Artificial Intelligence

Solving problems by searching I

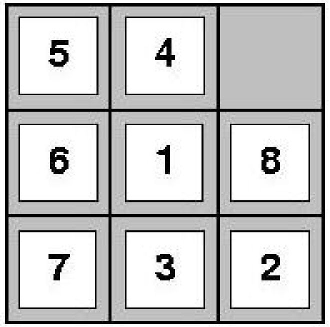
“Artificial Intelligence - A Modern Approach”, Chapters 3 and 4

POLITÉCNICO DE LEIRIA

Slides by Carlos Grilo

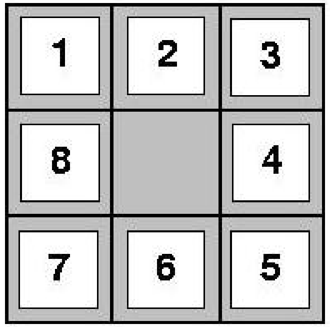
Problem examples

8-Puzzle



5	4	
6	1	8
7	3	2

Start State



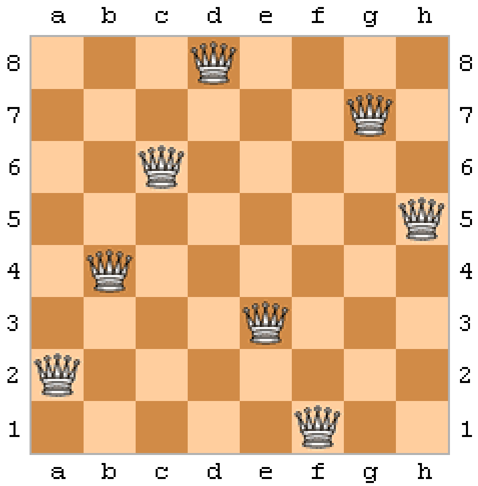
1	2	3
8		4
7	6	5

Goal State

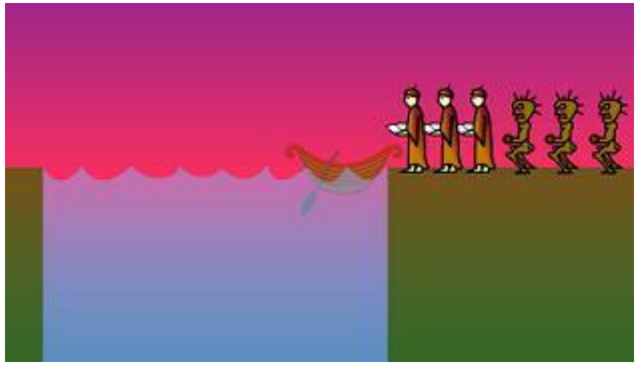
Cryptarithmetic

FORTY	29786
+ TEN	+ 850
+ TEN	+ 850
--------	--------
SIXTY	31586

N-queens

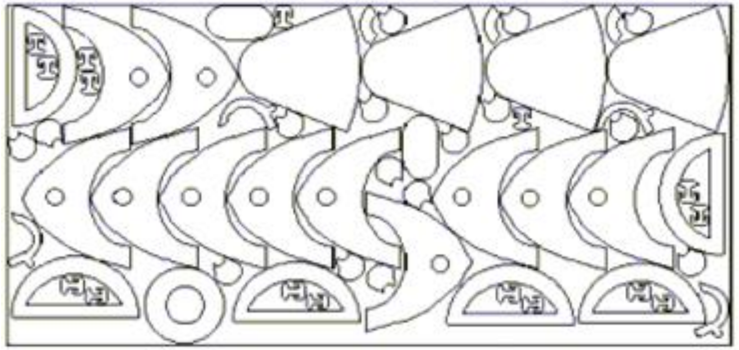


Missionaries and cannibals

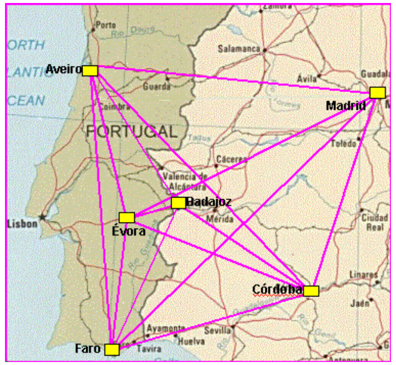


Problem examples

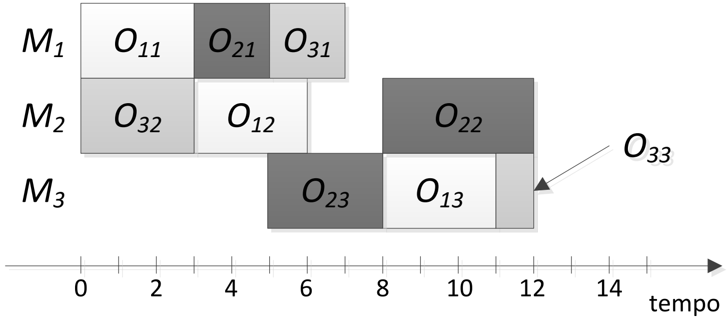
Space optimization



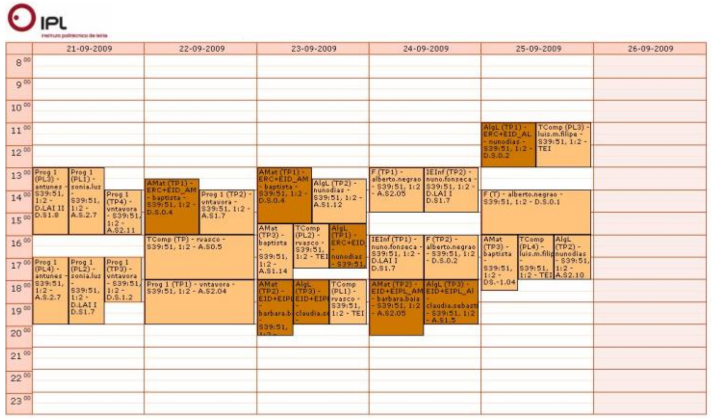
Travelling salesman



Job scheduling

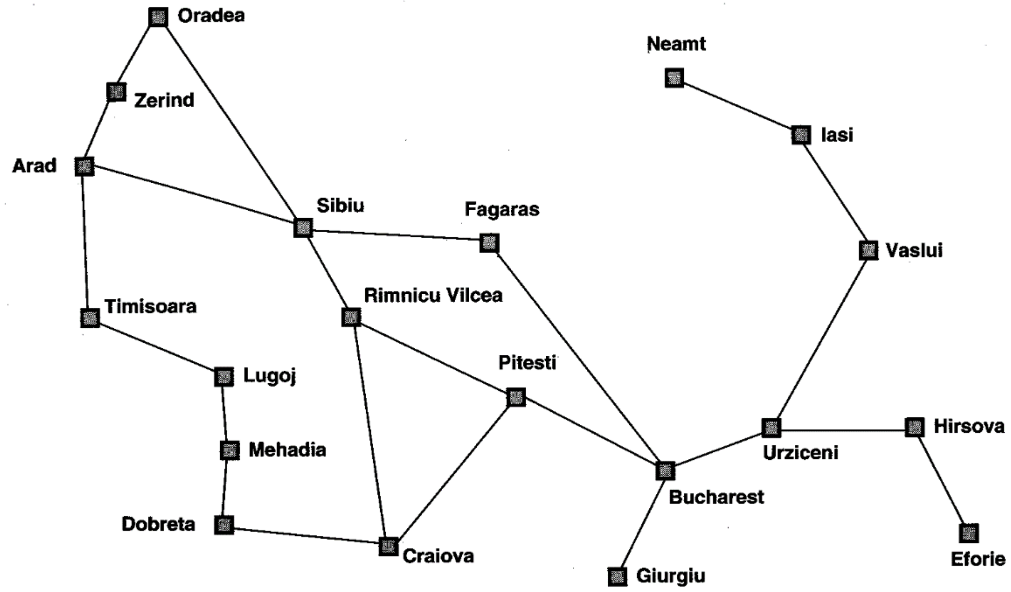


Timetables



Problem examples

Path finding



VLSI

Problems

- We will divide problems in two types:
  - Iterative improvement: Problems in which the information needed to reach a solution is contained in the state itself (ex: n-queens, timetables, VLSI)
  - Best first: Problems in which the agent departs from an initial environment state and wants to find an action sequence that leads it from that state to a goal state (ex: 8-puzzle, missionaries and cannibals, path finding)

