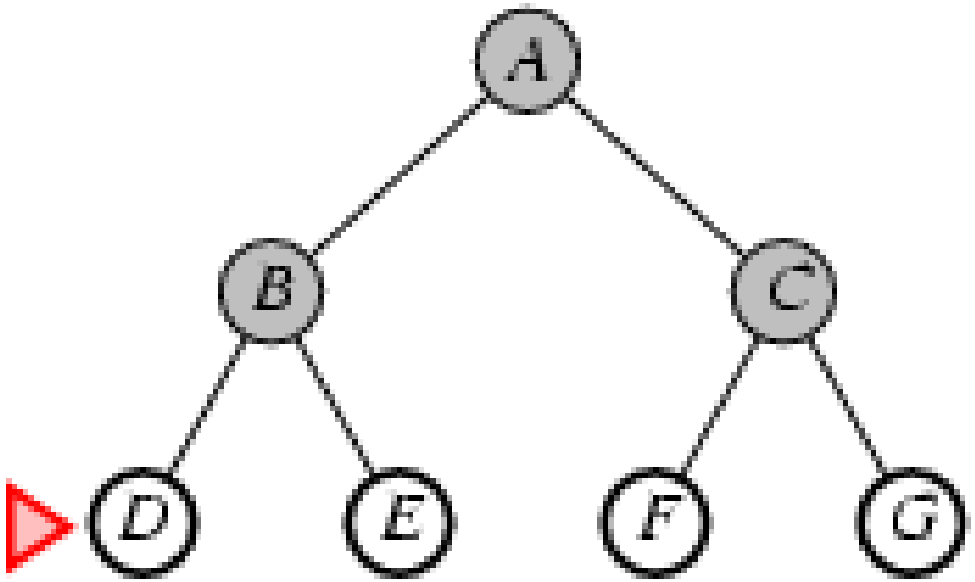
1. Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[D,E,F,G]

Is D a goal state?

Yes



1. Breadth-first search

Algorithm BREADTH: **Breadth first search in state space**

1. Init lists OPEN ← {S_i}, CLOSED ← {}
2. **if** OPEN = {}
   then return FAIL
3. Remove first node S from OPEN and insert it in CLOSED
4. Expand node S
   - 4.1. Generate all direct successors S_j of node S
   - 4.2. **for** each successor S_j of S **do**
     - 4.2.1. Make link $S_j \rightarrow S$
     - 4.2.2. **if** S_j is final state
       then
       - i. Solution is (S_j, S, .., S_i)
       - ii. **return** SUCCESS
     - 4.2.3. Insert S_j in OPEN, ***at the end***
5. **repeat from** 2

end.

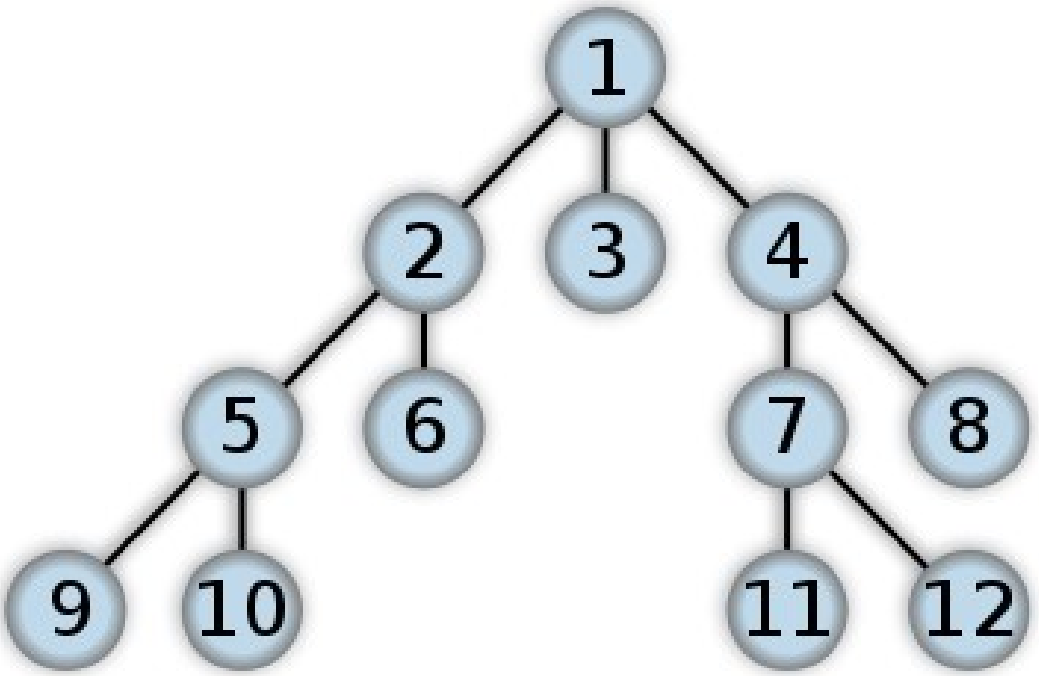
Properties of breadth-first search

- Complete? Yes it always reaches goal (if b is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d+(b^{d+1}-b)) = O(b^{d+1})$
  (this is the number of nodes we generate)
- Space? $O(b^{d+1})$ (keeps every node in memory, either in fringe or on a path to fringe).
- Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

Space is the bigger problem (more than time)

Solve it using BFS

Start node (1), goal node (8), give path?



2. Uniform-cost search

Note: If all step costs are equal, this is identical to breadth First Search

Uniform-cost Search: Expand node with smallest path cost g(n). (Expand Least cost unexpanded node)

Implementation:

fringe = Queue ordered by path cost (priority queue)

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant).

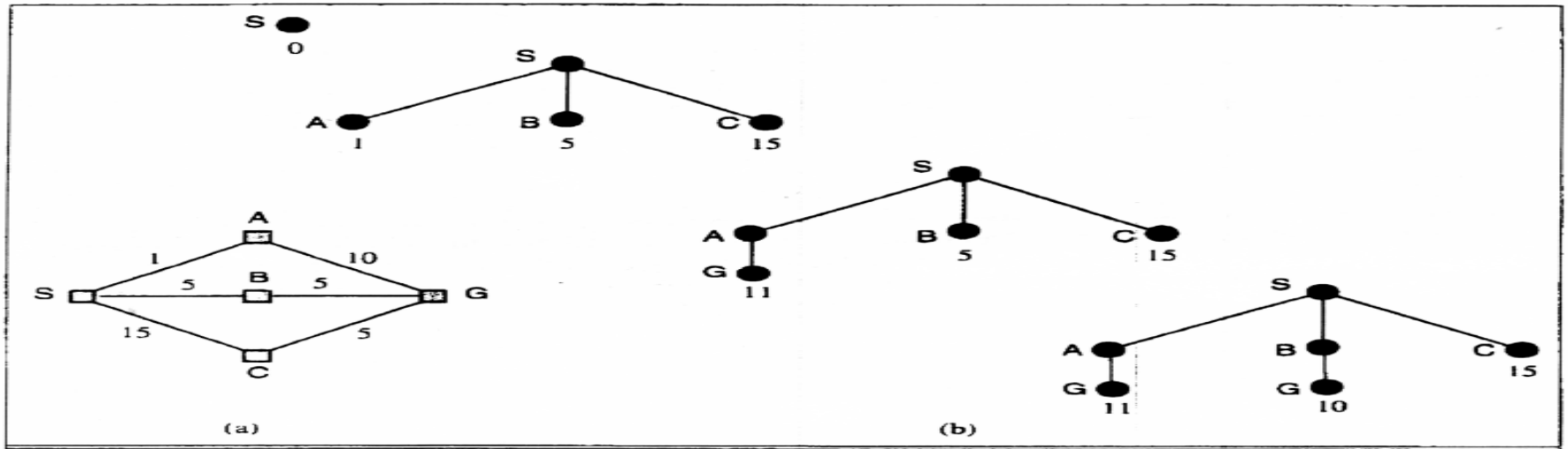


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

2. Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost ≥ ε
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* ≤ cost of optimal solution.

Space? # of nodes on paths with path cost ≤ cost of optimal solution.

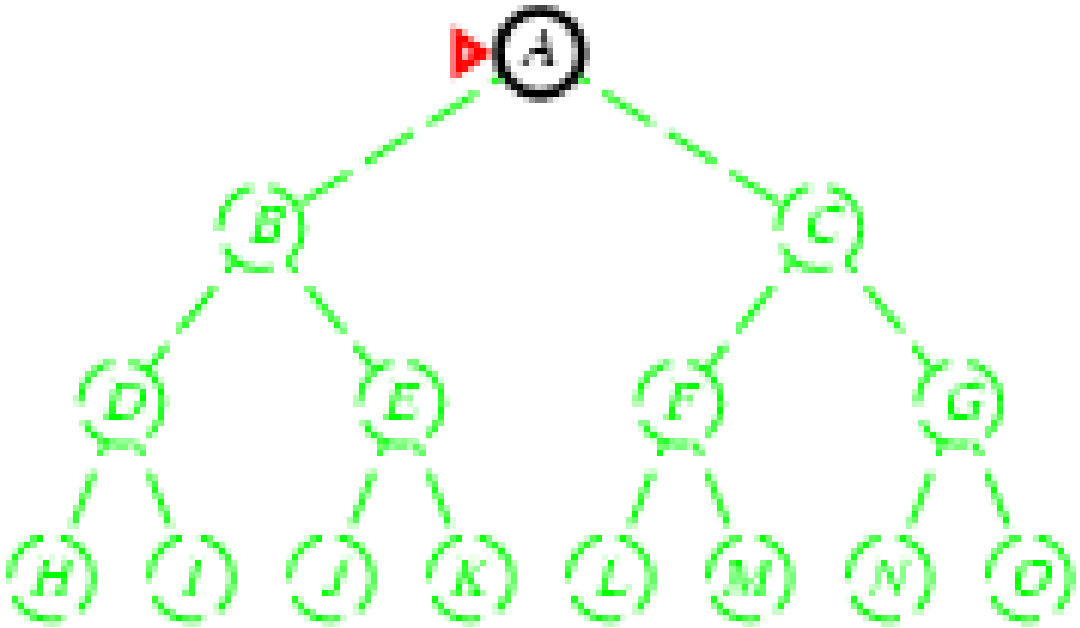
Optimal? Yes, for any step cost.

3. Depth-first search

Explain Depth First Search with example. (DEC 2012)

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIFO) stack, i.e., put successors at front

Is A a goal state?

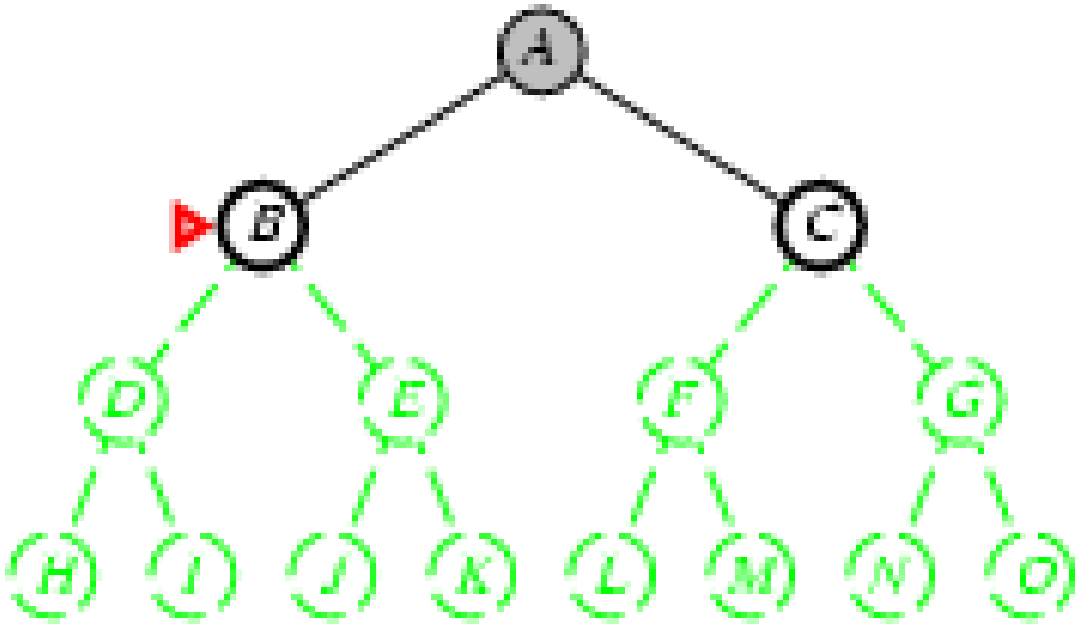


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[B,C]

Is B a goal state?

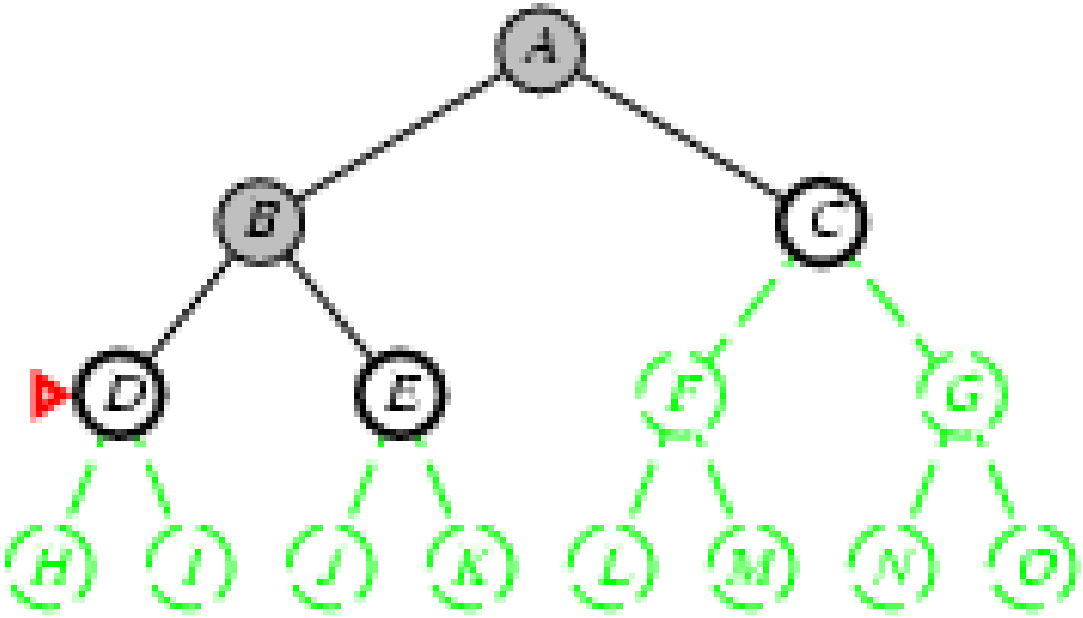


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[D,E,C]

Is D = goal state?

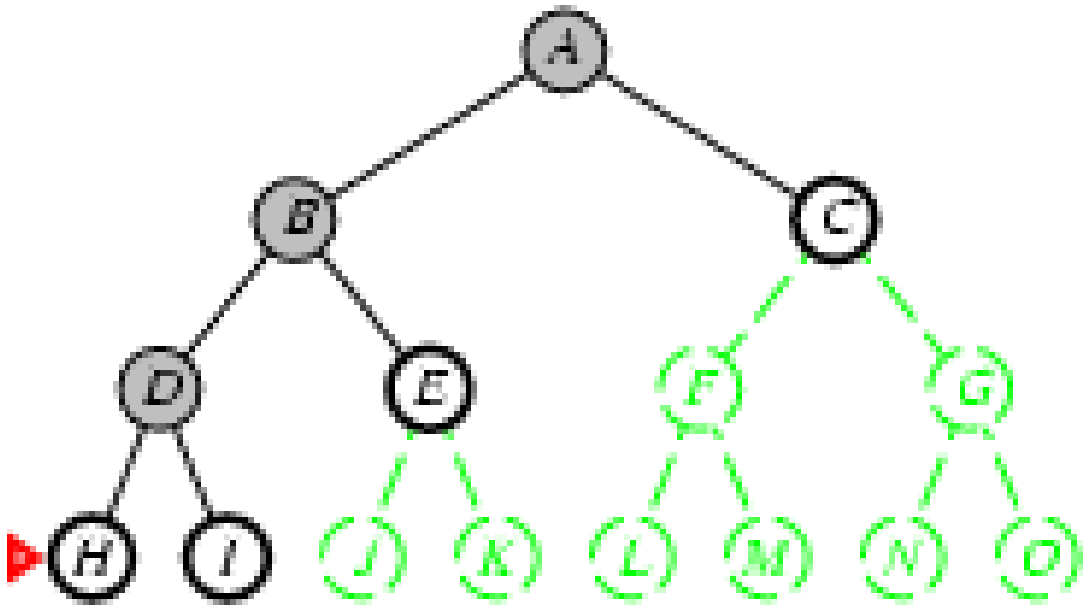


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[H,I,E,C]

Is H = goal state?

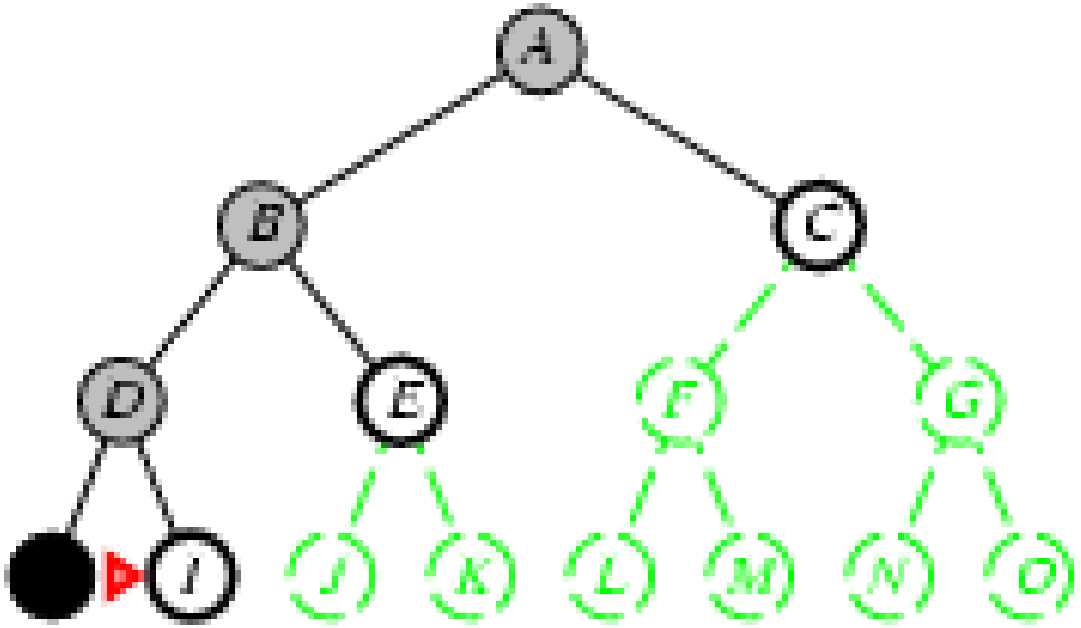


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[I,E,C]

Is I = goal state?

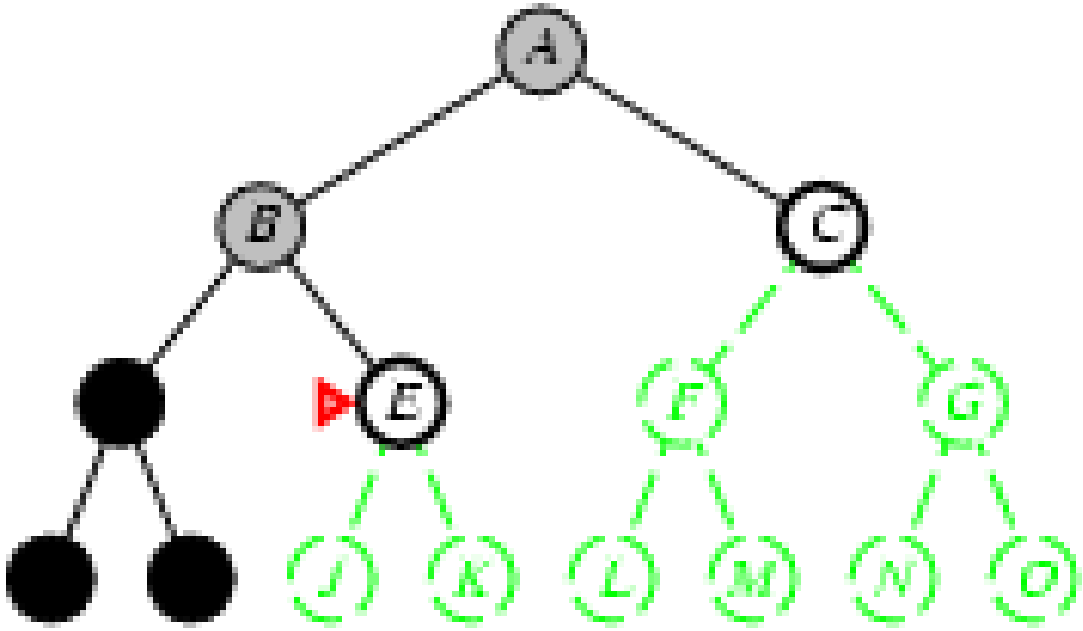


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[E,C]

Is E = goal state?