Best First Problems

Problem resolution steps

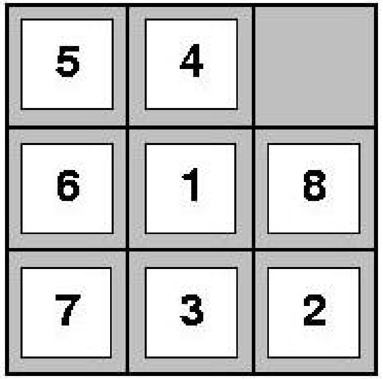
1. Goal formulation
   - The goal may be defined as a set of environment states or some abstract property
2. Problem formulation
   - Decide what actions and states to consider
3. Search
   - Process of searching the sequence of actions that allow the agent to reach the goal departing from the initial state
4. Execution

Problem formulation

- Initial state
- Actions(s) - Returns the set of actions that can be executed in state *s*
- Result(s, a) - Returns the state that results from doing action *a* in state *s*
- Goal_test(s) - Test that is applied in order to verify if some state *s* is a goal state
- Path_cost(p)
    - Assigns a numeric cost to the path *p* from one state to another
    - In general, the cost of a path can be described as the sum of the costs of the individual actions along the path

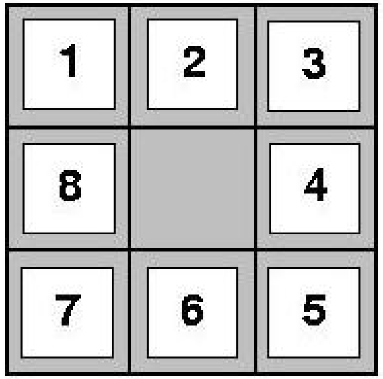
Problem formulation

- 8 Puzzle



5	4	
6	1	8
7	3	2

Start State



1	2	3
8		4
7	6	5

Goal State

- Initial state: particular configuration of the board
- Actions: 4 movements of the blanc space
- Goal test: goal configuration
- Path cost: number of steps in the path

State space

- Initial state + actions -> state space

- State space
    - Set of all states reachable from the initial state by any sequence of actions

The resolution of a problem can be viewed as the search in a state space

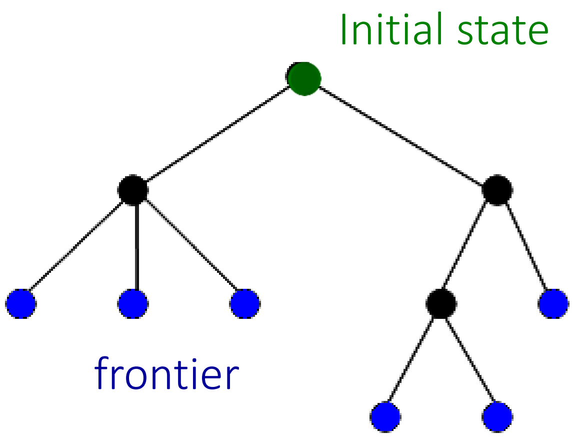
Measuring the agent’s performance

- The agent finds a solution?
- The solution found is good?
- What is the search cost (time and memory) to find a solution?
- Total cost = solution cost + search cost
- In general, the larger the search cost, the smaller the solution cost and vice-versa

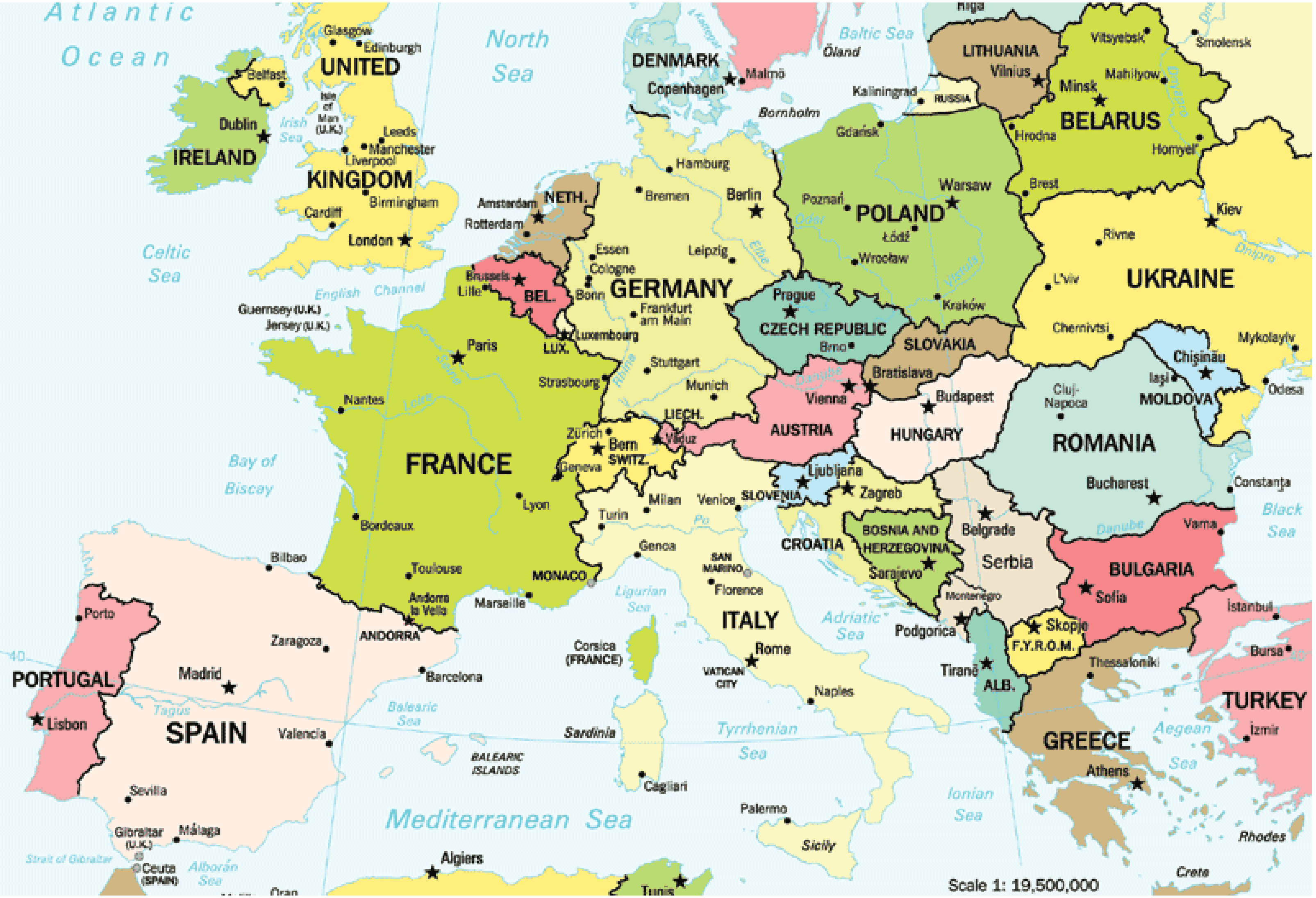
Search methods

- The search process builds a tree where the root is the initial state and the leaves are nodes (states) that were not expanded yet or that have no successors

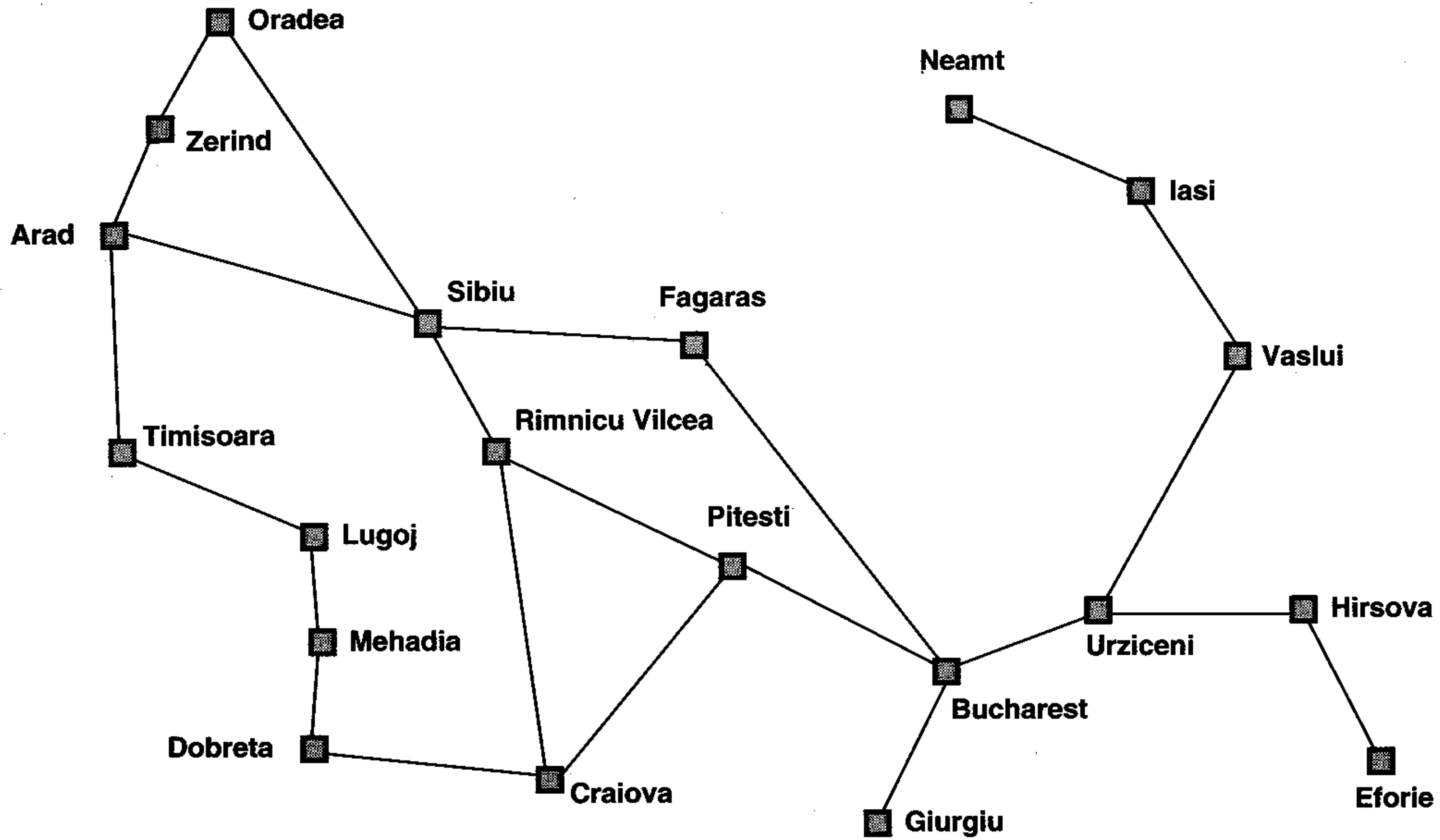
- The set of nodes that have not been expanded yet is called the frontier



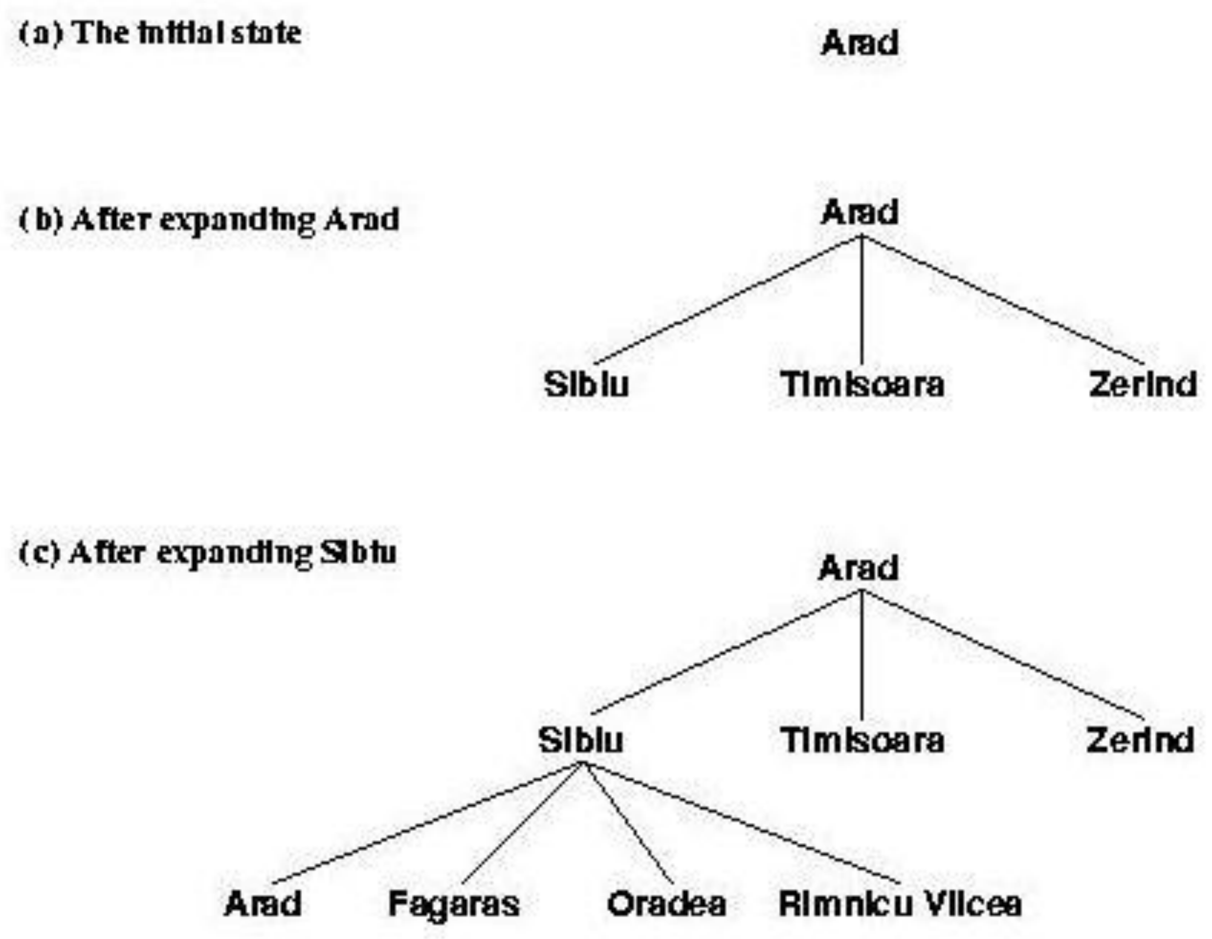
Europe’s map?!


Atlantic Ocean
Glasgow
Edinburgh
UNITED KINGDOM
Belfast
Dublin
IRELAND
Irish Sea
Isle of Man (U.K.)
Leeds
Manchester
Liverpool
Birmingham
Cardiff
London
North Sea
DENMARK
Copenhagen
Malmö
Kattegat
Baltic Sea
Öland
Bornholm
Riga
LITHUANIA
Vilnius
Kaliningrad
RUSSIA
Vitsyebsk
Smolensk
Mahilyow
Minsk
BELARUS
Hrodna
Homyel'
Dnieper
Gdańsk
Hamburg
Bremen
Berlin
Amsterdam
NETH.
Rotterdam
Essen
Cologne
Leipzig
Elbe
Oder
Poznań
Warsaw
POLAND
Łódź
Wrocław
Vistula
Brest
Kiev
Rivne
Dnipro
L'viv
UKRAINE
Kraków
Celtic Sea
English Channel
Guernsey (U.K.)
Jersey (U.K.)
Brussels
Lille
BEL.
Bonn
GERMANY
Frankfurt am Main
Prague
CZECH REPUBLIC
Brno
SLOVAKIA
Luxembourg
LUX.
Paris
Seine
Rhine
Stuttgart
Strasbourg
Munich
Danube
Vienna
Bratislava
Chernivtsi
Chişinău
Mykolayiv
Iaşi
MOLDOVA
Odesa
Cluj-Napoca
Budapest
HUNGARY
AUSTRIA
ROMANIA
Nantes
Loire
LIECH.
Vaduz
Zürich
Bern
SWITZ.
Geneva
FRANCE
Bay of Biscay
Lyon
Ljubljana
SLOVENIA
Zagreb
Bucharest
Constanța
Black Sea
Bordeaux
Milan
Venice
Turin
Po
CROATIA
BOSNIA AND HERZEGOVINA
Sarajevo
Belgrade
Serbia
Varna
BULGARIA
Sofia
Bilbao
Toulouse
MONACO
Genoa
SAN MARINO
Florence
Montenegro
Podgorica
Skopje
F.Y.R.O.M.
İstanbul
Porto
Andorra la Vella
ANDORRA
Marseille
Ligurian Sea
ITALY
Adriatic Sea
Zaragoza
Corsica (FRANCE)
Rome
VATICAN CITY
Tiranë
ALB.
Thessaloniki
Bursa
40
PORTUGAL
Madrid
Barcelona
Naples
TURKEY
Tagus
Lisbon
SPAIN
Balearic Sea
Valencia
Sardinia
Tyrrhenian Sea
GREECE
Aegean Sea
İzmir
BALEARIC ISLANDS
Cagliari
Athens
Ionian Sea
Sevilla
Mediterranean Sea
Palermo
Sicily
Rhodes
Gibraltar (U.K.)
Málaga
Strait of Gibraltar
Ceuta (SPAIN)
Alborán Sea
Algiers
Oran
Tunis
Crete
Scale 1: 19,500,000

Romenia’s simplified road map?!