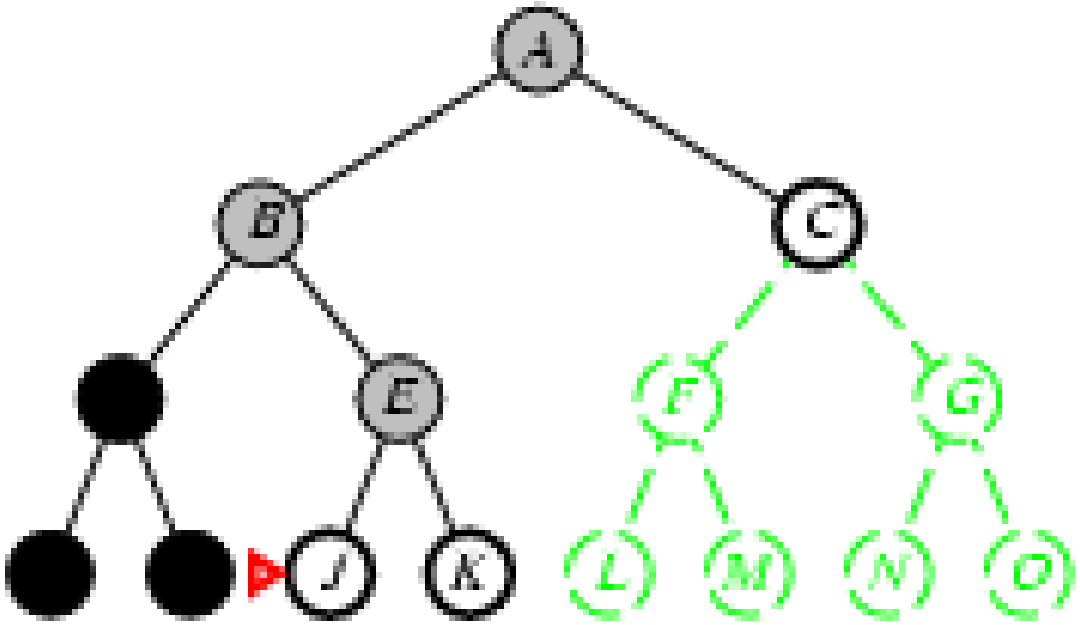


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[J,K,C]

Is J = goal state?

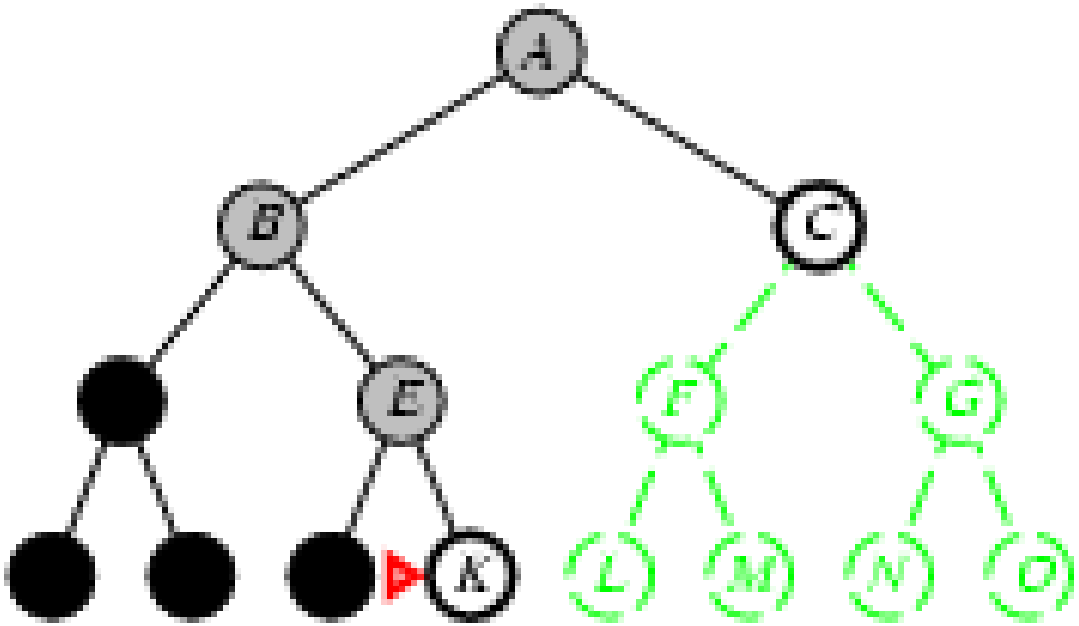


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[K,C]

Is K = goal state?

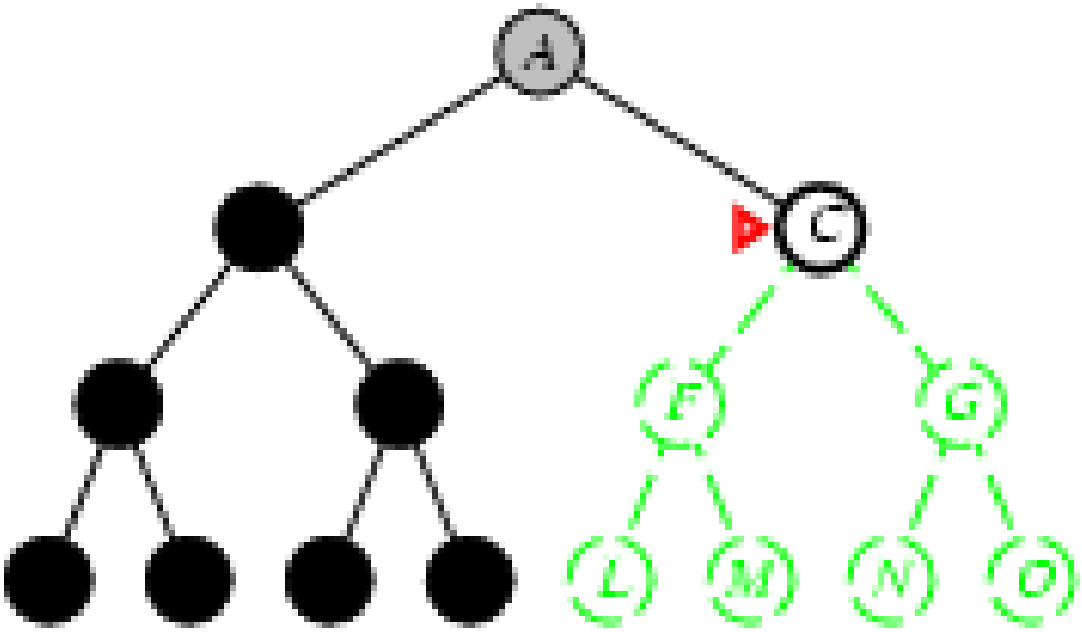


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[C]

Is C = goal state?

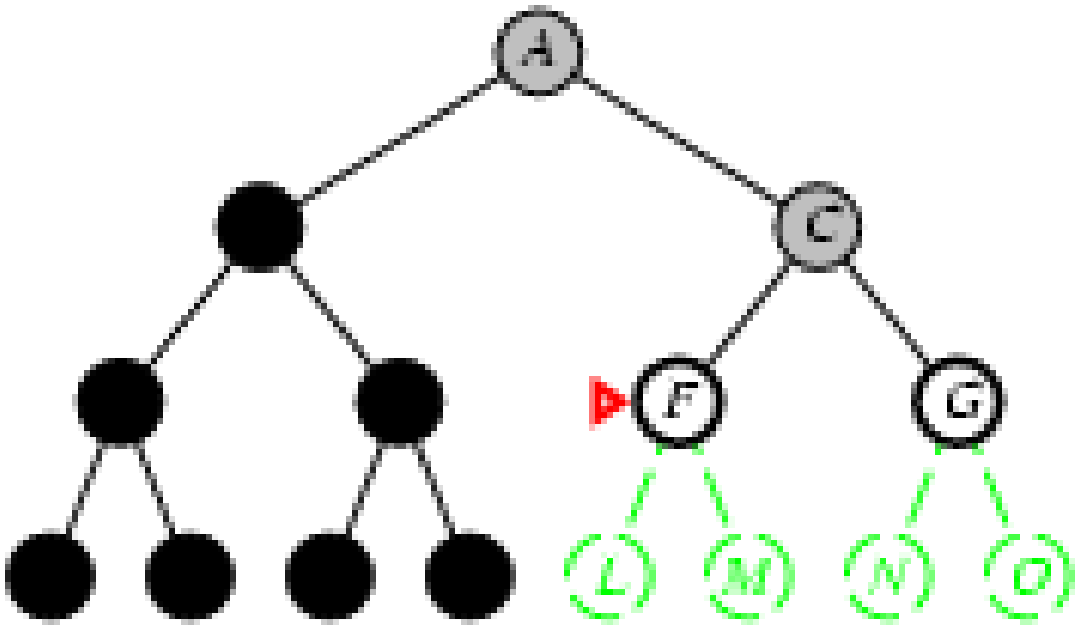


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[F,G]

Is F = goal state?

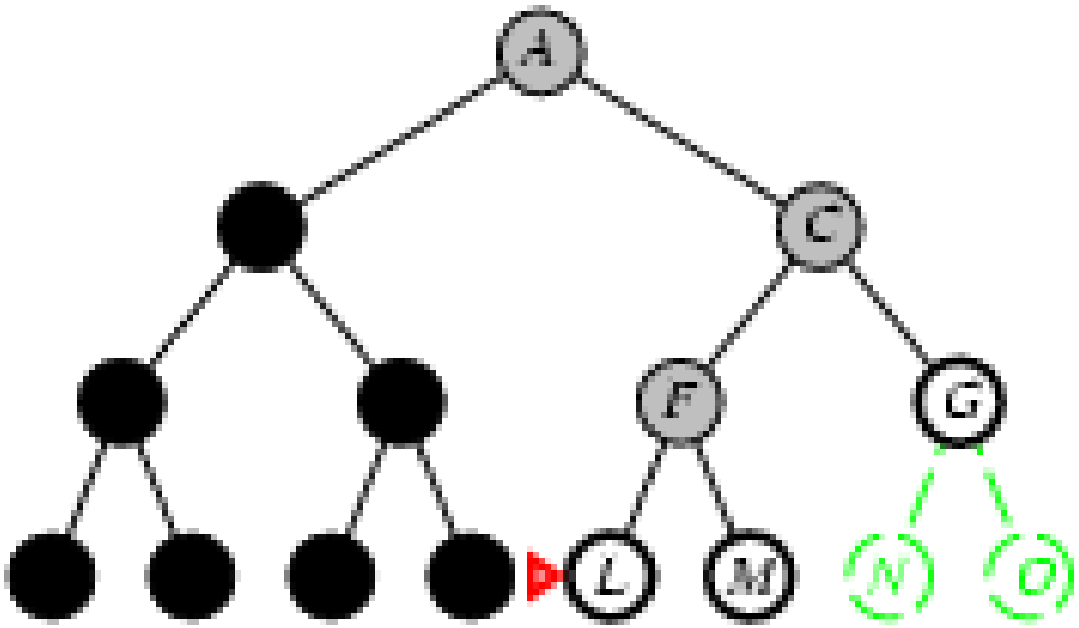


3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[L,M,G]

Is L = goal state?



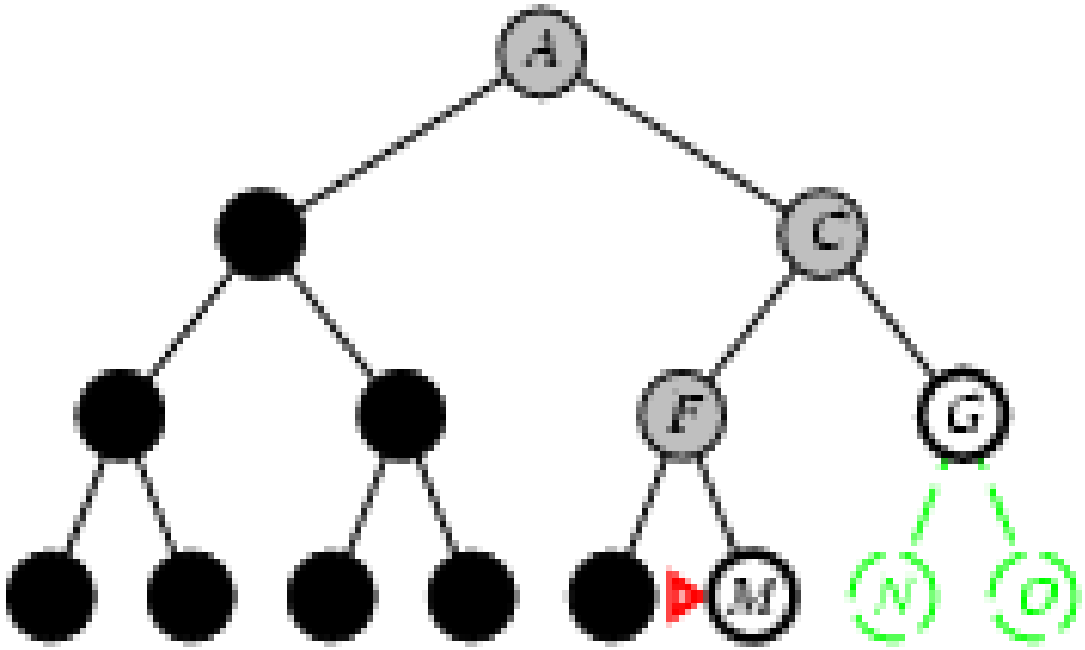
3. Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO stack, i.e., put successors at front

queue=[M,G]

Is M = goal state?

Yes



3. Depth-first search

Algorithm DEPTH(AdMax**)**: **Depth first search in state space**

1. Init lists OPEN ← {S_i}, CLOSED ← {}
2. **if** OPEN = {}
   then return FAIL
3. Remove first node S from OPEN and insert it into CLOSED

3'. **if** Ad(S) = AdMax **then repeat from 2**

4. Expand node S
   - 4.1. Generate all successors S_j of node S
   - 4.2. **for** each succesor S_j of S **do**
     - 4.2.1. Set link $S_j \rightarrow S$
     - 4.2.2. **if** S_j is final state
       then
       - i. Solution is (S_j,.., S_i)
       - ii. **return** SUCCESS
     - 4.2.3. Insert S_j in OPEN, ***at the beginning***

5. **repeat from** 2

end.

MU Question on DFS (May 2016) 10 Marks

Consider the graph given in Figure 3 below. Assume that the initial state is A and the goal state is G Find a path from the initial state to the goal state using DFS. Also report the solution cost

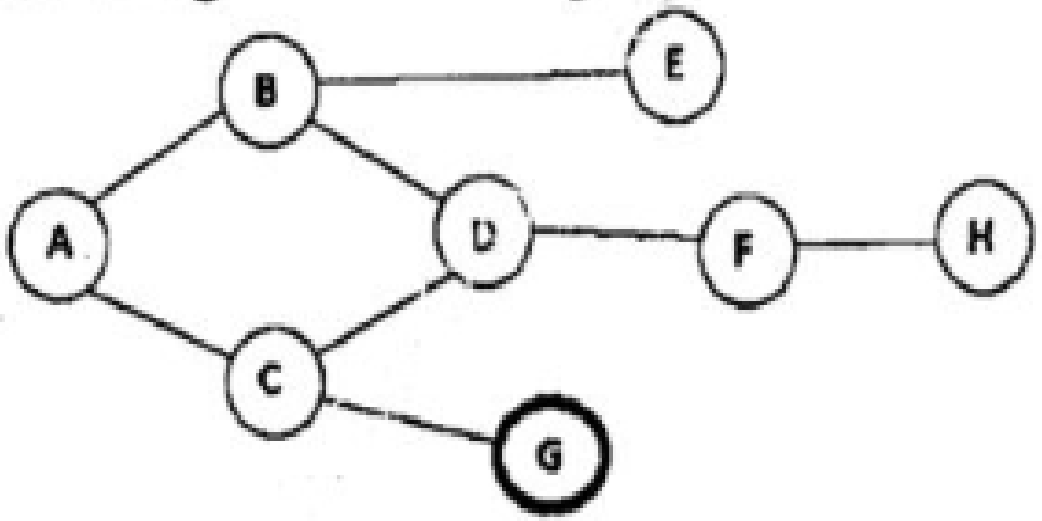
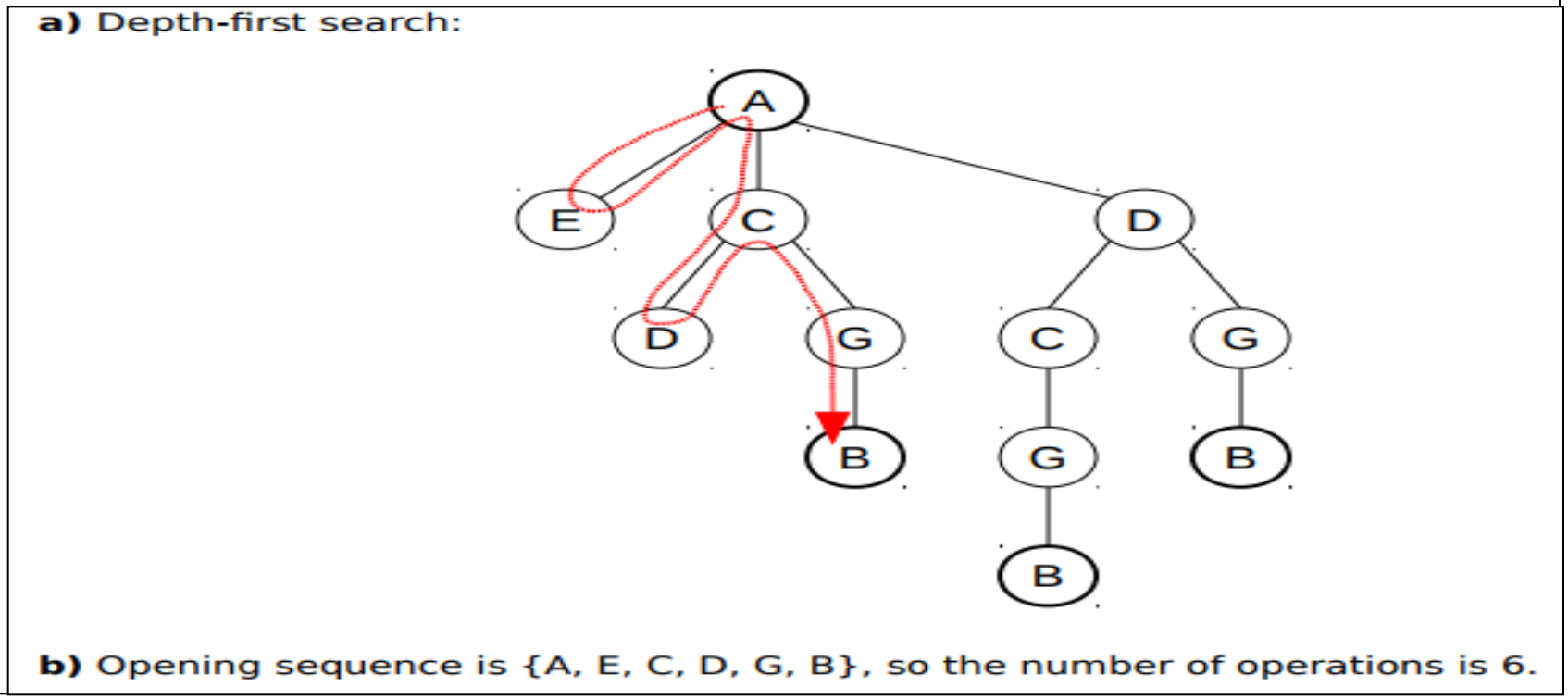


Figure 3.

Another example

- To find Cost in Tree:
  - **Path cost:** For example, this could be the number of nodes opened while reaching the current node from



3. Properties of depth-first search

- Complete? No: fails in infinite-depth spaces

  Can modify to avoid repeated states along path
- Time? $O(b^m)$ with m=maximum depth
- terrible if *m* is much larger than *d*
  - but if solutions are dense, may be much faster than breadth-first
- Space? *O(bm),* i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)
- Optimal? No (It may find a non-optimal goal first)

4. Depth Limited Search

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don't expand beyond depth L.

→ Depth-Limited Search = Depth first Search with depth limited l (i.e. node at depth l have no successors)

4. Properties of depth-Limited search

- Is DF-search with depth limit l.
  - i.e. nodes at depth l have no successors.
  - Problem knowledge can be used
- Solves the infinite-path problem.
- If $l < d$ then incompleteness results.
- If $l > d$ then not optimal.
- Time complexity: $O(b^l)$
- Space complexity: $O(bl)$

5. Iterative Deepening Search- DFS with increasing depth limit

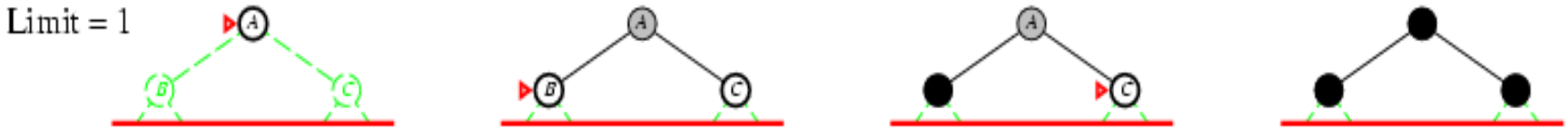
- What of solution is deeper than L? → Increase L iteratively.
  → Iterative Deepening Search

- Finds the best depth limit
  ✓ Combines the benefits of DFS and BFS. As we shall see: this inherits the memory advantage of Depth-First search.