Search methods



Each search method uses its own strategy to select the next node (state) to be expanded

We will soon see how this can be done

Search methods’ dont's

- Situations that search methods should avoid:
  - Generation of loops (in particular, return to the previous state)
  - Generation of redundant paths (e.g. if there are two paths to some state, only the better one should be generated)

General graph-search

function graph_search(*problem*) **returns** a solution, or failure

1. Initialize the ***frontier*** with node, n_0, containing the initial state of the problem

2. Initialize the ***explored set*** to be empty

3. **Loop do**

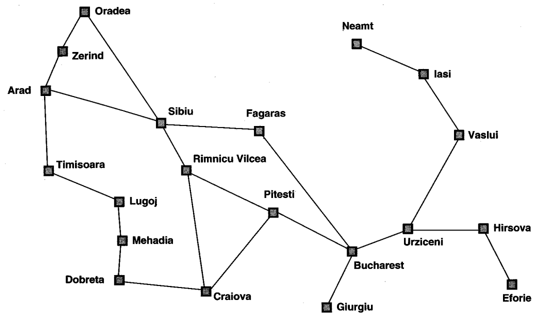
4. **If** the **frontier** is empty **then** return failure

5. Remove the first node, ***n***, from the ***frontier***

6. **If** ***n*** contains a goal state, **then** return the corresponding ***solution***; the solution is obtained traversing the path from ***n*** to n_0 (the arcs are created in step 8)

7. Add the state contained in ***n*** to the ***explored set***

8. Expand ***n*** and add its successors to the ***frontier*** according to the strategy of the search method, only if they are not already in the ***frontier*** or in the ***explored set.*** Create arcs between the added successors and ***n***



General graph-search

- As can be seen in the algorithm, the main difference between search methods is in the order by which they add the successors to the frontier, that is, the order by which nodes are expanded

- This is defined in step 8 when successors are added to the frontier (and also by the fact that the next node to be expanded is always the first node in the frontier)

Frontier

- The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy
- The next node to be explored is always popped form the head of the frontier (the first node)
- Most common data structures:
  - FIFO (queue): first-in, first-out
  - LIFO (stack): last-in, first-out
  - Priority queue: nodes are ordered according to some criterion

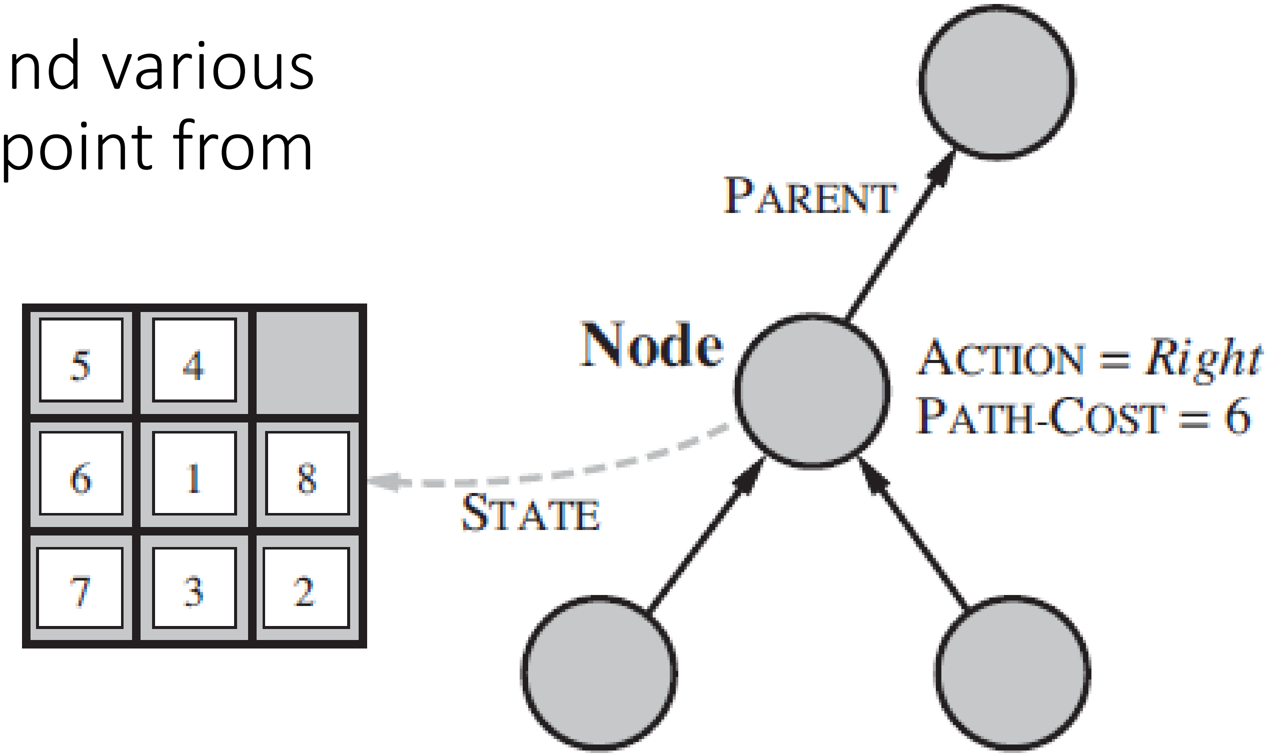
Node data structure

- Search algorithms require a data structure to keep track of the search tree that is being constructed

- For each node ***n*** of the tree, we have a structure that contains four components:
  - State: the state in the state space to which the node corresponds
  - Parent: the node in the search tree that generated this node
  - Action: the action that was applied to the parent to generate the node (in practice, we will save this in the state data structure)
  - Cost: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers

Node data structure

- Nodes are the data structures from which the search tree is constructed
- Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent



Measuring problem-solving performance

- Completeness: Is the algorithm guaranteed to find a solution when there is one?
- Optimality: Does the algorithm find the best solution when there is more than one alternative?
- Time complexity: How long does it takes to find a solution?
- Space complexity: How much memory is needed to perform the search?

Types of search methods

- Uninformed search methods (blind search)

Have no additional information about states beyond that provided in the problem definition, namely

- Number of states that that can be reached from the current sate
- Path cost from the current state to the goal state

- Informed search methods:
  - Have access to this information

Types of search methods

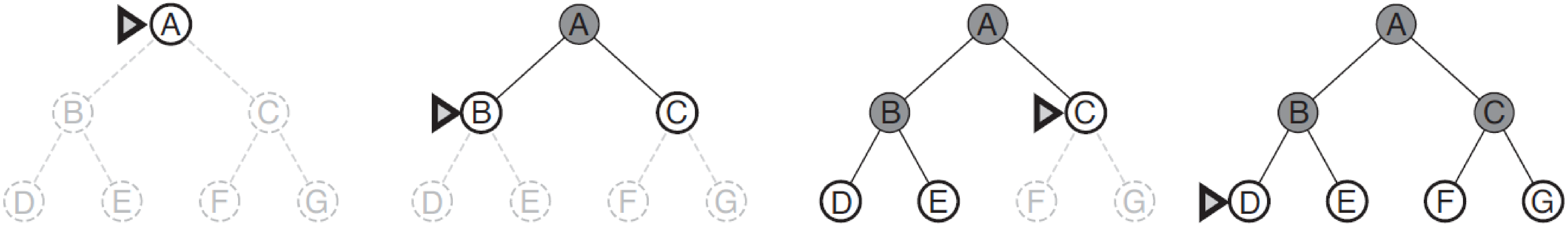
- In general, informed search methods are more efficient than uninformed search methods
- However, uninformed search methods are also important because there are many problems where no additional information exists

Uninformed search methods

- Breadth-first search
- Uniform-cost search
- Deep-first search
- Depth-limited search
- Iterative deepening depth-first search
- Bidirectional search

Breadth-first search

- Idea: nodes from level *n* are all expanded before nodes from level *n* + 1



Breadth-first search

- Frontier: FIFO (queue)
- Characteristics:
  - Complete
  - Optimal, if the path cost is a nondecreasing function of the depth of the node. The most common such scenario is that all actions have the same cost
  - Time complexity: $O(b^d)$
  - Space complexity: $O(b^d)$

b: tree's ramification factor

d: solution depth

Big O notation

```
System.out.println("AI");
System.out.println("AI");
System.out.println("AI");

for(int i = 0; i < n; i++)
    System.out.println("AI");

for(int i = 0; i < n; i++)
    System.out.println("AI");

for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        System.out.println("AI");

for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        for(int k = 0; k < n; k++)
            System.out.println("AI");

for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        for(int k = 0; k < n; k++)
            System.out.println("AI");
```

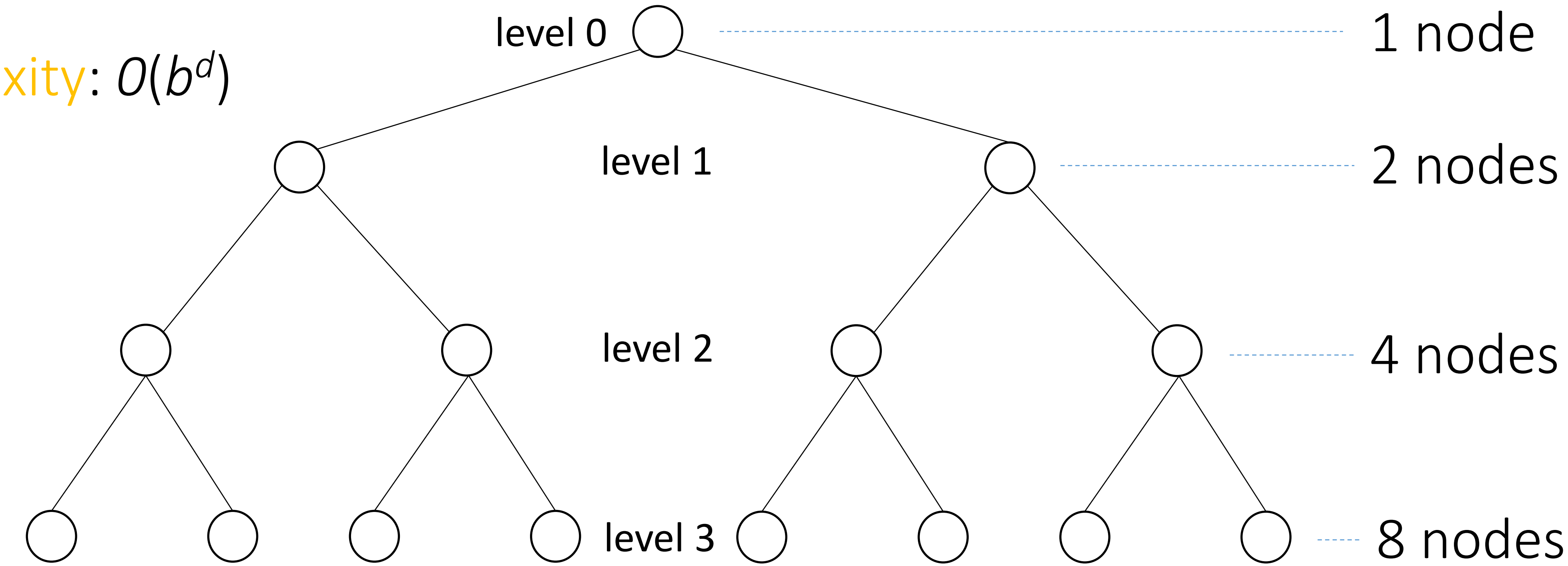
How many times is “AI” written?

$3 + 2n + n^2 + 2n^3$ -> for very large n, the value of n^3 completely dominates the rest of the expression, so, in this case, the algorithm (time) complexity is $O(n^3)$

Breadth-first search complexity

- Time complexity: $O(b^d)$
- Space complexity: $O(b^d)$

$b = 2$



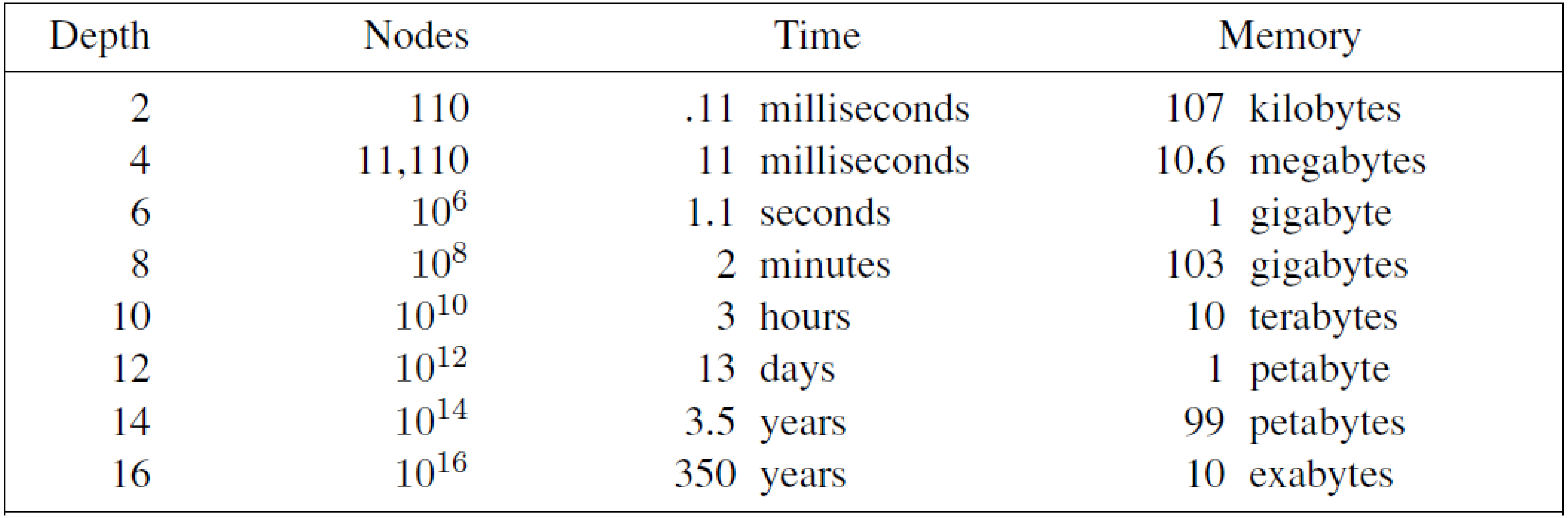
$$1 + 2 + 4 + 8 = 2^0 + 2^1 + 2^2 + \boxed{2^3} = O(b^d)$$

Breadth-first search

- It takes too long if the solution has many steps, because it checks all shorter possibilities

- Inefficient for problems with a large ramification factor, where its application may be impossible

Breadth-first search



Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

b = 10; 1 million nodes per second; 1000 bytes per node

Uniform-cost search

- Idea: always expand the node from the frontier with smaller cost (given by the *g*(*n*) function)