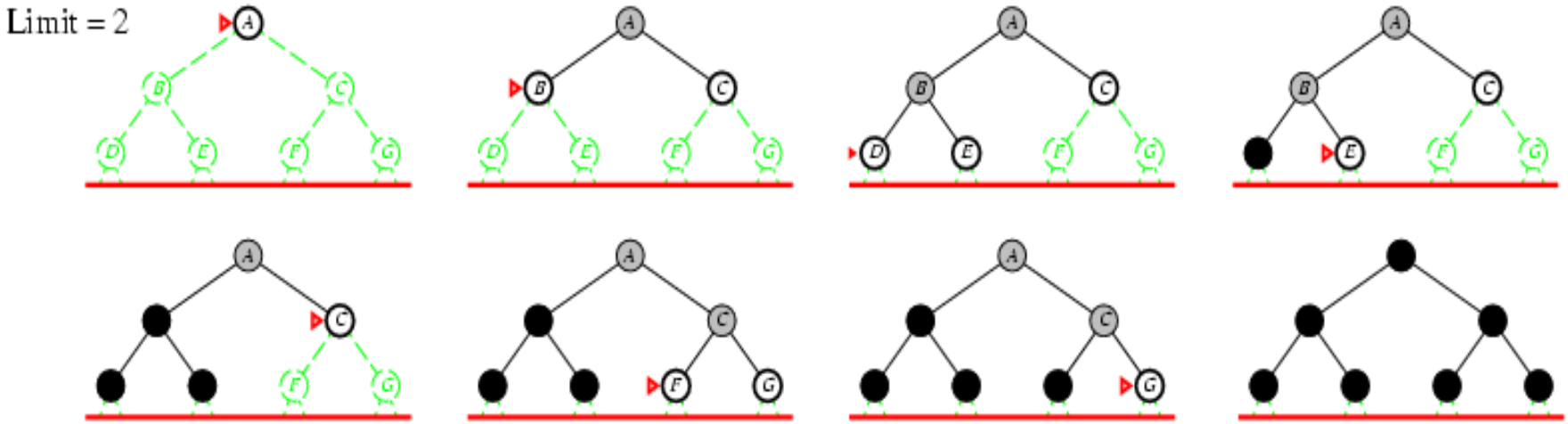
Iterative deepening search *L*=0

Limit = 0

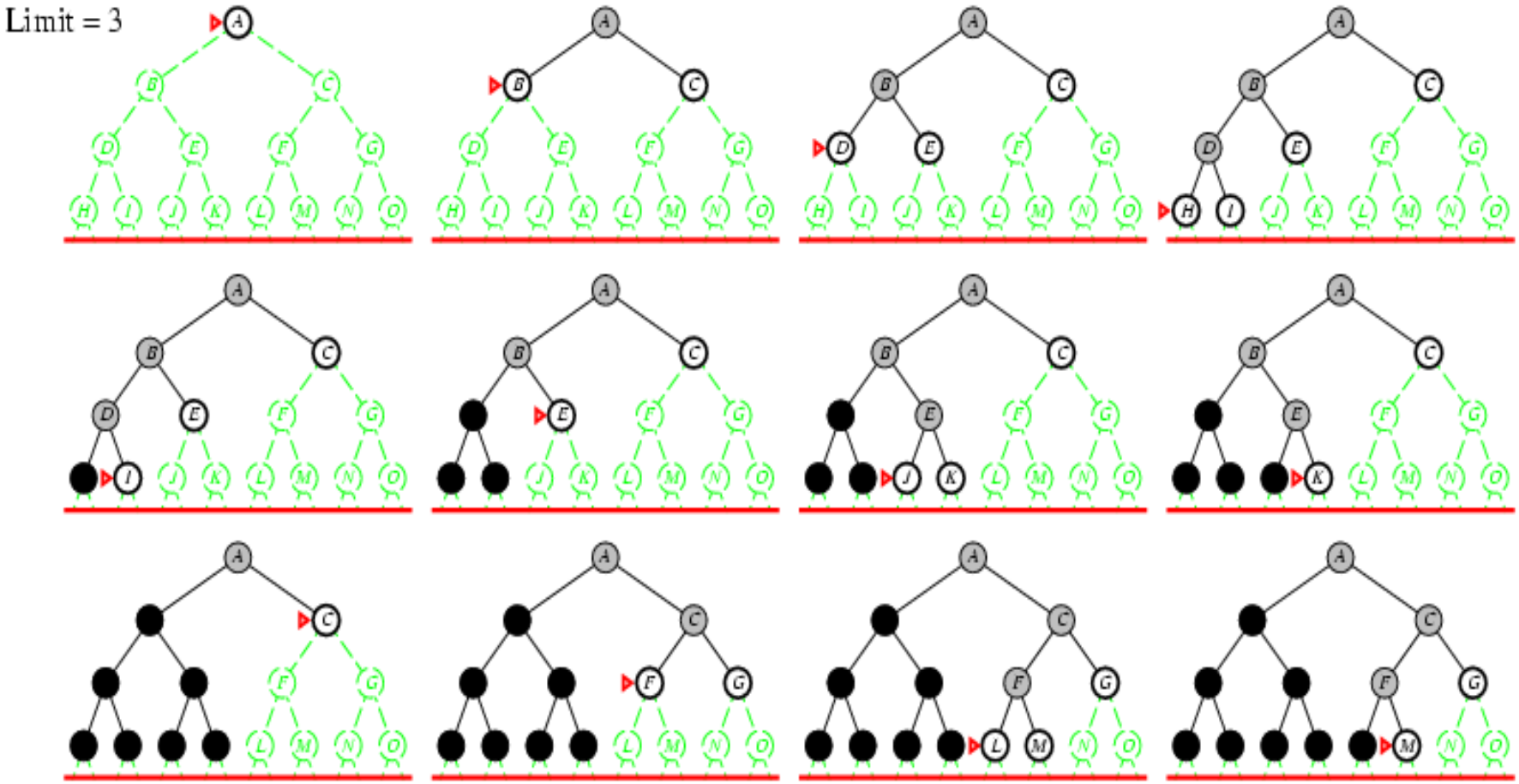
Iterative deepening search *L*=1



Iterative deepening search *l*L=2



Iterative deepening search *l*=3



5. Properties of Iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

Summary of Uniformed Search algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

MCQ on Uninformed search

1. **Which search is implemented with an empty first-in-first-out queue?**

   a)Depth-first search

   b)Breadth-first search

   c)Bidirectional search

   d) None of the mentioned

2. **Which search implements stack operation for searching the states?**

   a)Depth-first search

   b)Breadth-first search

   c)Bidirectional search

   d) None of the mentioned

MCQ on Uninformed search

3. Which search algorithm imposes a fixed depth limit on nodes?

a)Depth-limited search

b)Depth-first search

c)Iterative deepening search

d) Bidirectional search

INFORMED SEARCH ALGORITM

Informed search strategies

- Informed: When Strategies Can Determine Whether One Non-goal State Is Better Than Another.
  - More efficient than uninformed search.

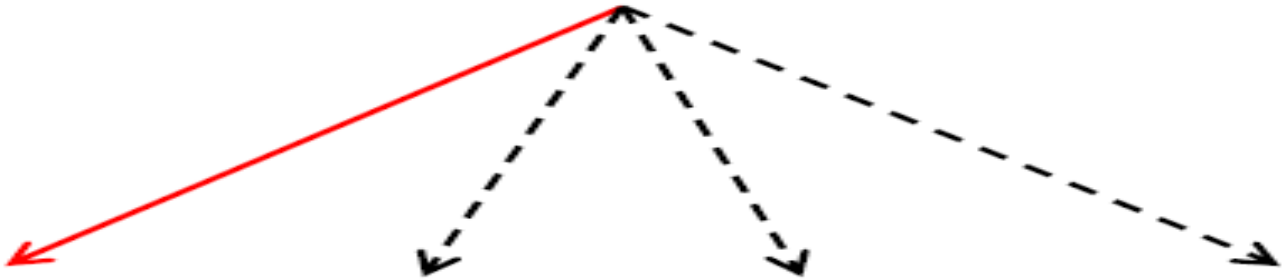
- Various Search strategies:
  1. Best First Search
  2. Generate and Test
  3. A*
  4. IDA*

University Asked Questions on Informed

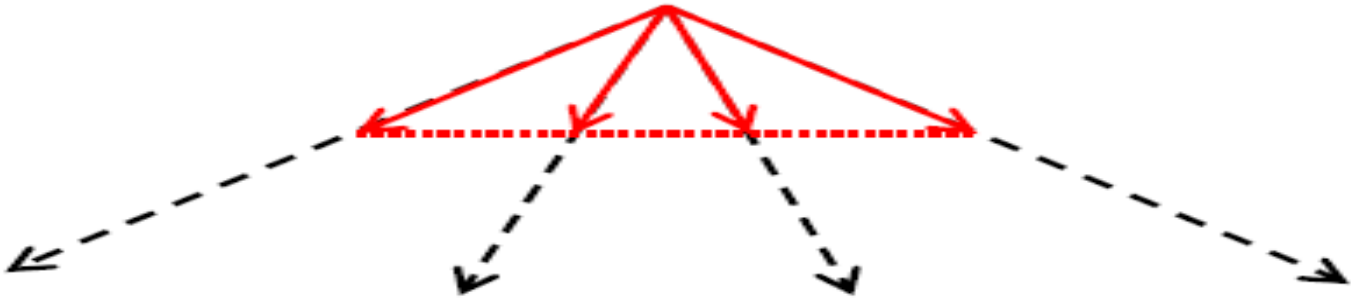
1. Explain Heuristic function with example? (DEC 2011, DEC 2012)
2. Give and Explain A* Algorithm. Drawbacks of A* and Show that A* is optimally efficient? (DEC 2010, DEC 2011, JUNE 2013)
3. What do you mean by admissible heuristic search function, explain with example? (DEC 2012)
4. Give and Explain IDA* Search Algorithm. (DEC 2012, JUNE 2011)

Drawbacks of BFS and DFS

- Depth-first search:
  - Pro: not having to expand all competing branches
  - Con: getting trapped on dead-end paths

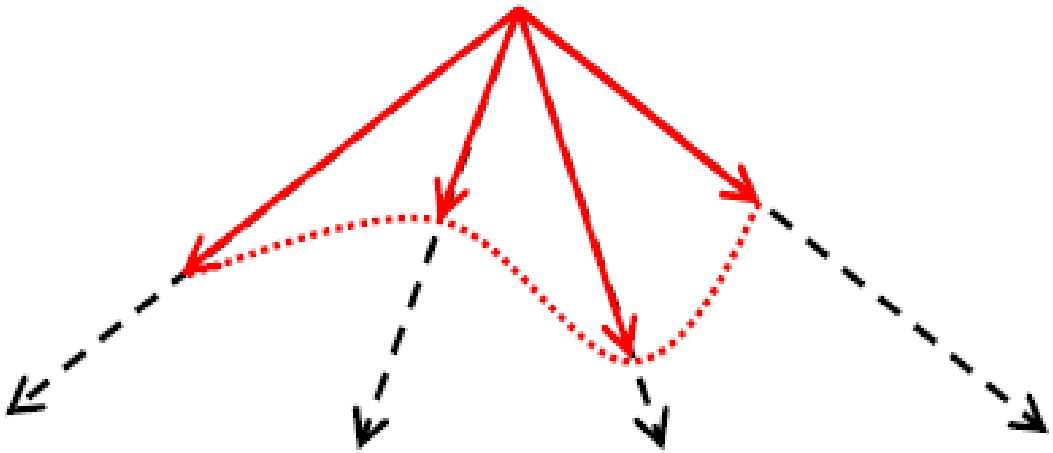


- Breadth-first search:
  - Pro: not getting trapped on dead-end paths
  - Con: having to expand all competing branches



1. Best First Search

⇒ Combining the two is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one.



Not having to expand all competing branches and Not getting trapped at dead ends

1. Best First Search

- OPEN: nodes that have been generated, but have not examined.

  This is organized as a priority queue.

- CLOSED: nodes that have already been examined.

  Whenever a new node is generated, check whether it has been generated before.

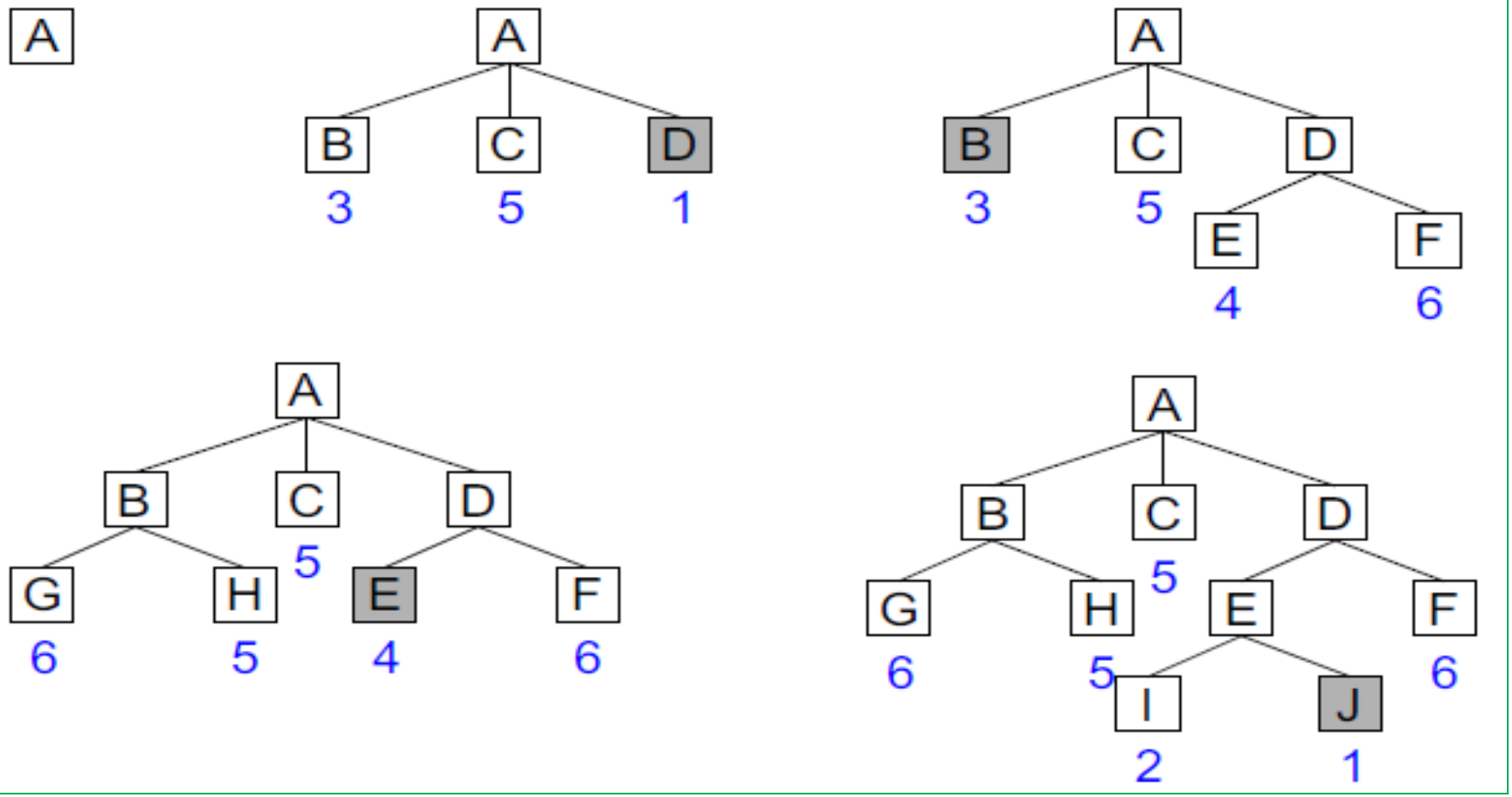
Algorithm

1. OPEN = {initial state}.
2. Loop until a goal is found or there are no nodes left in OPEN:
   - Pick the best node in OPEN
   - Generate its successors
   - For each successor:

     new → evaluate it, add it to OPEN, record its parent

     generated before → change parent, update successors

1. Best First Search-Example



2. Generate and Test

Algorithm

1. Generate a possible solution.
2. Test to see if this is actually a solution.
3. Quit if a solution has been found. Otherwise, return to step 1.

Advantage & Disadvantage:

- Acceptable for simple problems.
- Inefficient for problems with large space.

MU Asked Questions based on Heuristic Function

1. Explain Heuristic function with example? (DEC 2011, DEC 2012)
2. Define Heuristic Function. Give an example of heuristic function for blocks world Problem. Dec 2015, Dec 2016 (5 marks, DEC 2011, DEC 2012)
3. Find Heuristics value for a particular state of the Blocks World (5 marks) Dec 2015

Heuristic Function

Explain Heuristic function with example? (DEC 2011, DEC 2012)

- The heuristic function h(N) ≥ 0 estimates the cost to go from STATE(N) to a goal state

  Its value is **independent of the current search tree**; it depends only on STATE(N) and the goal test GOAL?

- Example:

5		8
4	2	1
7	3	6

STATE(N)

1	2	3
4	5	6
7	8	

Goal state

$h_1(N)$ = number of misplaced numbered tiles = 6

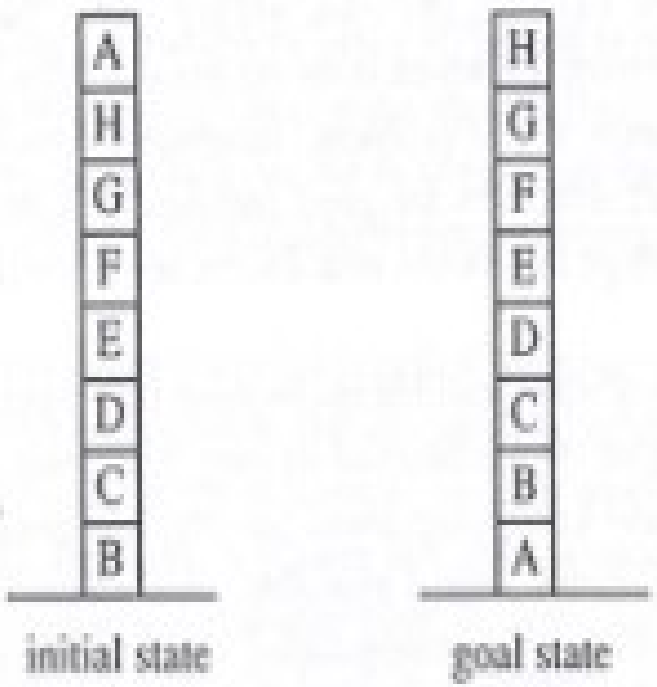
Heuristic Function

Define Heuristic Function. Give an example of heuristic function for blocks world Problem. **Dec 2015, Dec 2016 (5 marks, DEC 2011, DEC 2012)**

Local heuristic function

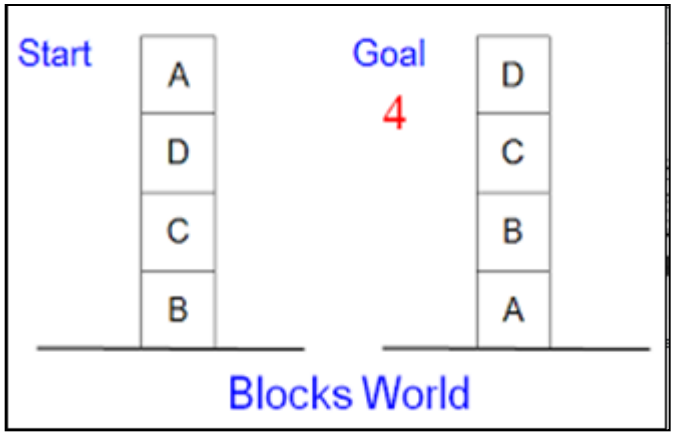
1. Add one point for every block that is *resting on the thing it is supposed to be resting on.*
2. Subtract one point for every block that is *sitting on the wrong thing.*

- Initial state score = 4 (6-2)
  - C,D,E,F,G,H correct = 6
  - A,B wrong = -2
- Goal state score = 8
  - A,B,C,D,E,F,F,H all correct