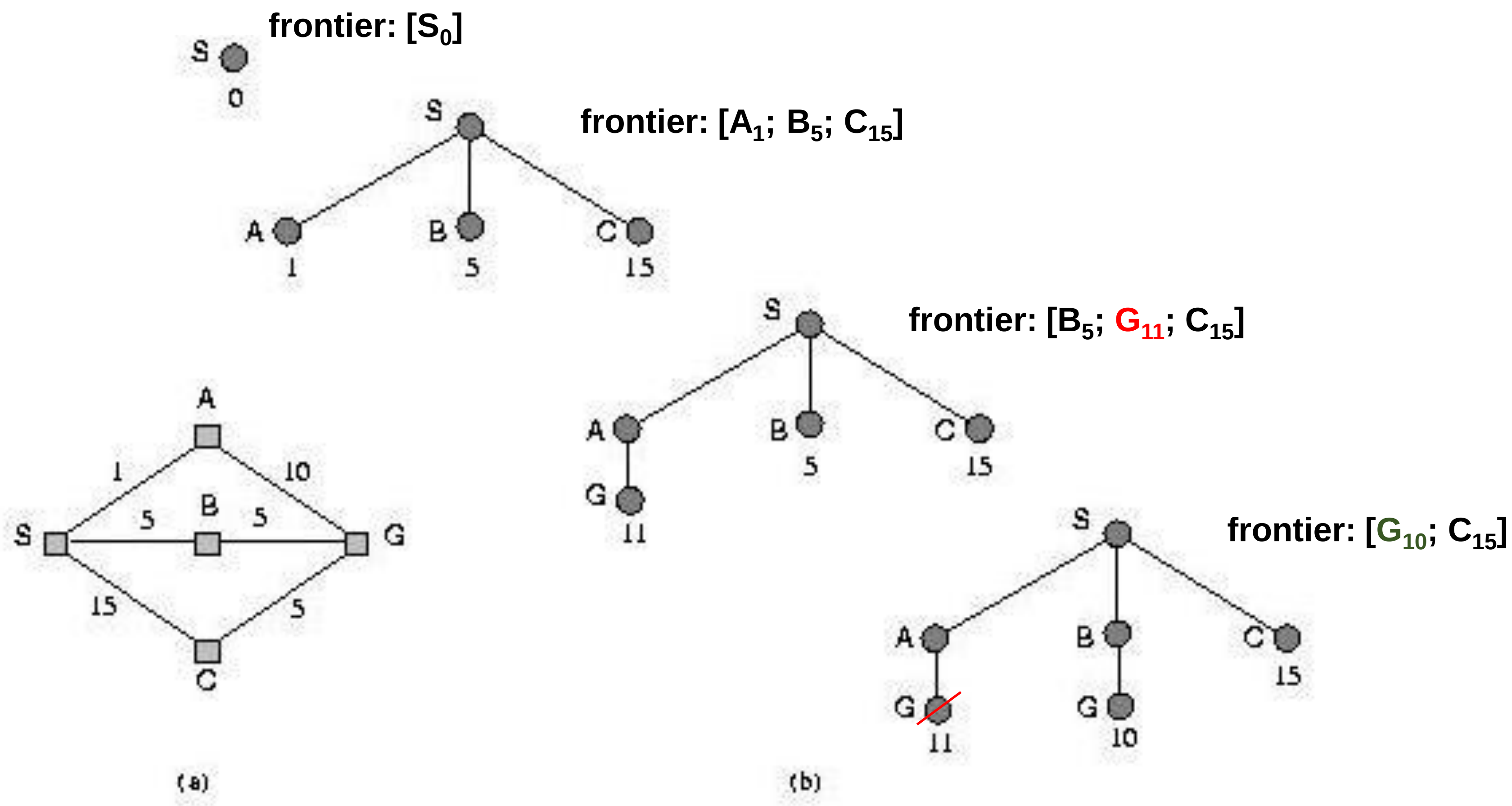
- This method is equivalent to breadth-first search when *g*(*n*) = depth(*n*)

Uniform-cost search

- Frontier: priority queue where nodes are ordered by the value of $f(n)$
- In this case, $f(n) = g(n)$
- Characteristics:
  - Complete
  - Optimal
  - Time complexity: $O(b^m)$
  - Spatial complexity: $O(b^m)$

m: maximum depth of the search tree

Uniform-cost search



frontier: [S_0]

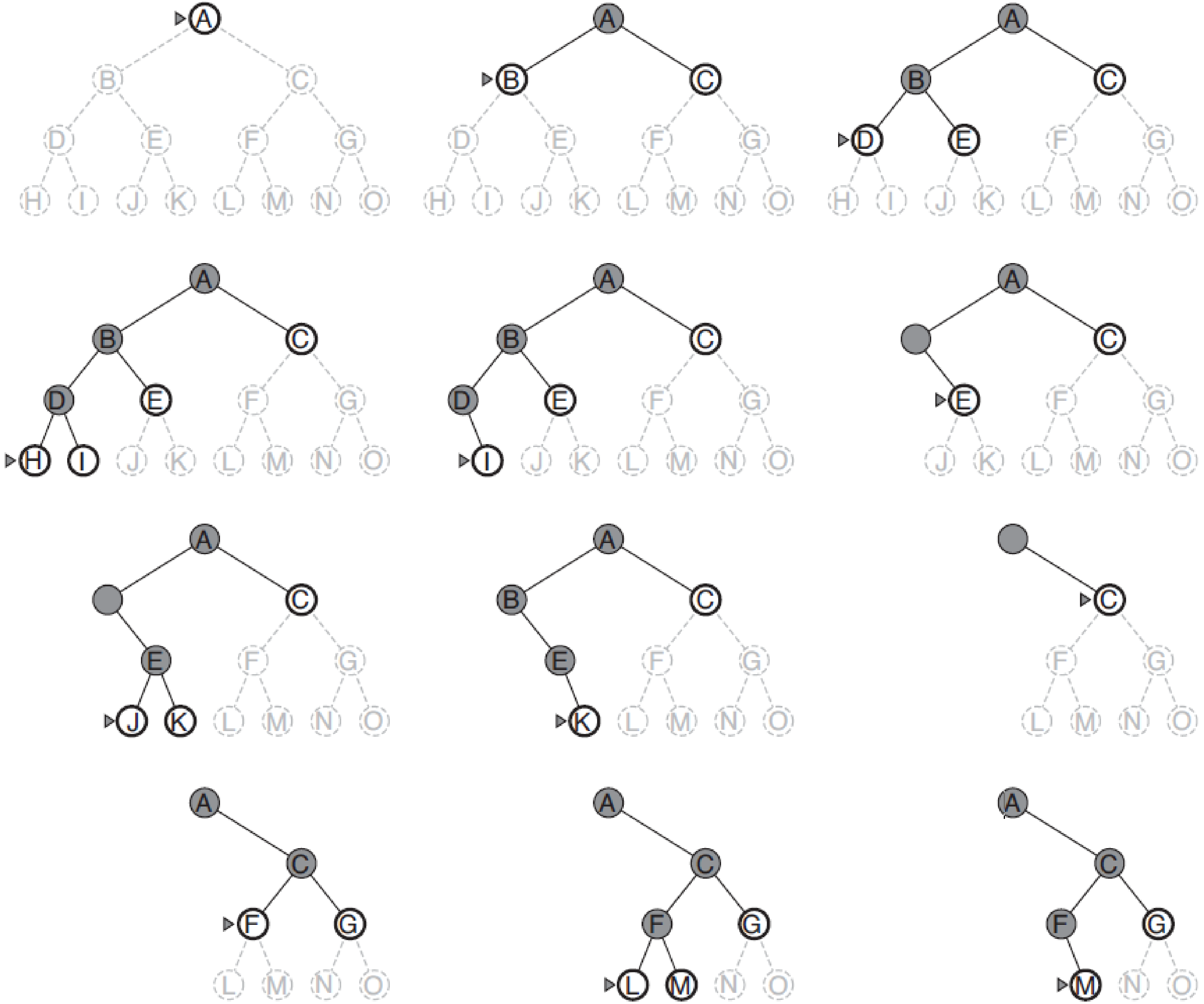
frontier: [A_1; B_5; C_{15}]

frontier: [B_5; G_{11}; C_{15}]

frontier: [G_{10}; C_{15}]

Depth-first search

- Idea: always expand the deepest node


A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G H I J K L M N O
A B C D E F G I J K L M N O
A C E F G J K L M N O
A C E F G J K L M N O
A B C E F G K L M N O
C F G L M N O
A C F G L M N O
A C F G L M N O
A C F G M N O

Depth-first search

- Frontier: LIFO (stack)

- Characteristics:
  - Complete (in finite spaces)
  - Nonoptimal
  - Time complexity: $O(b^n)$
  - Space complexity: $O(b * n)$ if we don't use the explored set. If the explored set is used, space complexity is $O(b^n)$