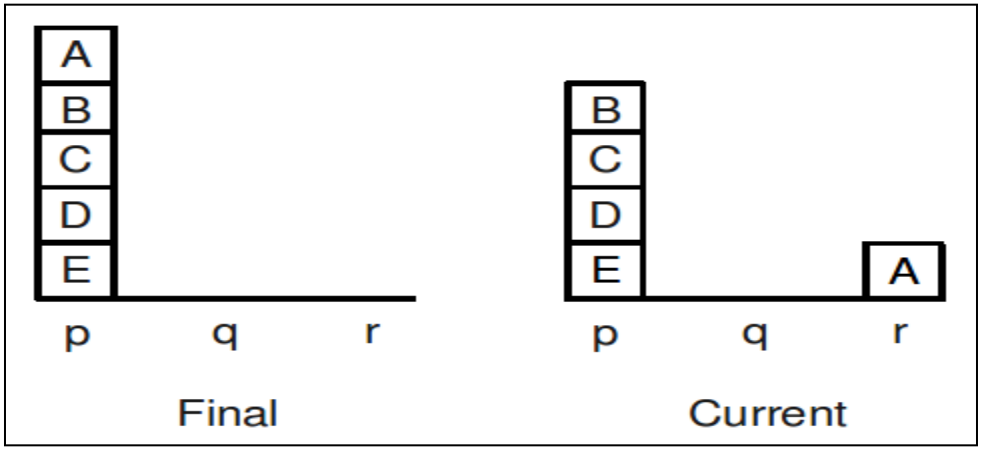


Block world

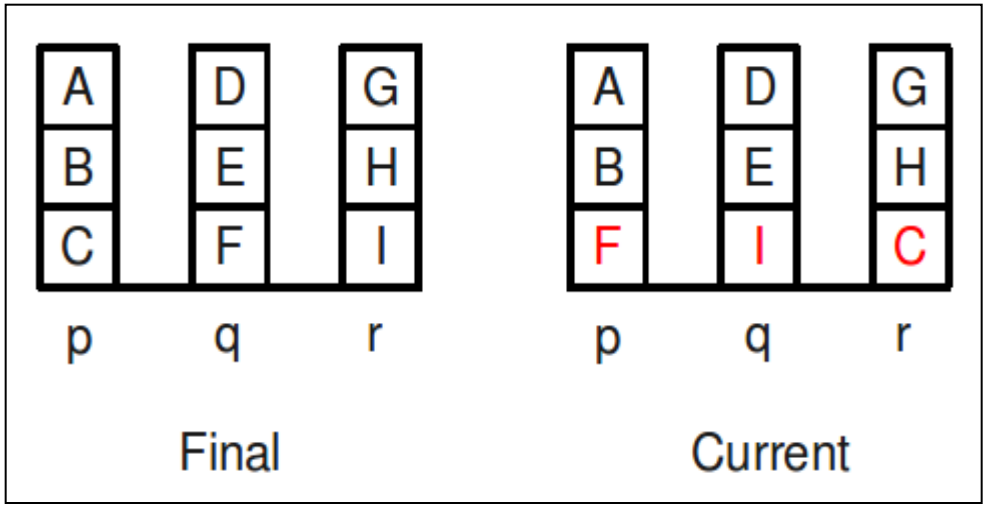
Find heuristic for the given block world problem:



(A)



(B)



(C)

3.Algorithm A

•A is the most widely known form of Best-First search

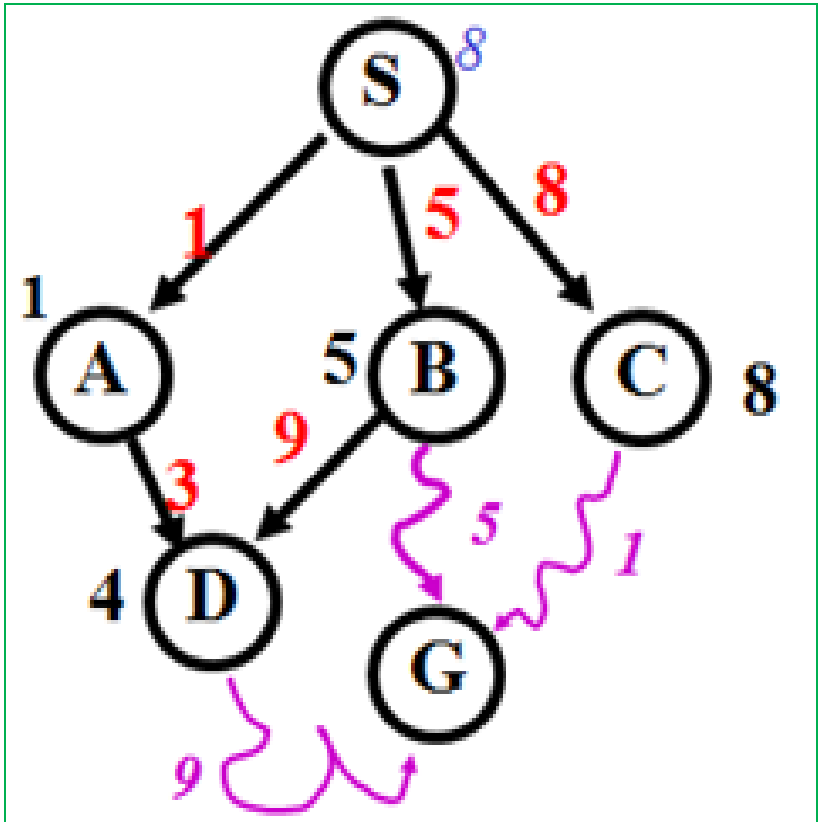
✓It evaluates nodes by combining g(n) and h(n)

f(n) = g(n) + h(n)

•**g(n)** = cost from the initial state to the current state n

•**h(n)** = estimated cost of the cheapest path from node n to a goal node

•**f(n)** = evaluation function to select a node for expansion (usually the lowest cost node)



f(D)=4+9=13
f(B)=5+5=10
f(C)=8+1=9

G is chosen next to expand

3.Algorithm A*

- Algorithm A with constraint that **h(n) <= h*(n)**
- **h*(n)** = true cost of the minimal cost path from n to any goal.
- **g*(n)** = true cost of the minimal cost path from S to n.
- **f*(n) =** h*(n)+g*(n) = true cost of the minimal cost solution path from S to any goal going through n.

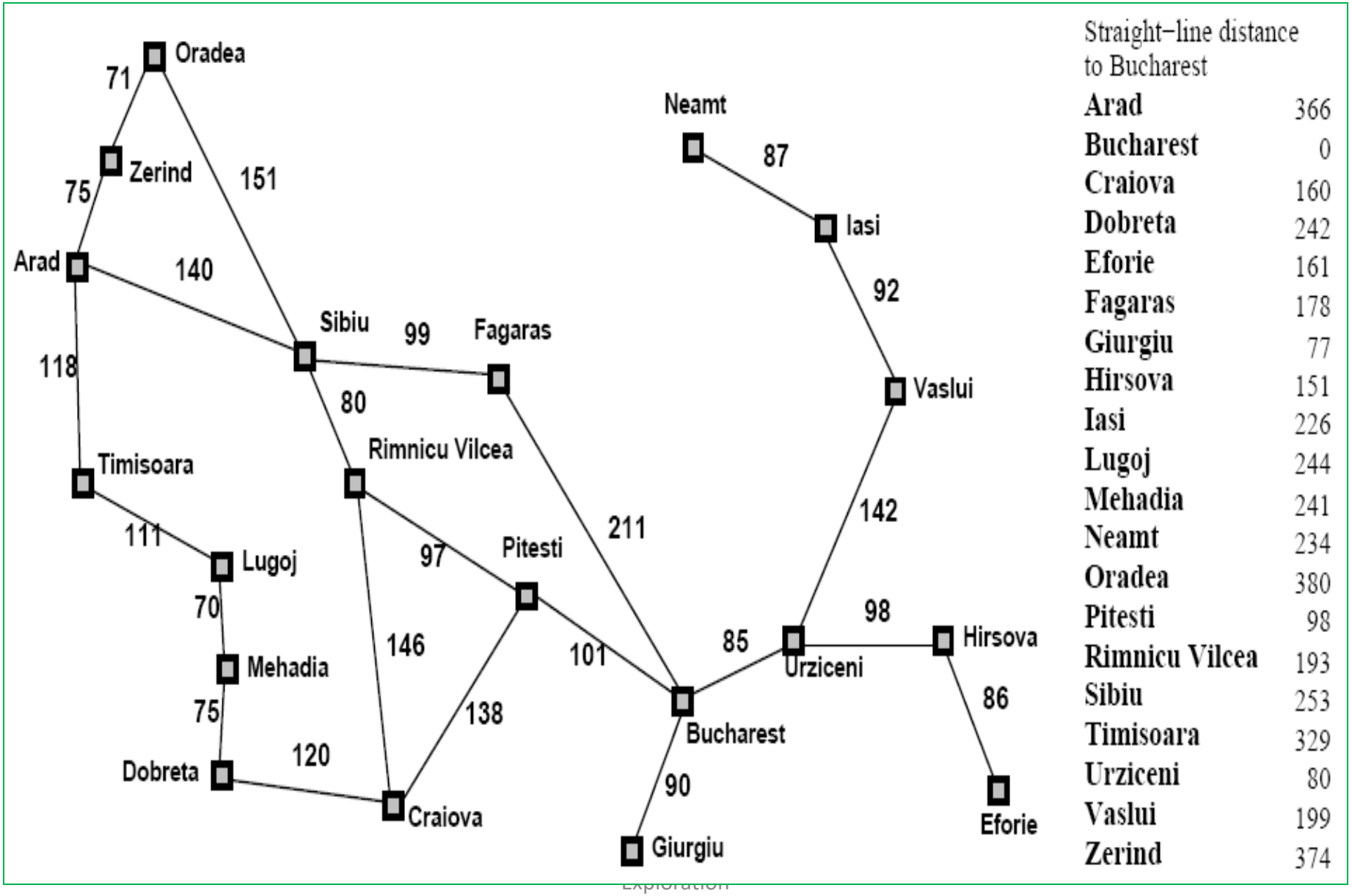
3.Algorithm A*

1. **Start with OPEN containing only initial node.**
   - Set that node's g value to 0, its h' value to whatever it is, and its f' value to h'+0 or h'. Set CLOSED to empty list.
2. **Until a goal node is found, repeat the following procedure:**
   - If there are no nodes on OPEN, report failure.
   - Otherwise pick the node on OPEN with the lowest f' value. Call it BESTNODE. Remove it from OPEN. Place it in CLOSED. See if the BESTNODE is a goal state. If so exit and report a solution.
   - Otherwise, generate the successors of BESTNODE but do not set the BESTNODE to point to them yet.

Algorithm A* (with any h on search Graph)

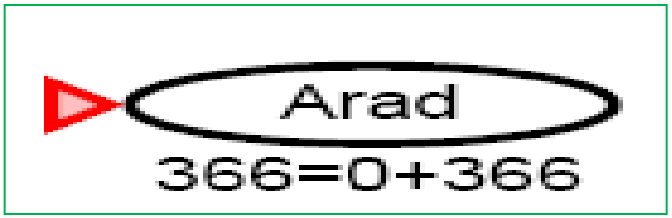
- Input: an implicit search graph problem with cost on the arcs
- Output: the minimal cost path from start node to a goal node.
  - 1. Put the start node s on OPEN.
  - 2. If OPEN is empty, exit with failure
  - 3. Remove from OPEN and place on CLOSED a node n having minimum f.
  - 4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s.
  - 5. Otherwise, expand n generating its children and directing pointers from each child node to n.
    - For every child node n' do
      - evaluate h(n') and compute $f(n') = g(n') + h(n') = g(n) + c(n,n') + h(n')$
      - If n' is already on OPEN or CLOSED compare its new f with the old f. If the new value is higher, discard the node. Otherwise, replace old f with new f and reopen the node.
      - Else, put n' with its f value in the right order in OPEN
  - 6. Go to step 2.

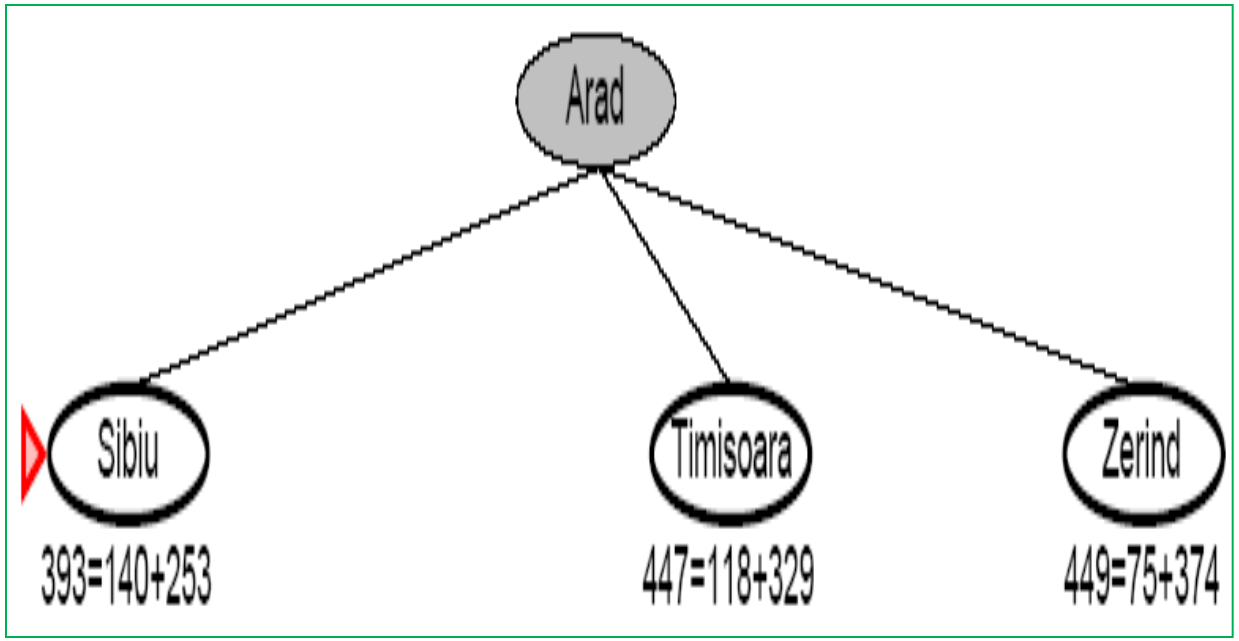
3.A* Search



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

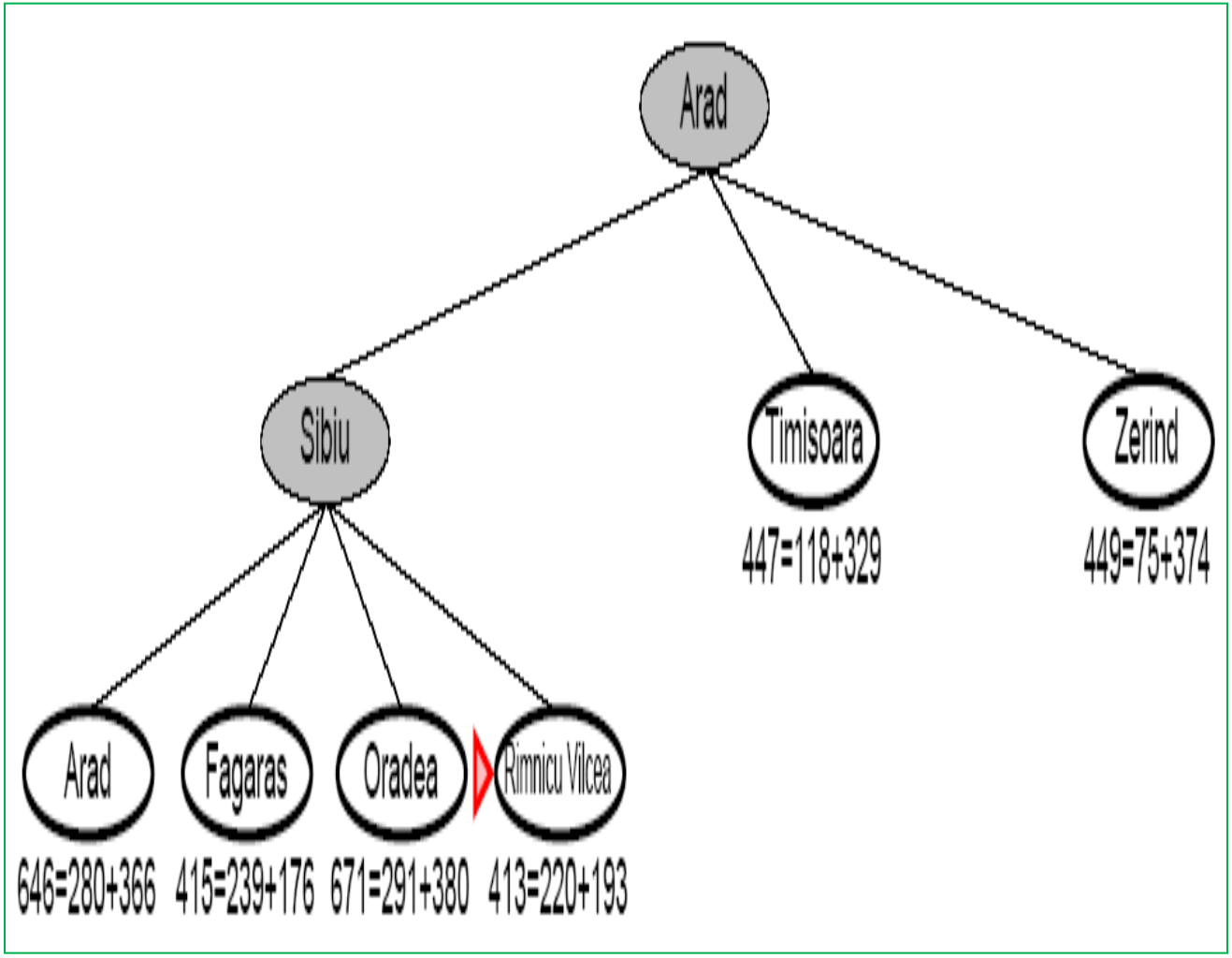
3.A* Search





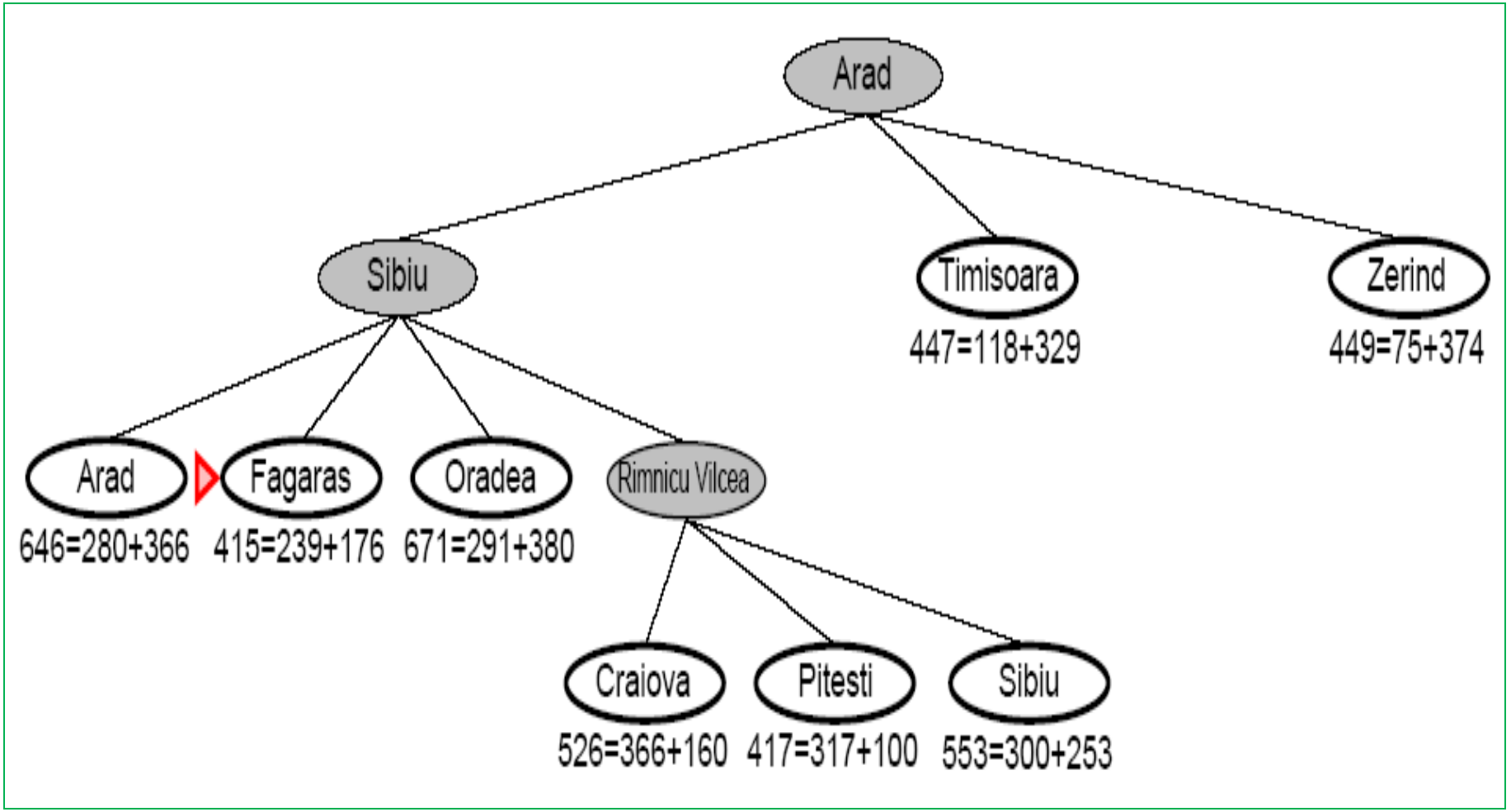
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

3.A* Search

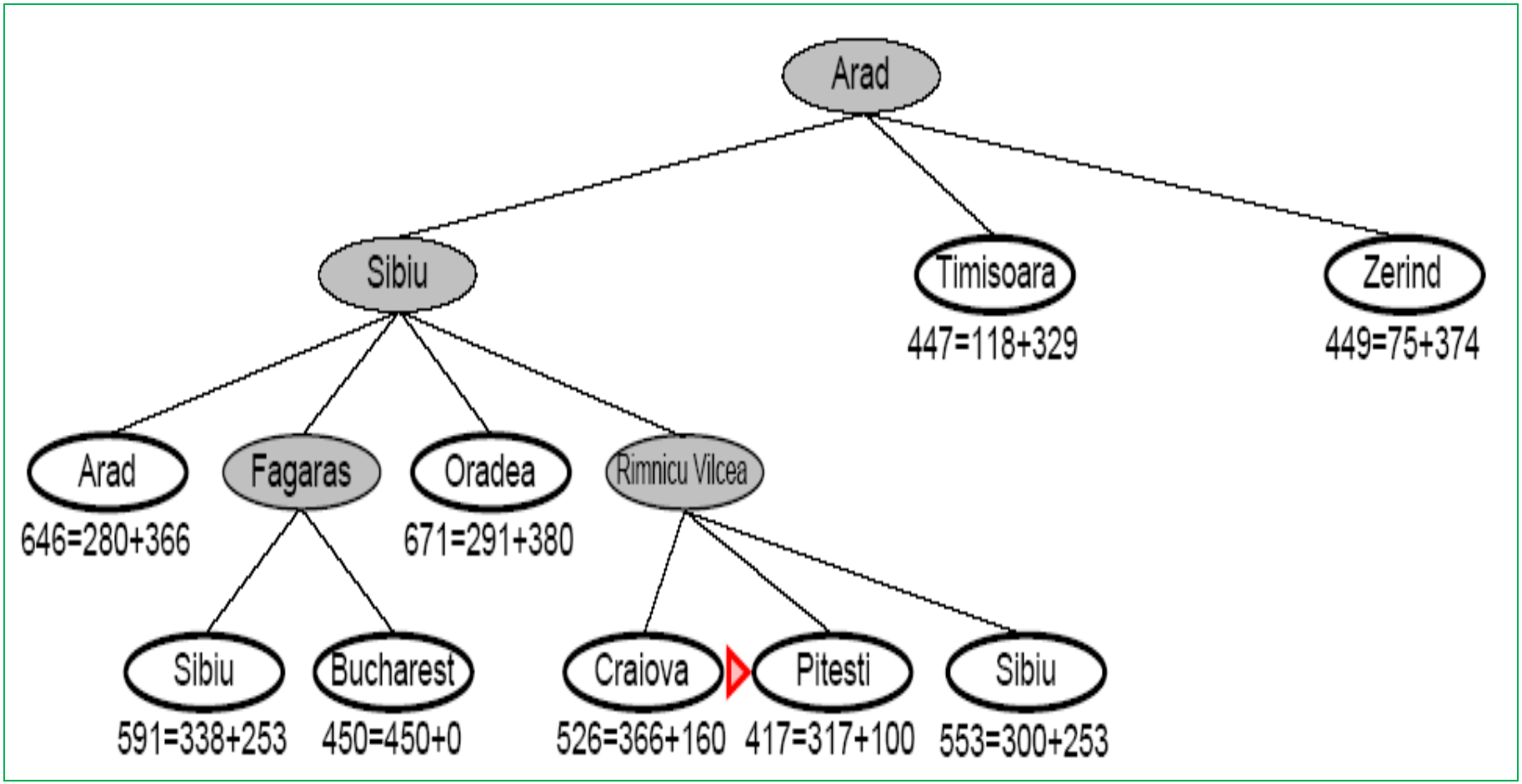


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

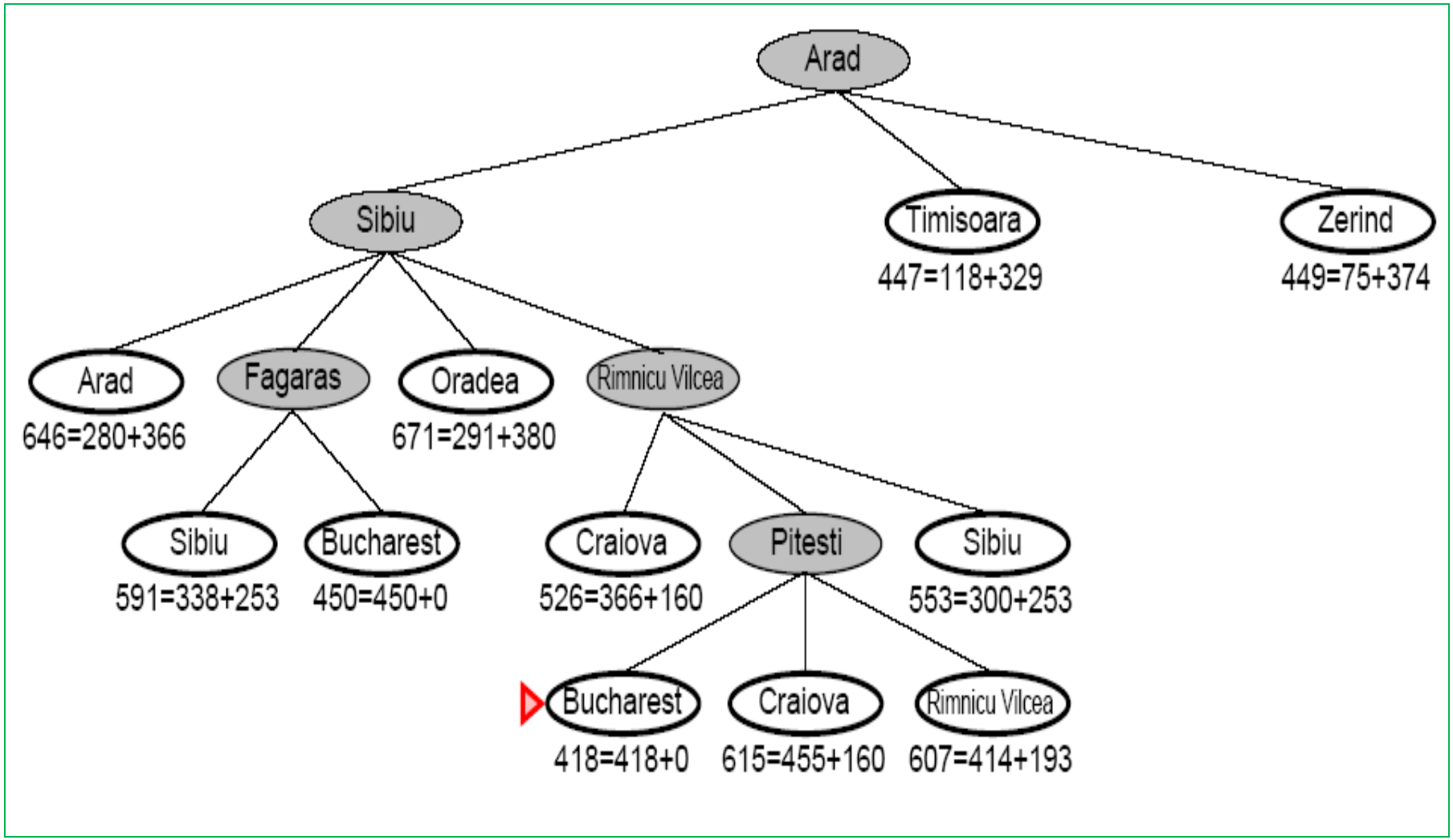
3.A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

3.A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

3.A* Search


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Numerical on A* (Dec 2015) 10 Marks

Q2. (a) Consider the graph given in Figure 1 below. Assume that the initial state is S and the goal state is 7. Find a path from the initial state to the goal state using A* Search. Also report the solution cost. The straight line distance heuristic estimates for the nodes are as follows: h(1) = 14, h(2) = 10, h(3) = 8, h(4) = 12, h(5) = 10, h(6) = 10, h(S) = 15. [10]

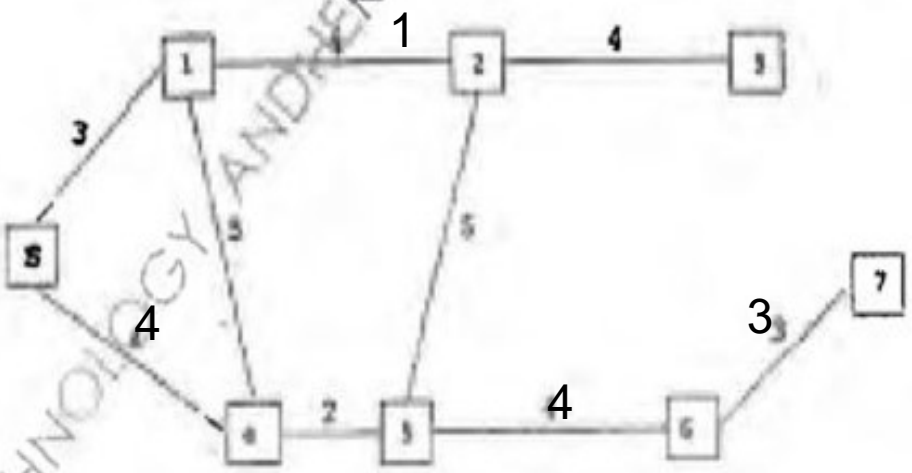
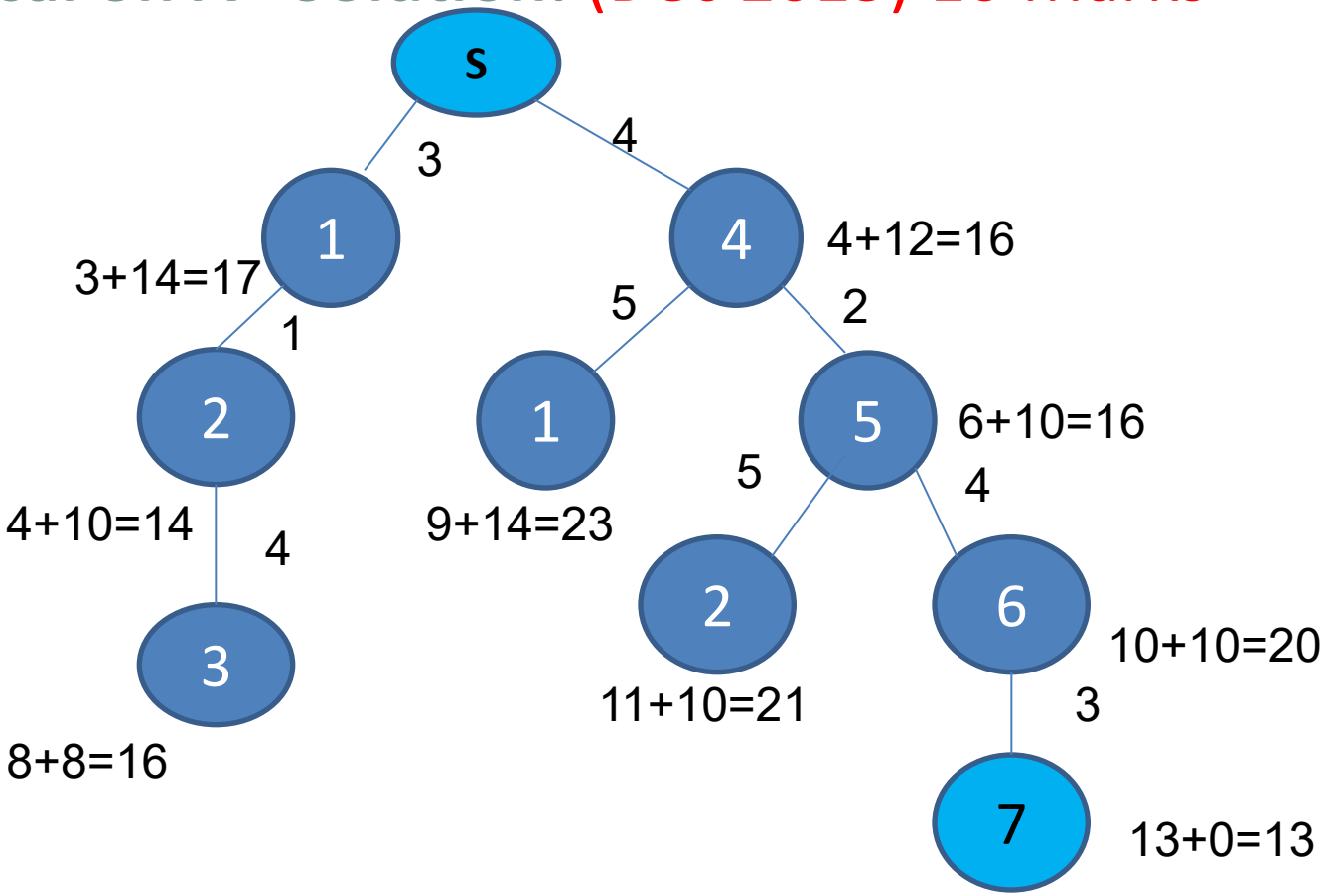


Figure 1.

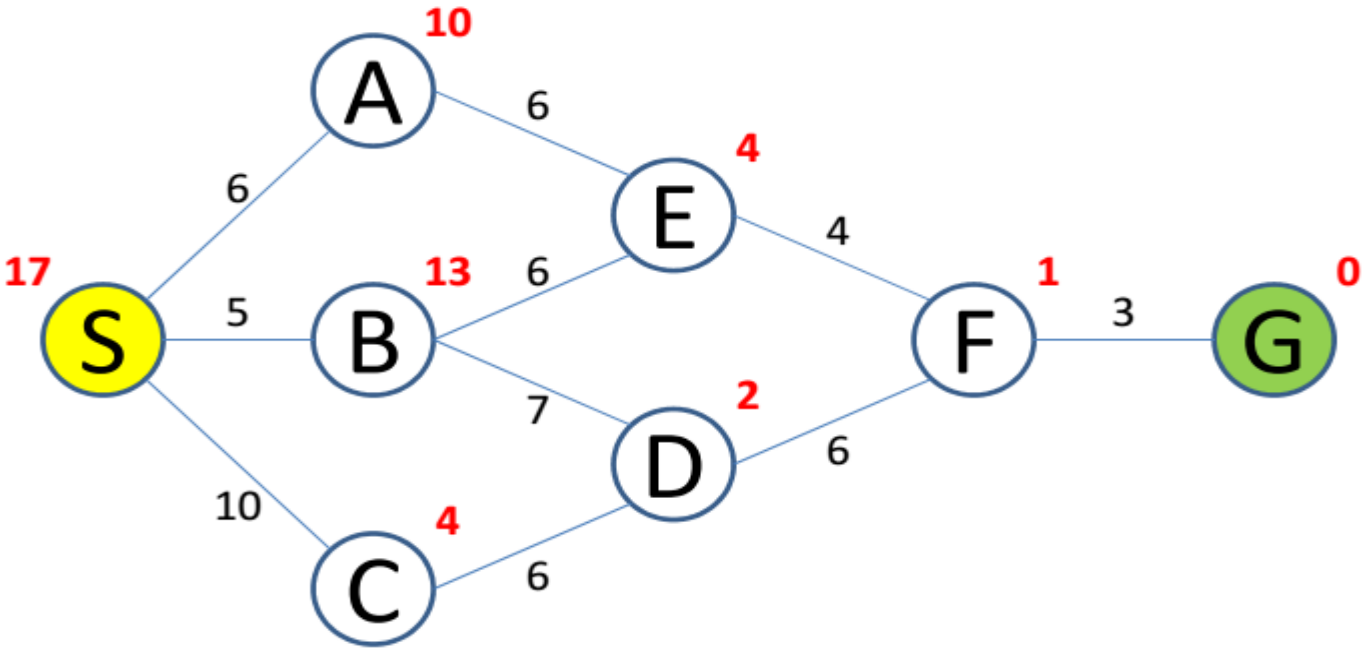
Numerical on A* Solution: (Dec 2015) 10 Marks



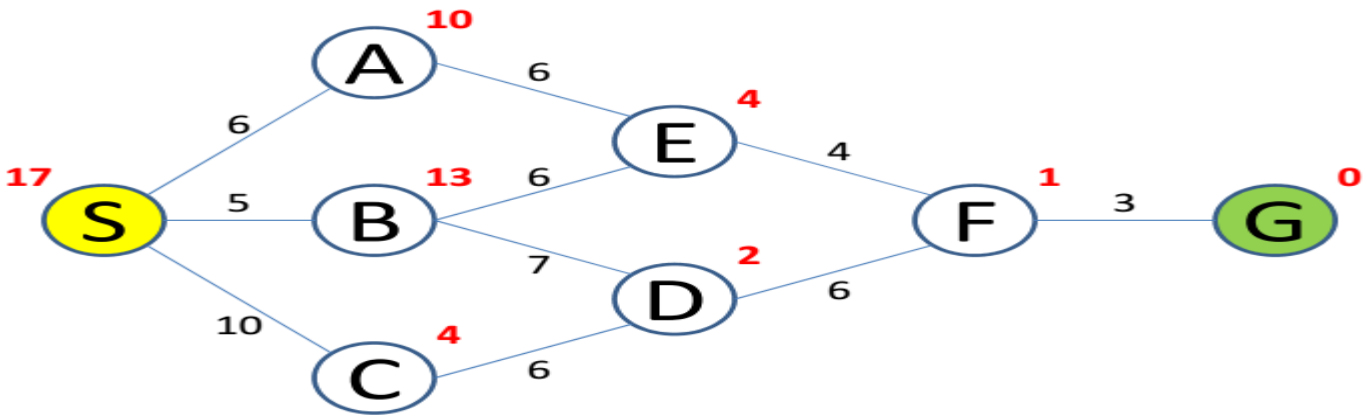
As its an Informed search Initial to Goal the Path is : **S-4-5-6-7**
Path Cost is 13

Numeric to solve on A*

For the given graph below 'S' is the start state and 'G' is the goal state. h(S)=17, h(A)=10, h(B)=13, h(C)=4, h(D)=2, h(E)=4, h(F)=1,h(G)=0. find path and path cost using A* algorithm.



For the given graph below 'S' is the start state and 'G' is the goal state. h(S)=17, h(A)=10, h(B)=13, h(C)=4, h(D)=2, h(E)=4, h(F)=1,h(G)=0. find path and path cost using A* algorithm.



Open: S(17), A(16),B(18), C(14), D(18), E(16), F(17), G(19),E(15), D(14), F(16), G(18)
Closed 1: S,C, A, E, F,D, B
Closed 2:S,C, A, B, D, E, F, G

SBEFG-18

MU Asked Questions based on Admissibility of A*

1. Give and Explain A* Algorithm. Drawbacks of A* and Show that A* is optimally efficient? (DEC 2010, DEC 2011, JUNE 2013)
2. What do you mean by admissible heuristic search function, explain with example? (DEC 2012)
3. Prove that A* is Admissible if it uses a monotone Heuristic? (May 2016, Dec 2016)

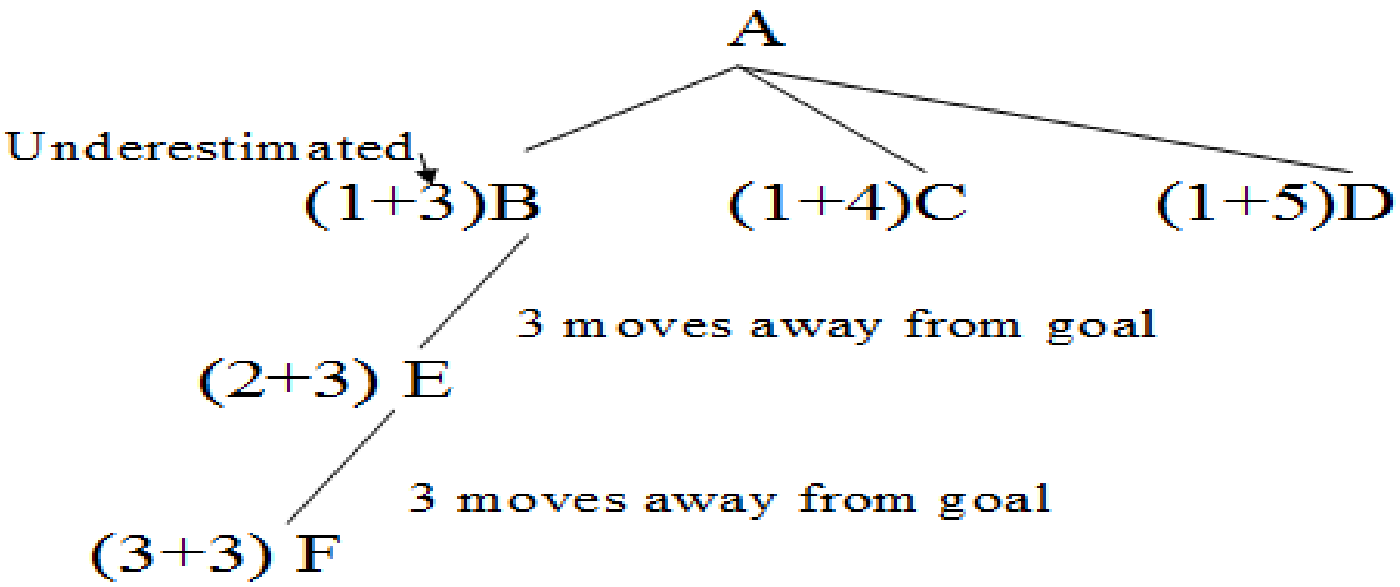
Admissibility of A* Algorithm

- **Property of A* Algorithm:**
  1. h is **admissible** when h(n) <= h*(n) holds. Using an admissible heuristic guarantees that the *first solution found will be an optimal one*. Therefore, A* is **admissible**.
  2. A* is **complete** whenever the branching factor is finite.