Depth-limited search

- Idea: expand the nodes in depth until some predefined level *l*

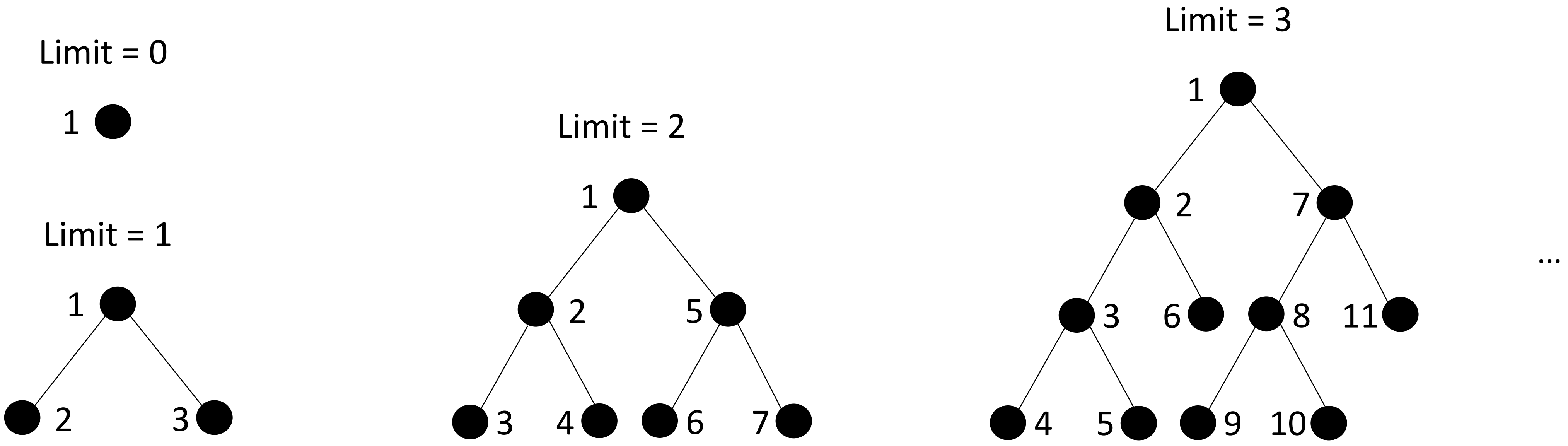
- Characteristics:
  - Complete if *l* >= *d*
  - Nonoptimal
  - Time complexity: $O(b^l)$
  - Spatial complexity: $O(b * l)$

Iterative deepening depth-first search

- Idea: Expand nodes in depth, first up to level 0, then **starts from the beginning** up to level 1, then **starts from the beginning** up to level 2, …
- Characteristics:
  - Complete
  - Optimal
  - Time complexity: $O(b^l)$
  - Spatial complexity: $O(b * l)$

It exhibits the best from breadth-first search and depth-first search algorithms

Iterative deepening depth-first search



Iterative deepening depth-first search

Cost of the process of multiple expansions (example: $b = 10$; $d = 5$)

- Depth-first search

  $1 + b + b^2 + b^3 + b^4 + b^5 =$

  $1 + 10 + 100 + 1000 + 10000 + 100000 = 111.111$

- Iterative deepening depth-first search

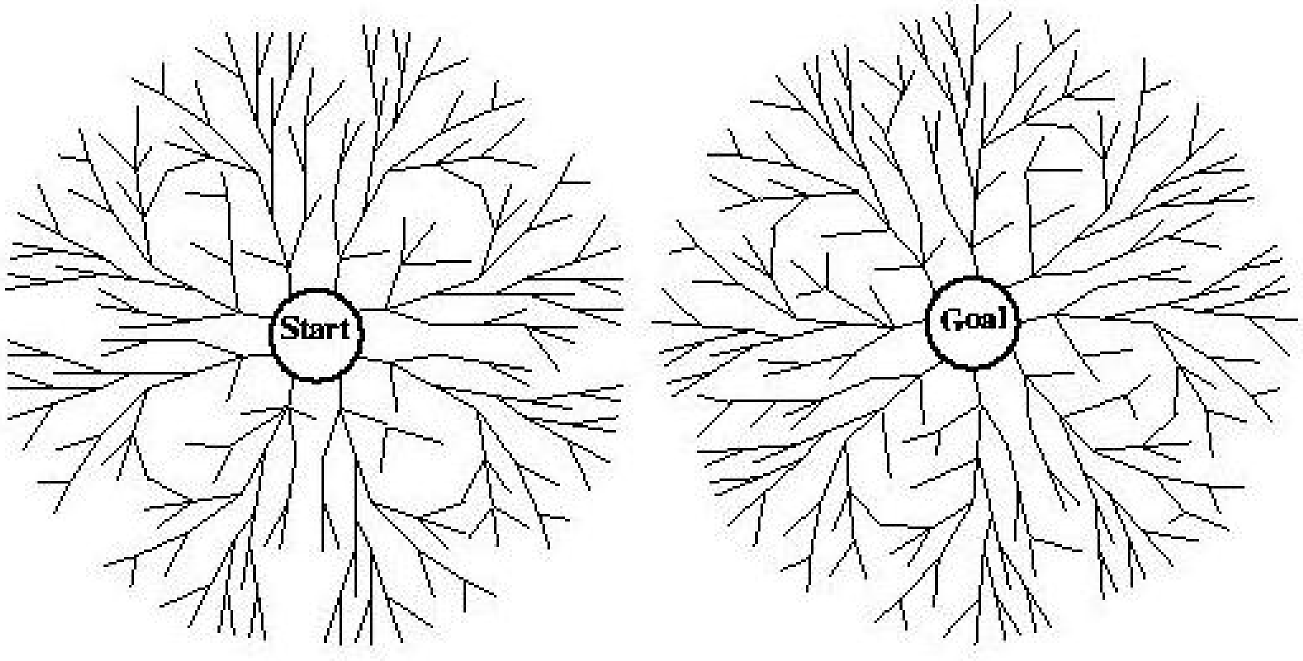
  $(d + 1) * 1 + d * b + (d - 1)\, b^2 + \ldots + 1 * b^5 =$

  $6 + 50 + 400 + 3000 + 20000 + 100000 = 123.456$

- In general, iterative deepening depth-first search is the method that should be used when the state space is large, the depth of the goal state is unknown and we want the best solution

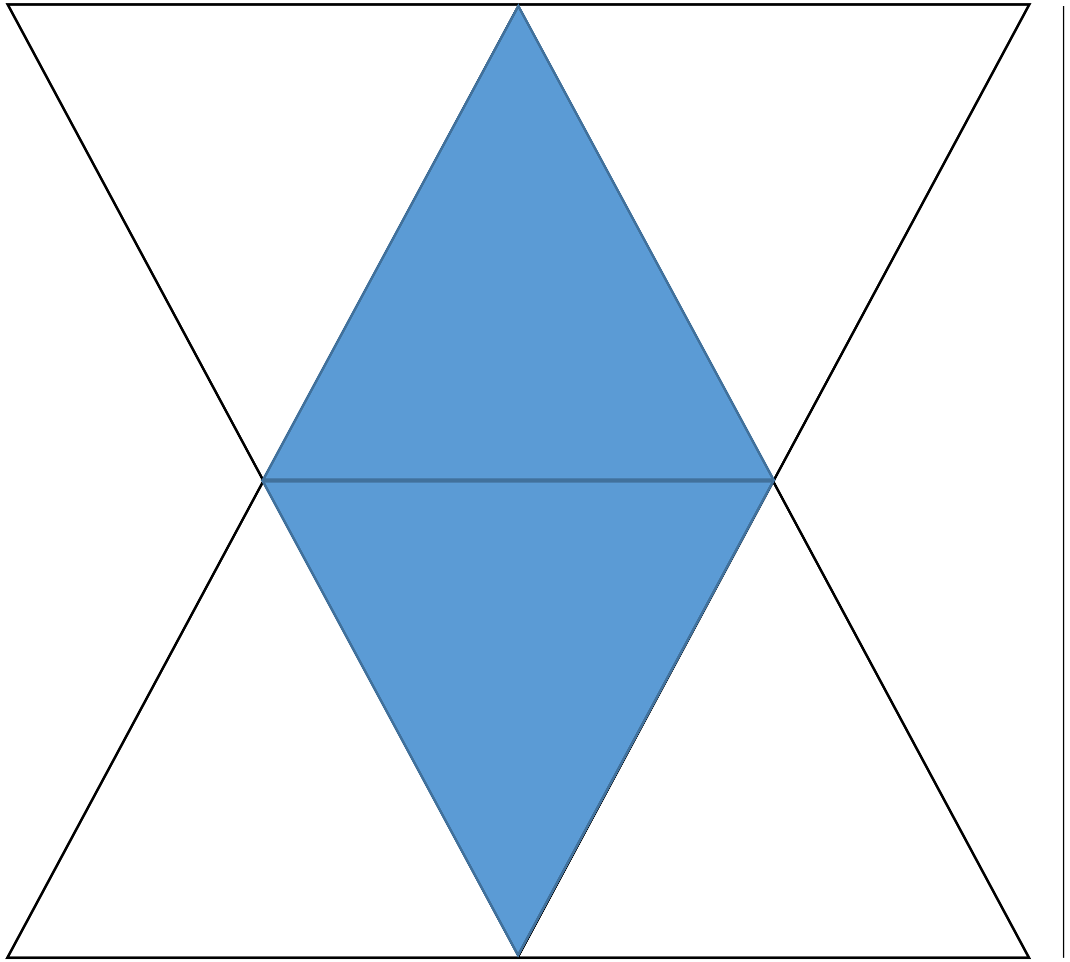
Bidirectional

- Idea: expand nodes, both from the initial state and the goal state(s)