3.Behavior of A*- Underestimation

Example – Underestimation – f=g+h

Here h is underestimated

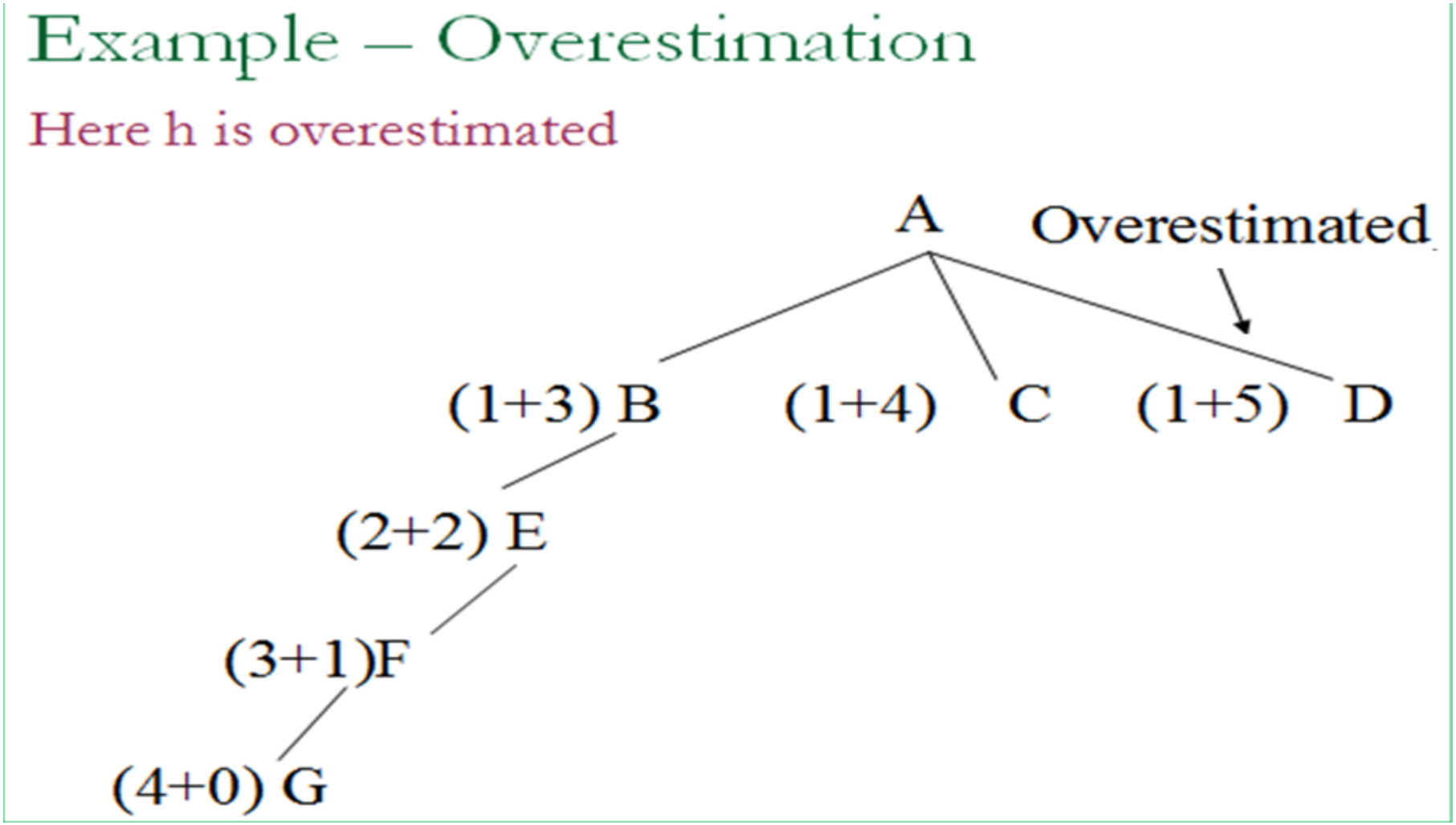


If we can guarantee that **h** never over estimates actual value from current to goal, then A* algorithm is guaranteed to find an optimal path to a goal, if one exists.

Explanation -Example of Underestimation

- Assume the cost of all arcs to be 1. A is expanded to B, C and D.
- 'f' values for each node is computed.
- B is chosen to be expanded to E.
- We notice that f(E) = f(C) = 5
- Suppose we resolve in favor of E, the path currently we are expanding. E is expanded to F.
- Clearly expansion of a node F is stopped as f(F)=6 and so we will now expand C.
- Thus we see that by underestimating h(B), we have wasted some effort but eventually discovered that B was farther away than we thought.
- Then we go back and try another path, and will find optimal path.

3.Behavior of A*- Overestimation



Explanation –Example of Overestimation

- A is expanded to B, C and D.
- Now B is expanded to E, E to F and F to G for a solution path of length 4.
- Consider a scenario when there a direct path from D to G with a solution giving a path of length 2.
- We will never find it because of overestimating h(D).
- Thus, we may find some other worse solution without ever expanding D.
- So by overestimating h, we can not be guaranteed to find the cheaper path solution.

3. Admissibility of A*

- A search algorithm is **admissible,** if
  - for any graph, it always terminates in an **optimal path** from initial state to goal state, if path exists.
- If heuristic function **h** is **underestimate** of actual value from current state to goal state, then it is called **admissible function**. **(h(n)<=h*(n))**
- Alternatively we can say that A* always terminates with the optimal path in case
  - h(x) is an **admissible heuristic function**.

3. Monotonicity of A*

Prove that A* is Admissible if it uses a monotone Heuristic? (May 2016, Dec 2016)

- A heuristic function **h** is monotone if

  – ∀ **states Xi and Xj such that Xj is successor of Xi**

  h(Xi)– h(Xj) ≤ cost (Xi, Xj) where, cost (Xi, Xj) actual cost of going from Xi to Xj

Example: h(Xi): Arad to Bucharest=366

h(Xj): Sibiu to Bucharest=253

h(Xi)– h(Xj):366-253=113

cost (Xi, Xj): Arad to Sibiu=140

⑩h (Goal) = 0

- In this case, heuristic is locally admissible i.e., *consistently finds the minimal path to each state they encounter in the search*.

Contd..

- Alternatively, the monotone property says:
  - that search space which is every where locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors.
- With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter.
- Each monotonic heuristic is admissible
  - A **cost function** f(n) is monotone. if f(n) ≤f(succ(n)), ∀n.
- For any admissible cost function f, we can construct a monotone admissible function.

3. A* Weaknesses

(DEC 2010, DEC 2011, JUNE 2013)

1. Still requires the use of a “natural” distance measure
2. Sorting takes time
3. Removing paths takes time

Heuristic Function for 8 Puzzle Problem

Example: Solve Eight puzzle problem using A* algorithm

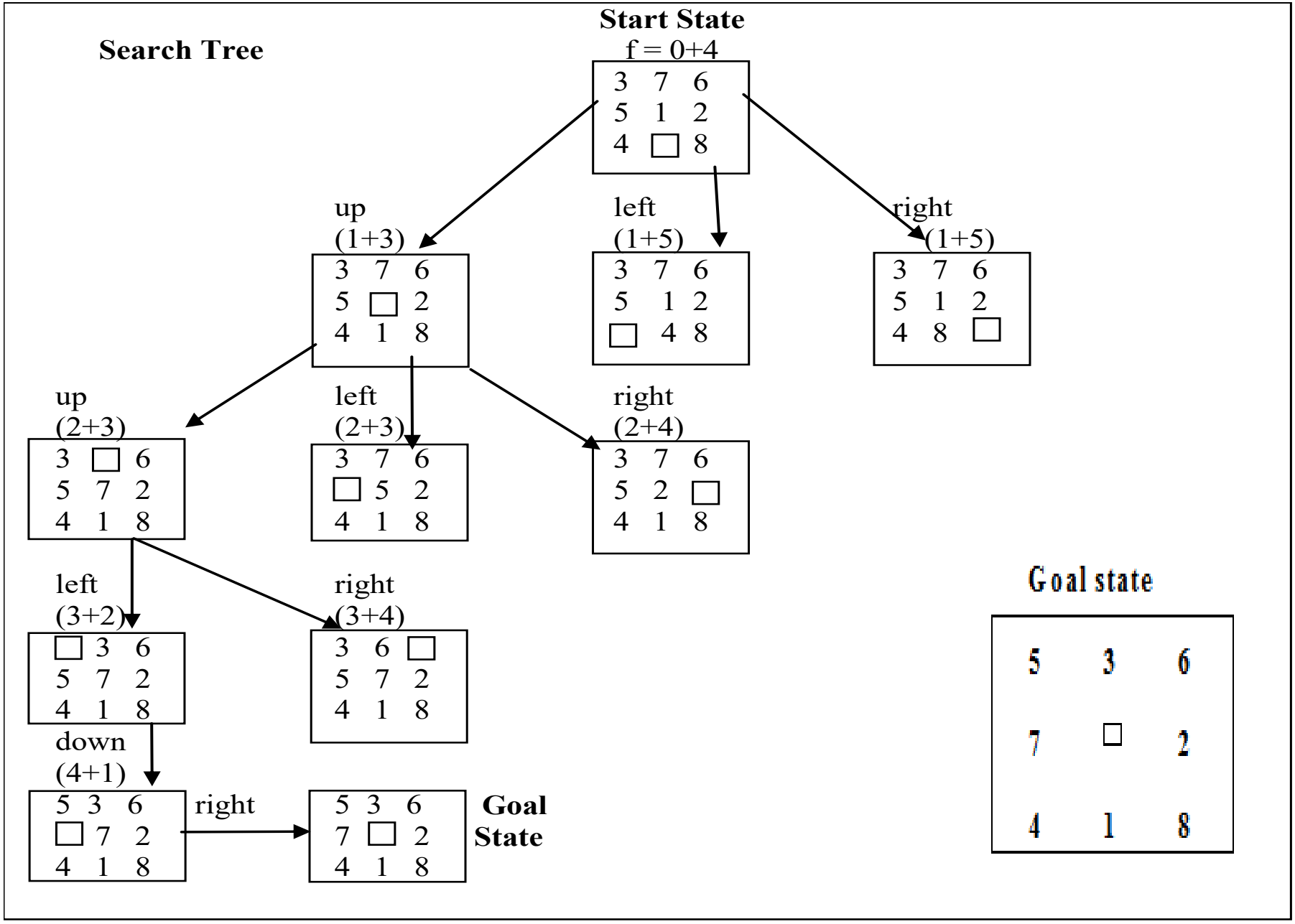
Start state

3	7	6
5	1	2
4	□	8

Goal state

5	3	6
7	□	2
4	1	8

- Evaluation function f (X) = g (X) + h(X)
  h (X) = the number of tiles not in their goal position in a given state X
  g(X) = depth of node X in the search tree
- Initial node has **f(initial_node)** = 4
- Apply A* algorithm to solve it.
- The choice of evaluation function critically determines search results.


Search Tree
Start State
f = 0+4
3 7 6
5 1 2
4 □ 8
up
(1+3)
3 7 6
5 □ 2
4 1 8
left
(1+5)
3 7 6
5 1 2
□ 4 8
right
(1+5)
3 7 6
5 1 2
4 8 □
up
(2+3)
3 □ 6
5 7 2
4 1 8
left
(2+3)
3 7 6
□ 5 2
4 1 8
right
(2+4)
3 7 6
5 2 □
4 1 8
left
(3+2)
□ 3 6
5 7 2
4 1 8
right
(3+4)
3 6 □
5 7 2
4 1 8
down
(4+1)
5 3 6
□ 7 2
4 1 8
right
5 3 6
7 □ 2
4 1 8
Goal
State
Goal state
5 3 6
7 □ 2
4 1 8

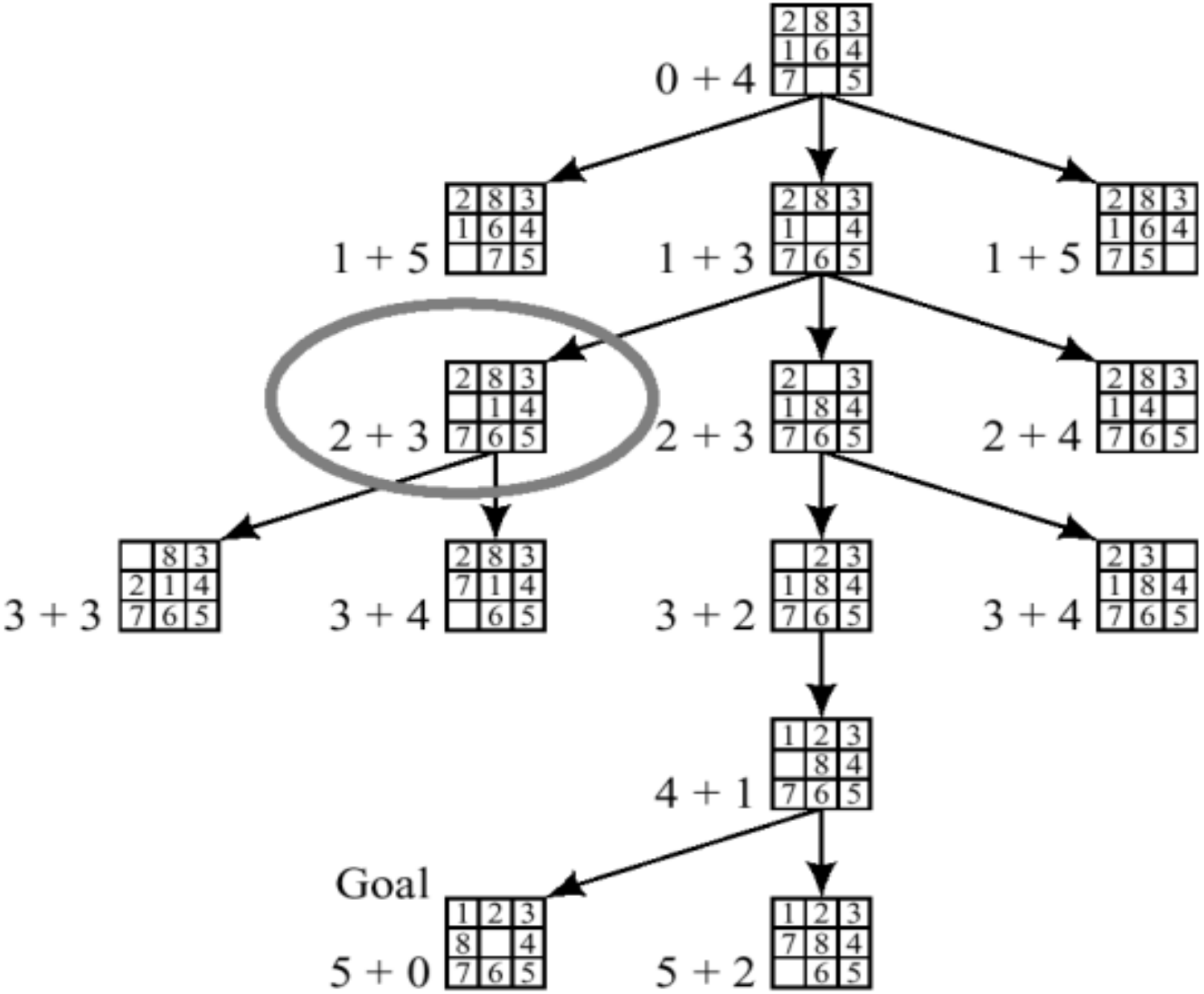
Solve Eight puzzle problem using A* algorithm

Initial State

2	8	3
1	6	4
7	□	5

Goal State

1	2	3
8	□	4
7	6	5


0 + 4
1 + 5
1 + 3
1 + 5
2 + 3
2 + 3
2 + 4
3 + 3
3 + 4
3 + 2
3 + 4
4 + 1
Goal
5 + 0
5 + 2

Harder Problem- **Solved by Manhattan distance**

- Harder problems can't be solved by heuristic function defined earlier.

Initial State

2	1	6
4	□	8
7	5	3

Goal State

1	2	3
8	□	4
7	6	5

- A better estimate function is to be thought.

  h (X) = the sum of the distances of the tiles from their goal position in a given state X

- Initial node has **h(initial_node)** = 1+1+2+2+1+3+0+2=12

LOCAL SEARCH ALGORITM

Local Search Algorithm

- In many optimization problems the path to a goal state is irrelevant. The goal state itself is the solution.
- Example:
  - Finding a configuration satisfying certain constraints,
    - e.g., the 8-queens problem or a job-shop scheduling problem. In such cases, we can use iterative improvement: start with a single current state, and try to improve it! The same framework is applicable to problems where the path appears to be of interest (e.g., TSP) if these problems can be casted in a more appropriate (but equivalent) way

Local Search Algorithm

- **Local search algorithms work as follows:**
  - **Step 1:** Pick a "solution" from the search space and evaluate it. Define this as the current solution.
  - **Step 2:** Apply a transformation to the current solution to generate and evaluate a new solution.
  - **Step 3:** If the new solution is better than the current solution then exchange it with the current solution; otherwise discard the new solution.
  - **Step 4:** Repeat steps 2 and 3 until no transformation in the given set improves the current solution.

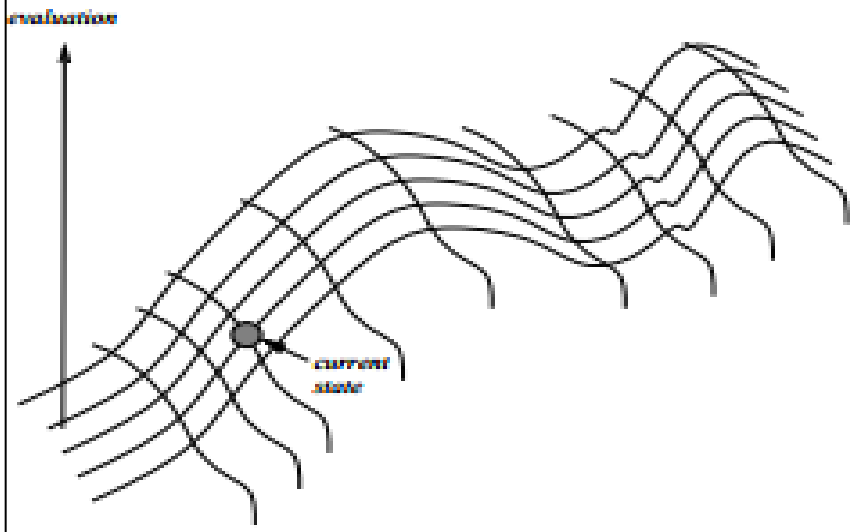
Local Search Algorithm

- Thus local search algorithms operate using a single current state (rather than multiple paths as e.g., A∗) and generally move only to neighbours of that state.
- At each step of a local search algorithm we have a complete but imperfect solution to a search problem.
- Other algorithms we saw previously (e.g., A∗) work with partial solutions and extend them to complete ones.
- **Good properties of local search algorithms:**
  - Constant space
  - Suitable for on-line as well as off-line problems.
  - Can find reasonable solutions in large solution spaces where exhaustive search would fail miserably

Local Search Algorithm

Iterative Improvement Algorithms

Idea: Start with a "solution" and make modifications until you reach a solution. Graphically:

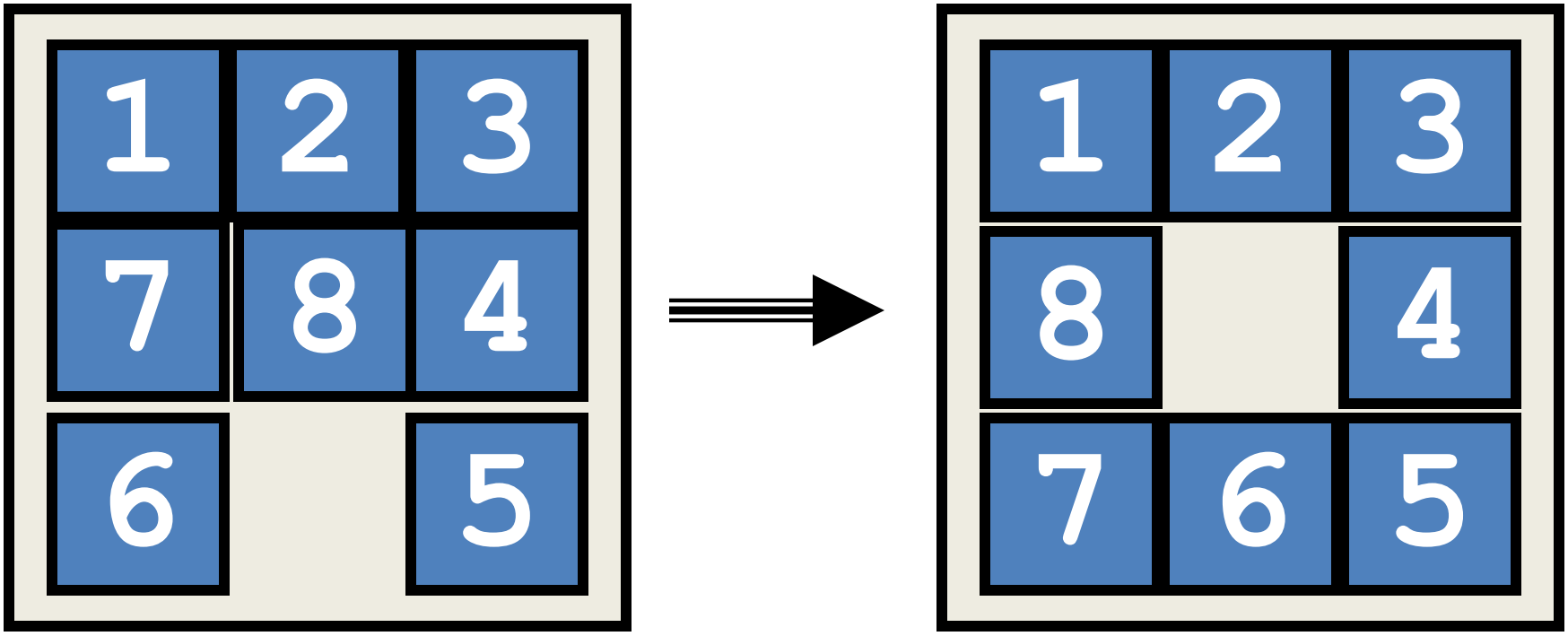


1. Hill Climbing Algorithm

- ***Step 1 :*** *Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.*
- ***Step 2 :*** *Loop until the solution state is found or there are no new operators present which can be applied to current state.*
- *a) Select a state that has not been yet applied to the current state and apply it to produce a new state.*
- *b) Perform these to evaluate new state*
  - *If the current state is a goal state, then stop and return success.*
  - *If it is better than the current state, then make it current state and proceed further.*
  - *If it is not better than the current state, then continue in the loop until a solution is found.*

Step 3 : *Exit.*

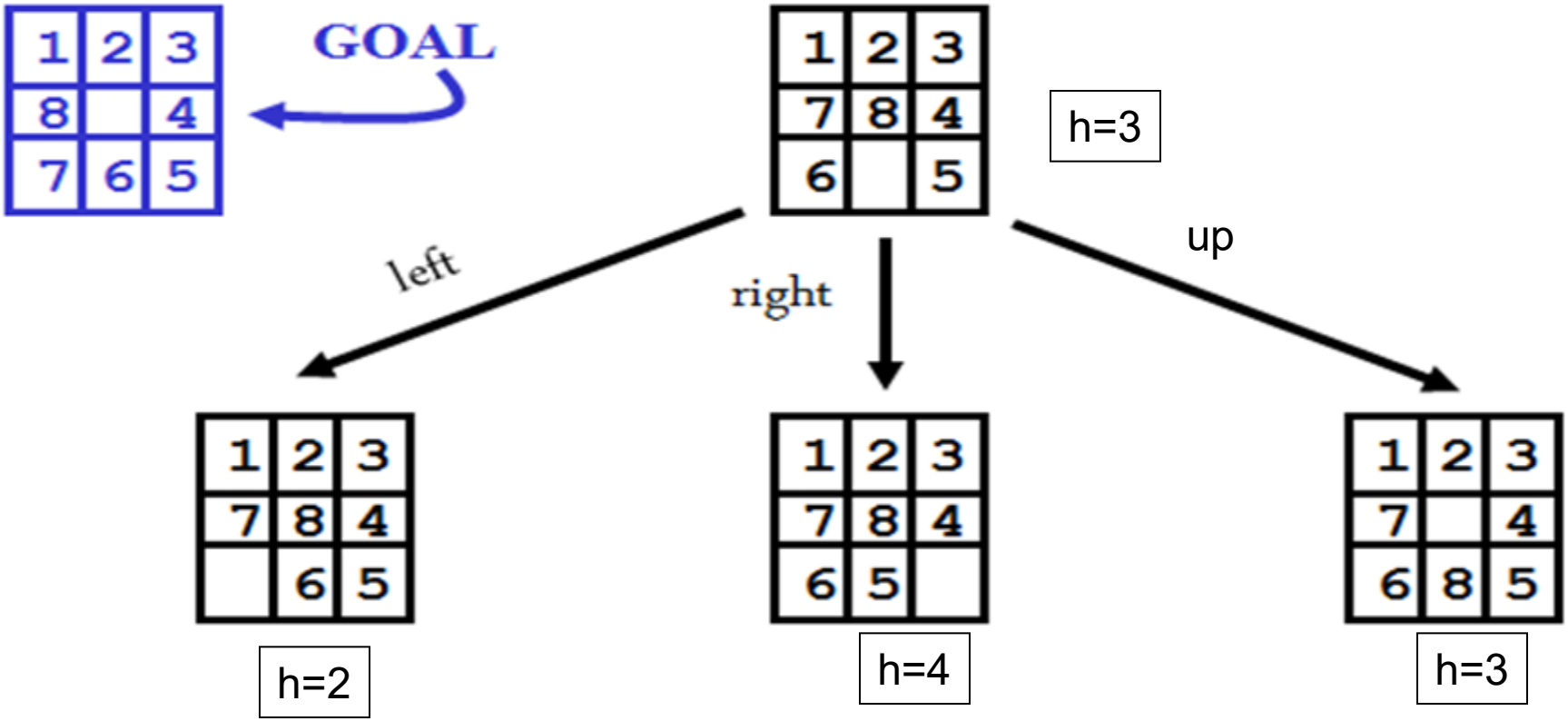
Example: 8 Puzzle



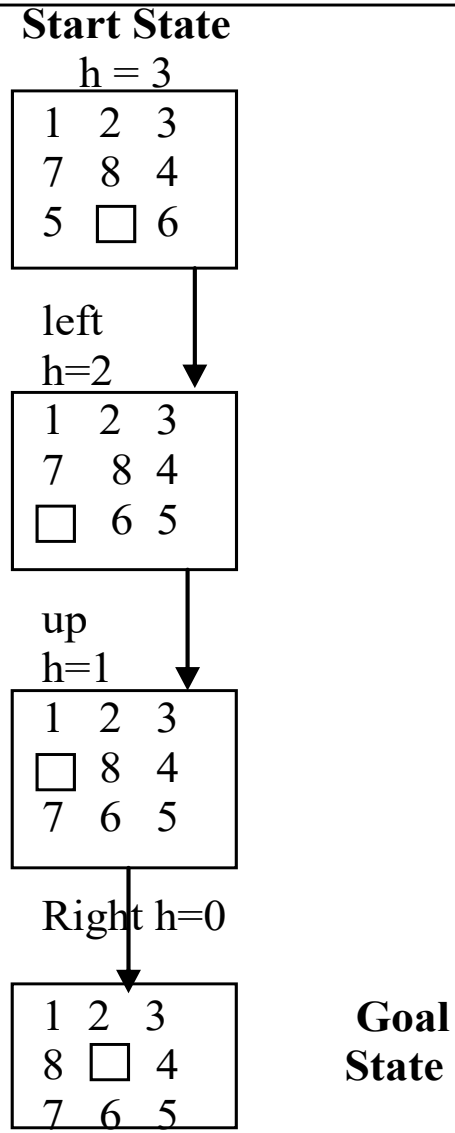
Initial State

Goal State

8 Puzzle with Hill climbing example



Search Tree


Start State
h = 3
1 2 3
7 8 4
5 □ 6
left
h=2
1 2 3
7 8 4
□ 6 5
up
h=1
1 2 3
□ 8 4
7 6 5
Right h=0
1 2 3
8 □ 4
7 6 5
Goal
State

Solve Eight puzzle problem using Hill Climbing algorithm

Initial State

2	8	3
1	6	4
7	□	5

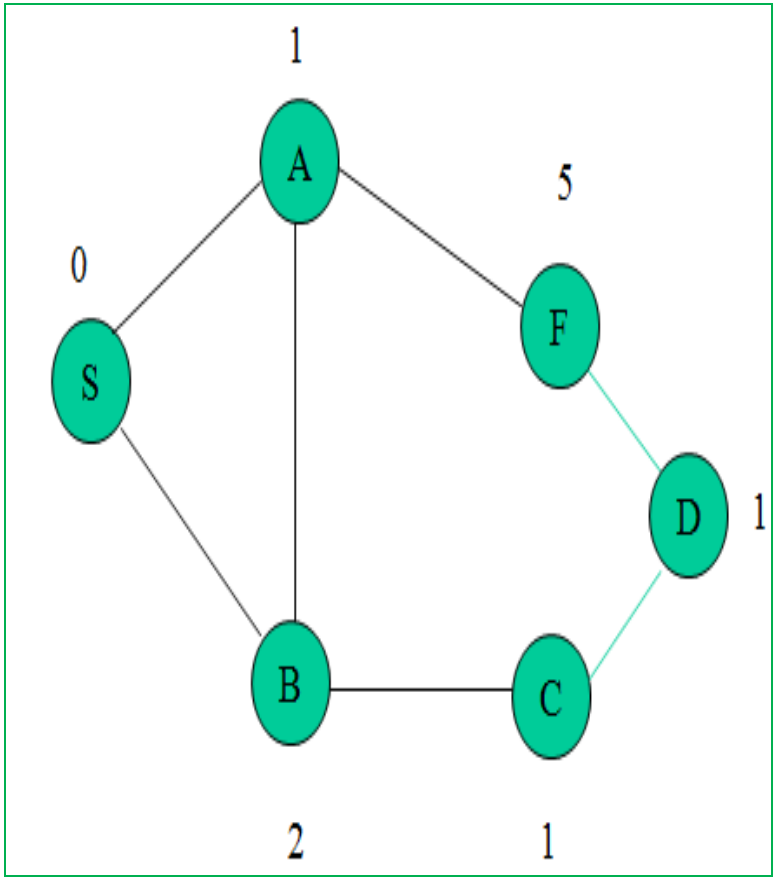
Goal State

1	2	3
8	□	4
7	6	5

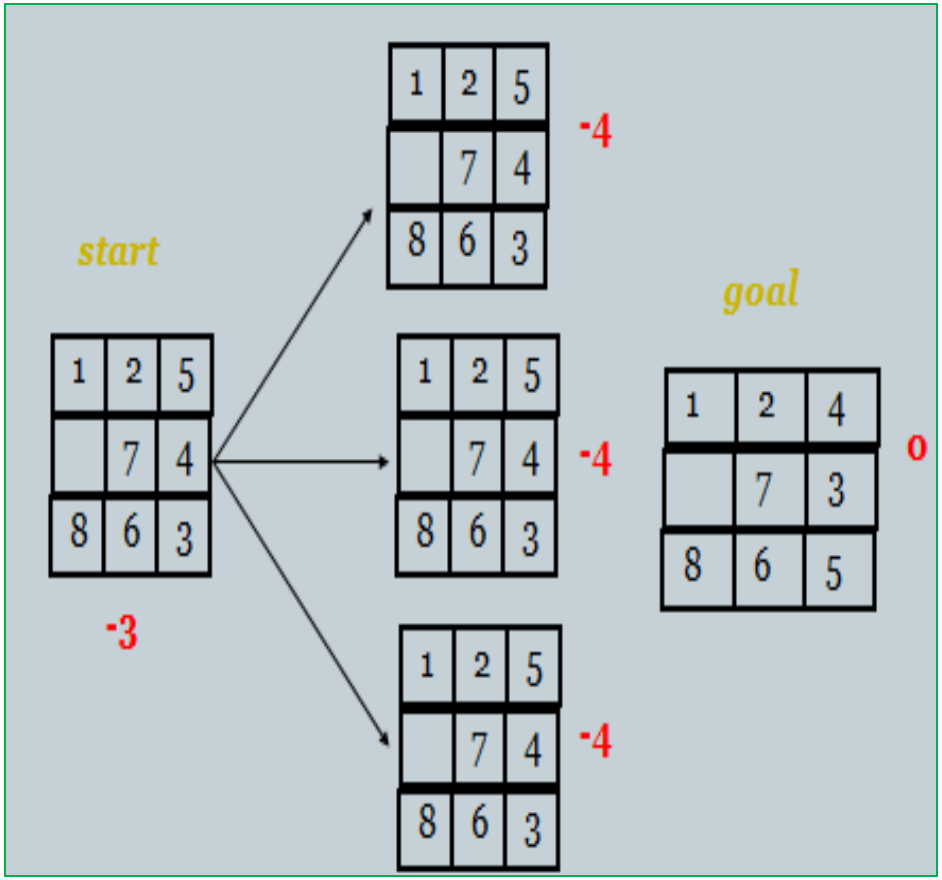
Problems in Hill Climbing

1. **Local maximum:** It is a state that is better than all its neighbors but is not better than some other states farther away. All moves appear to be worse.
   - *Solution to this is to backtrack to some earlier state and try going in different direction.*
2. **Plateau:** It is a flat area of the search space in which, a whole set of neighboring states have the same value. It is not possible to determine the best direction.
   - *Here make a big jump to some direction and try to get to new section of the search space.*
3. **Ridge:** It is an area of search space that is higher than surrounding areas, but that can not be traversed by single moves in any one direction. (Special kind of local maxima).
   - *Here apply two or more rules before doing the test i.e., moving in several directions at once.*

Problems in Hill Climbing



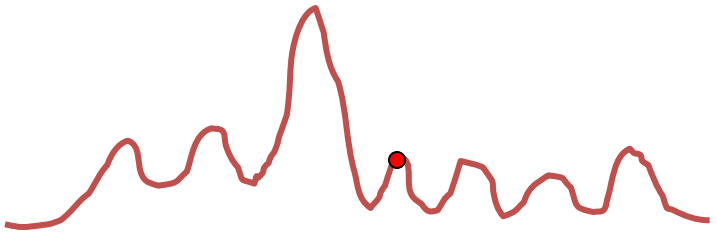
Example of a Local Maximum



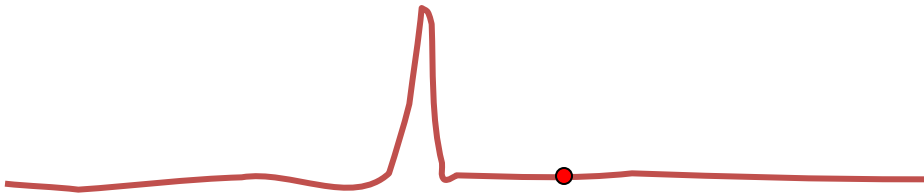
Example of a Plateau

Hill Climbing Problems

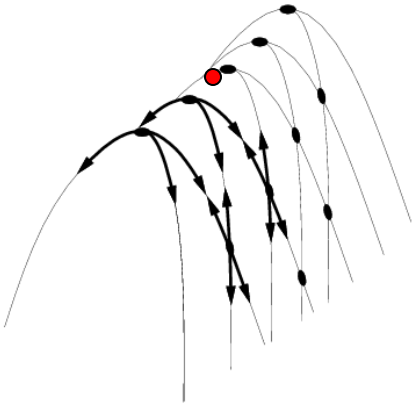
Local maxima



Plateaus



Ridges



Hill-Climbing variations

1. Stochastic hill-climbing
   - Reaches local maxima still goes down the hill assuming to get a solution better than the local maxima and traverse down hill.
2. First-choice hill-climbing
   - The moment you get a local maxima assuming it to be the solution and stop the search. (unlike Stochastic which still search for Global Maxima)
   - Useful when there are a very large number of successors
3. Random-restart hill-climbing
   - Tries to avoid getting stuck in local maxima, so restart from any point hoping to reach to the global maxima.

MU Asked Questions

1. What are the problems /frustrations that occur in Hill Climbing Technique. Illustrate with an example. (6 Marks, Dec 2016)
2. Explain Hill Climbing Architecture with an example. (5 Marks, May 2016)
3. Explain Hill Climbing Algorithm? Give its limitations? (JUNE 2013)

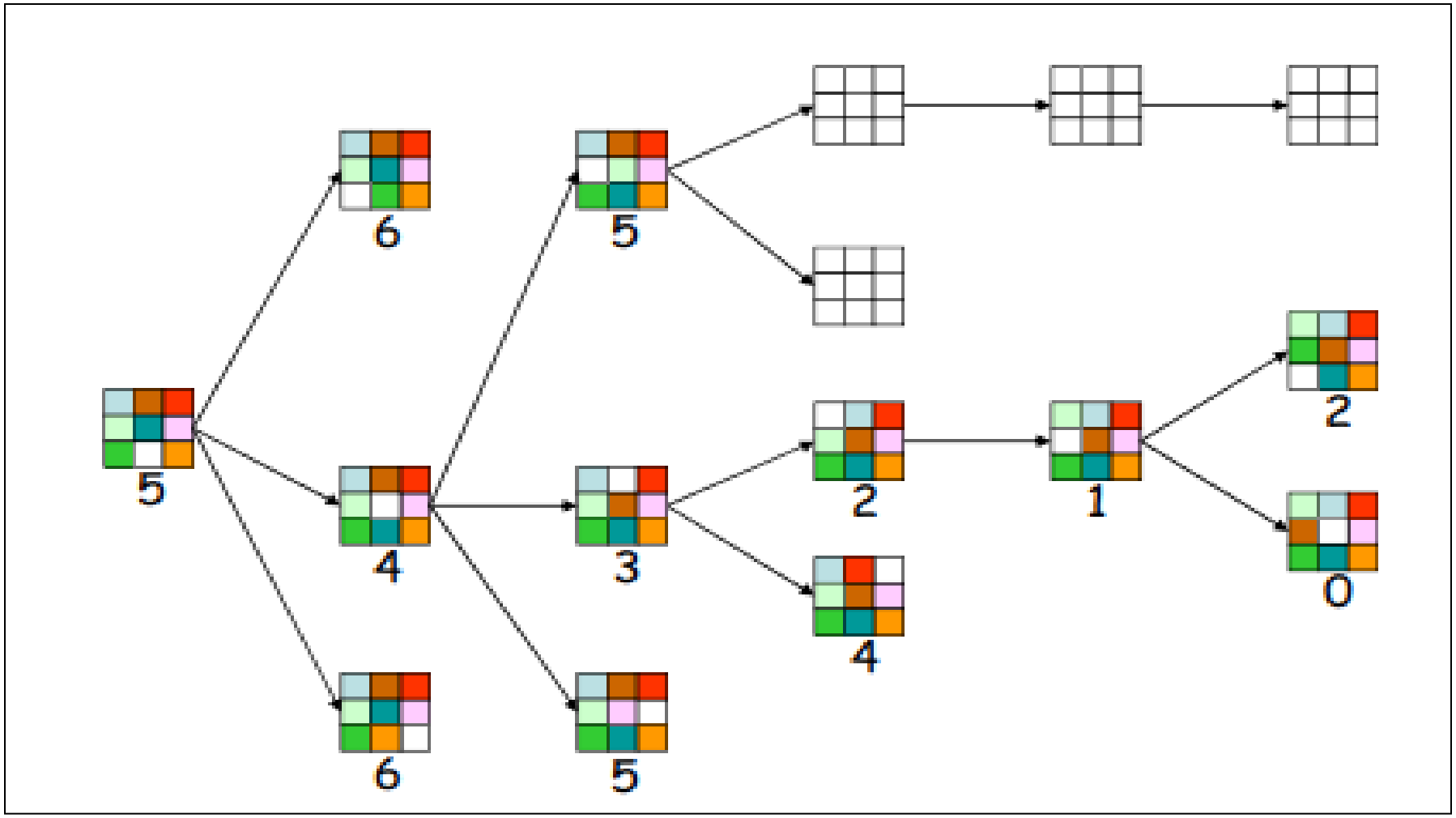
2. Steepest-Ascent Hill Climbing-(Gradient Search)

- Considers all the moves from the current state.
- Selects the best one as the next state.

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
   - SUCC = a state such that any possible successor of the current state will be better than SUCC (the worst state).
   - For each operator that applies to the current state, evaluate the new state:
     goal → quit
     better than SUCC → set SUCC to this state
   - SUCC is better than the current state → set the new current state to SUCC.

2.Steepest-Ascent Hill Climbing



Solve Eight puzzle problem using Steepest-Ascent Hill Climbing algorithm

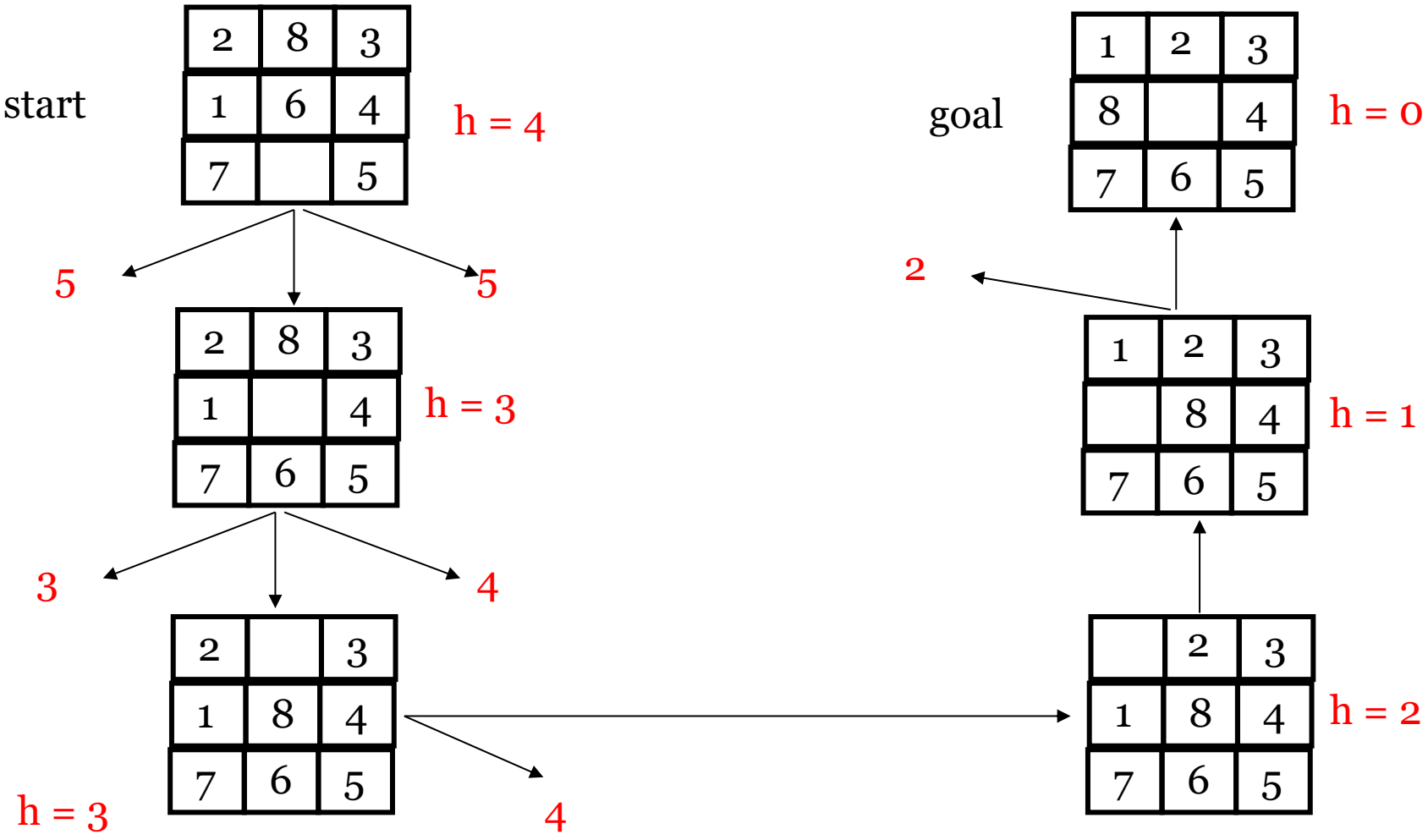
Initial State

2	8	3
1	6	4
7	□	5

Goal State

1	2	3
8	□	4
7	6	5

2. Steepest Ascent Hill climbing example



f(n) = (number of tiles out of place)

Solve Eight puzzle problem using Steepest-Ascent Hill Climbing algorithm

Initial State

1	2	3
7	8	4
6	□	5

Goal State

1	2	3
8	□	4
7	6	5

Difference between Simple Hill Climbing and Steepest Ascent Hill Climbing Algorithm

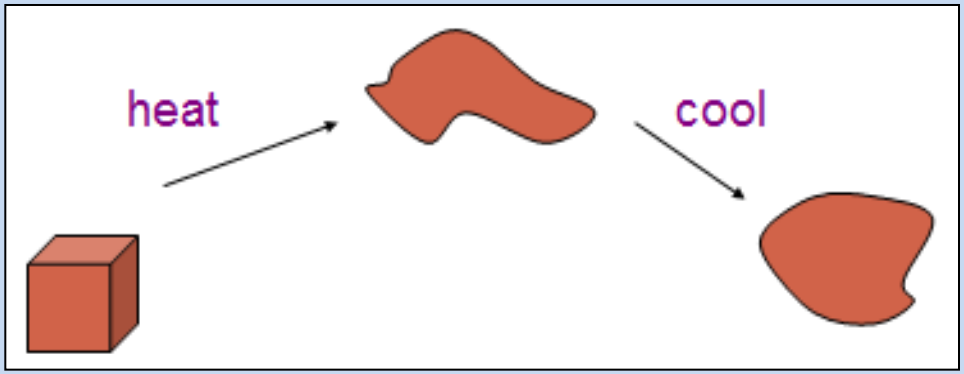
- Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.
- In **simple hill climbing**, the first closer node is chosen, whereas in **steepest ascent hill climbing** all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.

Another Local Search Algorithm- Simulated Annealing

- Motivated by the physical annealing process
- Material is heated and slowly cooled into a uniform structure
- Simulated annealing mimics this process
- The first SA algorithm was developed in 1953 (Metropolis)

Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

$$p = e^{(-\Delta E/kT)}$$

where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

- At the beginning, the temperature is high. As the temperature becomes lower
  - kT becomes lower, ΔE/kT gets bigger, (-ΔE/kT) gets smaller, e^(-ΔE/kT) gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

Simulated Annealing

- **Simulated annealing** is a process where the temperature is reduced slowly, starting from a random search at high temperature eventually becoming pure greedy descent as it approaches zero temperature.

- Compared to hill climbing which *gets stuck in Local maxima*, the main difference is that SA allows ***downwards steps*** *to avoid local maxima*

- **Simulated Annealing** also differs from hill climbing in that a move is selected at random and then decides whether to accept it

- In SA better moves are always accepted. Worse moves are not

Simulated Annealing Algorithm

- *current* ← start node;
- for each T on the schedule /* need a schedule */
  - *next* ← randomly selected successor of *current*
  - evaluate next; it it's a goal, return it
  - ΔE ← *next*.Value – *current*.Value /* already negated */
  - if $\Delta E > 0$
    - then *current* ← *next* /* better than current */
    - else *current* ← *next* with probability $e^{(\Delta E/T)}$

Quiz on simulated annealing:

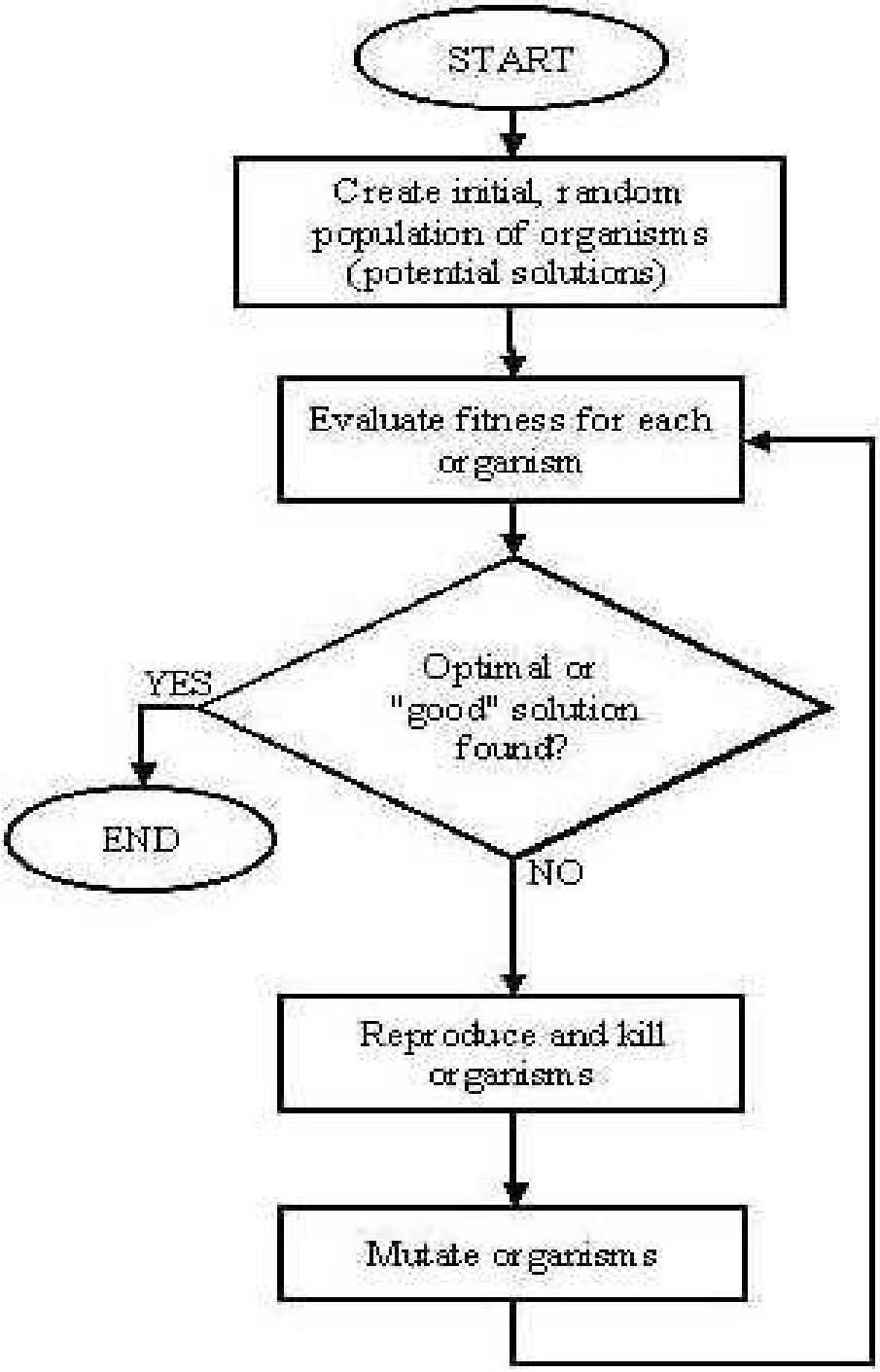
1. What is meant by simulated annealing in artificial intelligence?
   A. Returns an optimal solution when there is a proper cooling schedule
   B. Returns an optimal solution when there is no proper cooling schedule
   C. It will not return an optimal solution when there is a proper cooling schedule
   D. None of the mentioned

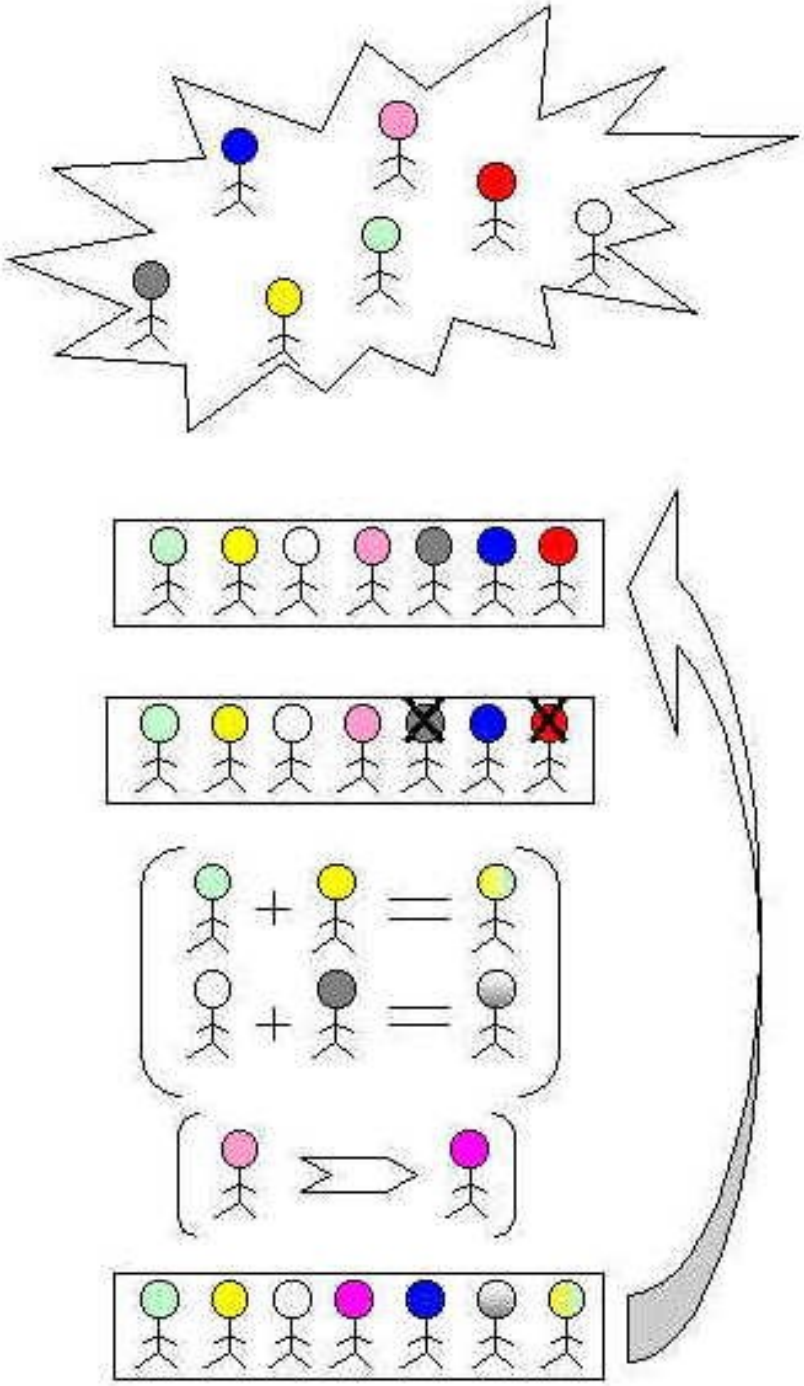
Local Search: Genetic Algorithm

Explain how genetic algorithm can be used to solve a problem by taking a suitable example? (May 2016, 10 marks)

What is Genetic Algorithm

- A **Genetic algorithm** (or **GA**) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
  - *The main principle of evolution used in GA is "survival of the fittest". The good solution survive, while bad ones die*
- Genetic algorithms are categorized as global search heuristics.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).


START
Create initial, random population of organisms (potential solutions)
Evaluate fitness for each organism
Optimal or "good" solution found?
YES
END
NO
Reproduce and kill organisms
Mutate organisms



Genetic Algorithms

= stochastic local beam search + generate successors from **pairs** of states

k states (individuals) - population

original population randomly generated

each individual represented as a string (chromosome)

each individual rated by an objective function (fitness function)

probability of being chosen for reproduction directly proportional to fitness

two parents produce offspring by crossover

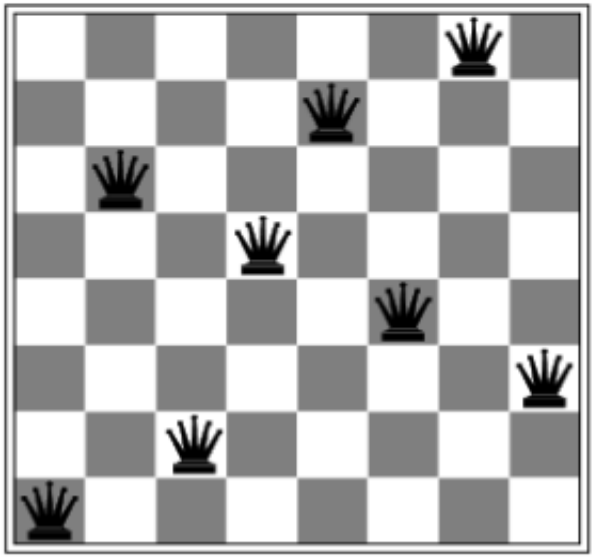
then with some small probablity, mutation (bits of the string changed)

Genetic Algorithms

Example: 8 queens problem states

States: assume each queen has its own column, represent a state by listing a row where the queen is in each column (digits 1 to 8)

for example, the state below will be represented as 16257483

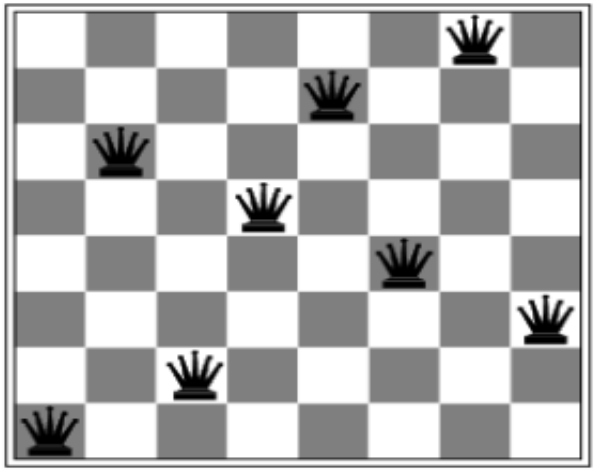


Genetic Algorithms

Example: 8 queens problem fitness

Fitness function: instead of $-h$ as before, use the number of **nonattacking** pairs of queens. There are 28 pairs of different queens, smaller column first, all together, so solutions have fitness 28. (Basically, fitness function is $28 - h$.)

for example, fitness of the state below is 27 (queens in columns 4 and 7 attack each other)



Genetic Algorithms

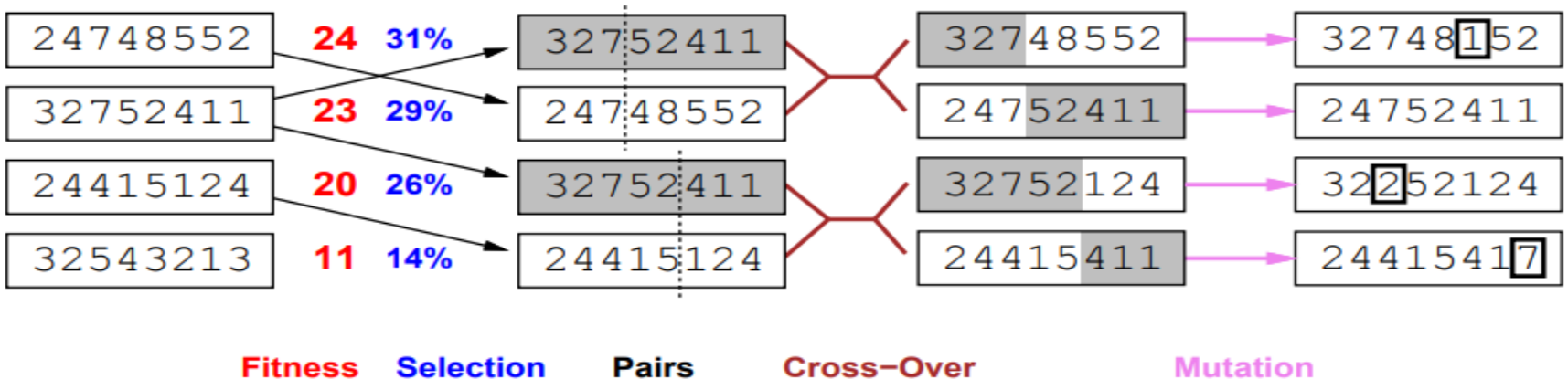
Example: 8 queens problem crossover

choose pairs for reproduction (so that those with higher fitness are more likely to be chosen, perhaps multiple times)

for each pair, choose a random crossover point between 1 and 8, say 3

produce offspring by taking substring 1-3 from the first parent and 4-8 from the second (and vice versa)

apply mutation (with small probablity) to the offspring



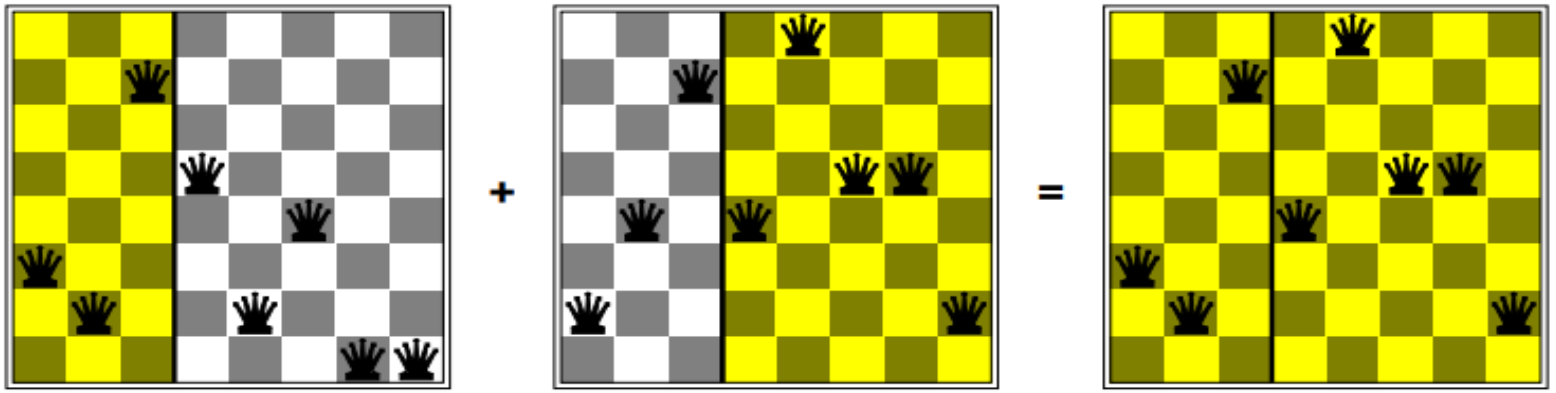
Genetic Algorithms

Importance of representation

Parts we swap in crossover should result in a well-formed solution (and in addition better be **meaningful**)

consider what would happen with binary representation (where position requires 3 digits)

also, chosen representation reduced search space considerably (compared to representing each square for example)



Genetic Algorithms

Genetic algorithm pseudocode

(this is one offspring per pair version, unlike in the example)

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
    inputs: population, a set of individuals
            FITNESS-FN, a function that measures the fitness of an individual
    repeat
        new-population ← empty set
        for i=1 to SIZE(population) do
            x ← RANDOM-SELECTION(population, FITNESS-FN)
            y ← RANDOM-SELECTION(population, FITNESS-FN)
            child ← REPRODUCE(x, y)
            if (small random probability) then child ← MUTATE(child)
            add child to new-population
    until some individual is fit enough or enough time has elapsed
    return the best individual in population, according to FITNESS-FN
```

Quiz on Genetic Algorithm:

1. How the new states are generated in genetic algorithm?
   a) Composition
   b) Mutation
   c) Cross-over
   d) Both Mutation & Cross-over
2. Genetic algorithms are used for minimization problems while simulated annealing is used for maximization problems?
3. Genetic algorithms maintain several possible solutions, whereas simulated annealing works with one solution.
4. Simulated annealing is guaranteed to produce the best solution, while genetic algorithms do not have such a guarantee

Summary of Search Algorithms

Uninformed and Informed

- **Search algorithms** like breadth-first, depth-first or A* explore all the search space systematically by keeping one or more paths in memory and by recording which alternatives have been explored.
- When a goal is found, the path to that goal constitutes a solution.

Local Search:

- **Local search algorithms** can be very helpful if we are interested in the solution state but not in the path to that goal.
- They operate only on the current state and move into neighboring states .
- They use very little memory

THANK YOU