Bidirectional

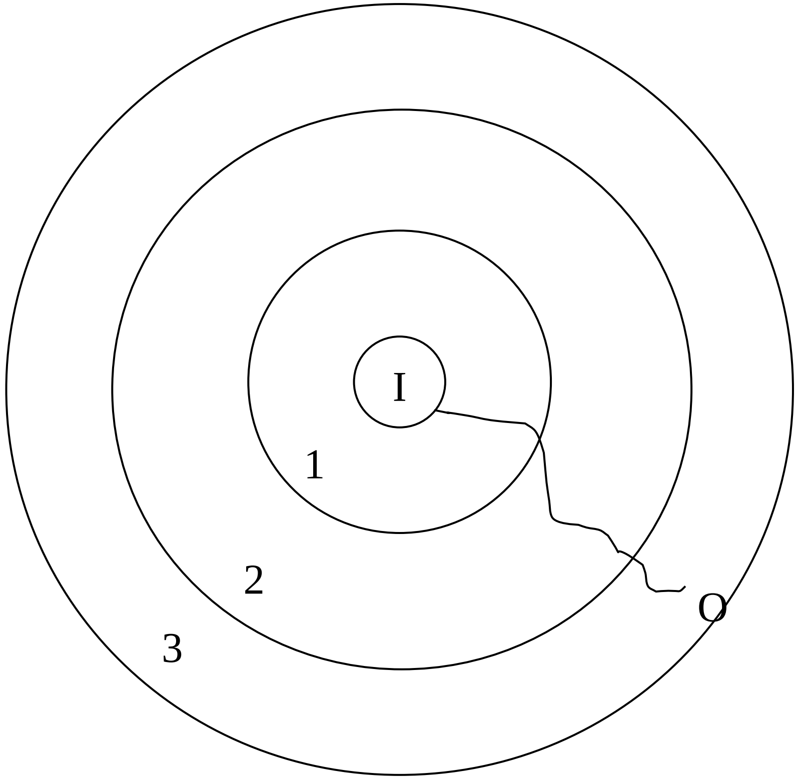
- Characteristics:
  - Complete (if applicable)
  - Optimal (if applicable)
  - Time complexity: $O(b^{d/2})$
  - Space complexity: $O(b^{d/2})$



Bidirectional

- In this method one must consider:
  - Predecessors generation
  - We may need to search from more than one goal state (described explicitly or implicitly)
  - There must be a method of identifying the same node in both search directions
  - Search method to be used in each search direction

Uniform-cost search again



- Uniform-cost search is the method using more problem information so far
- It first examines all paths with smaller cost (circle 1, then circle 2, then circle 3)
- It has no idea about the “distance” to reach O
- In circle 3, it examines half the nodes on average
- We need other type of information to improve the performance of the search process

Informed search methods

- They use information about the domain to select the best path

- For that, it is defined a heuristic function *h*(*s*) that estimates the cost (or distance) of the path from a state *s* to the goal state

- The frontier is a priority queue where nodes are added in ascending order of the value of a function *f*, which depends on the value of *h*

h(*s*) properties

- Properties that *h*(*s*) must have
  - *h(s) >= 0* if *s* is not a goal state
  - *h*(*s*) = 0 if *s* is a goal state
  - *h*(*s*) has an infinite value if it is impossible to reach the goal from *s*

Examples of heuristics

- Missionaries and cannibals: number of people in the departing margin
- 8-Puzzle: number of misplaced tiles
- 8-Puzzle: sum of the distances of the tiles to their goal positions

Informed search methods

- Greedy best-first search
- A* search
- Beam search
- Limited memory algorithms: IDA* and SMA*

Greedy best-first search

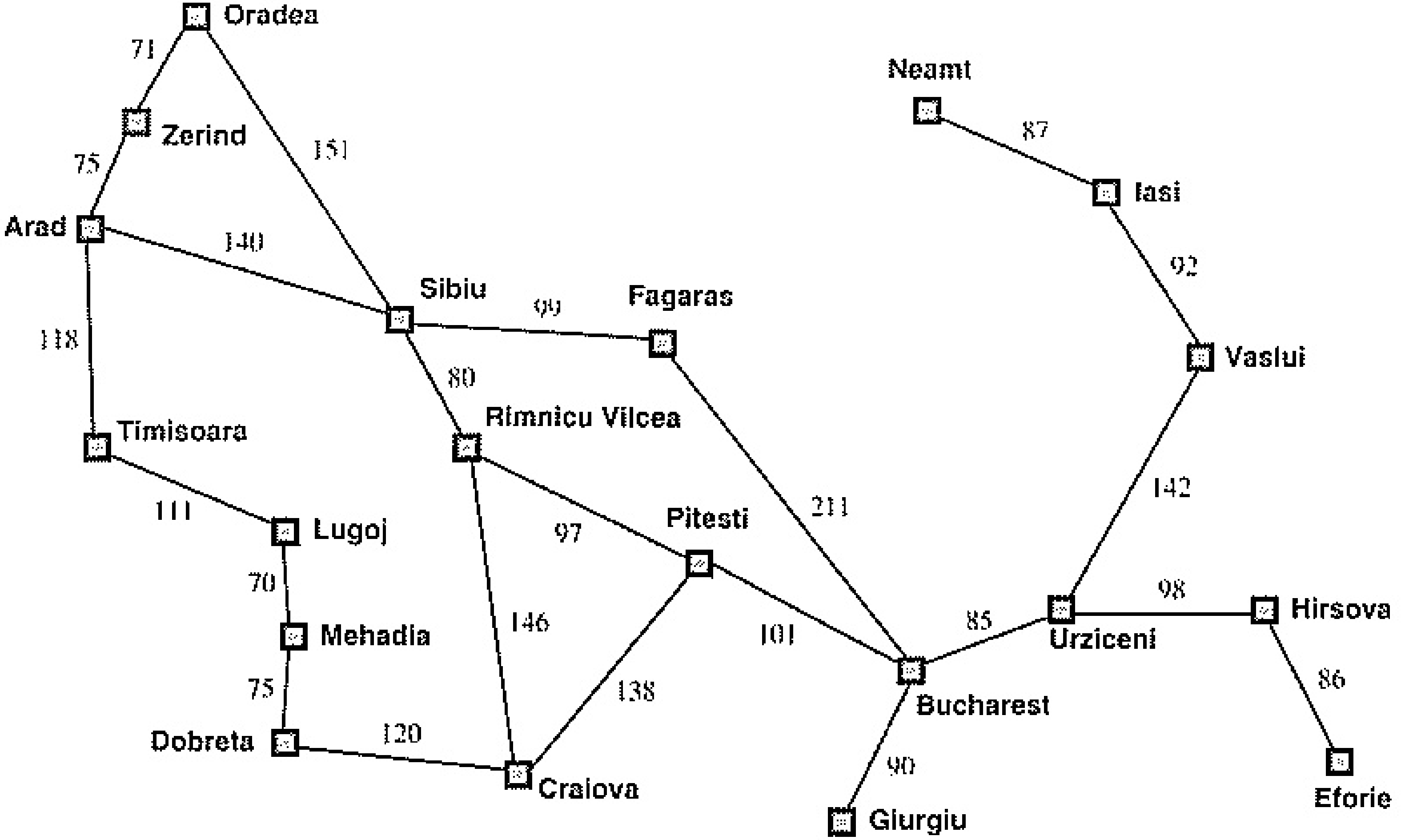
- Idea: selects, to expand, the node that is thought to be closer to the goal, that is, the one with smaller value of *h*

- $f(n) = h(n)$

- Characteristics:
  - Complete (in finite spaces)
  - Nonoptimal
  - Time and space complexity: $O(b^m)$, in the worst case

m – maximum depth of the state space

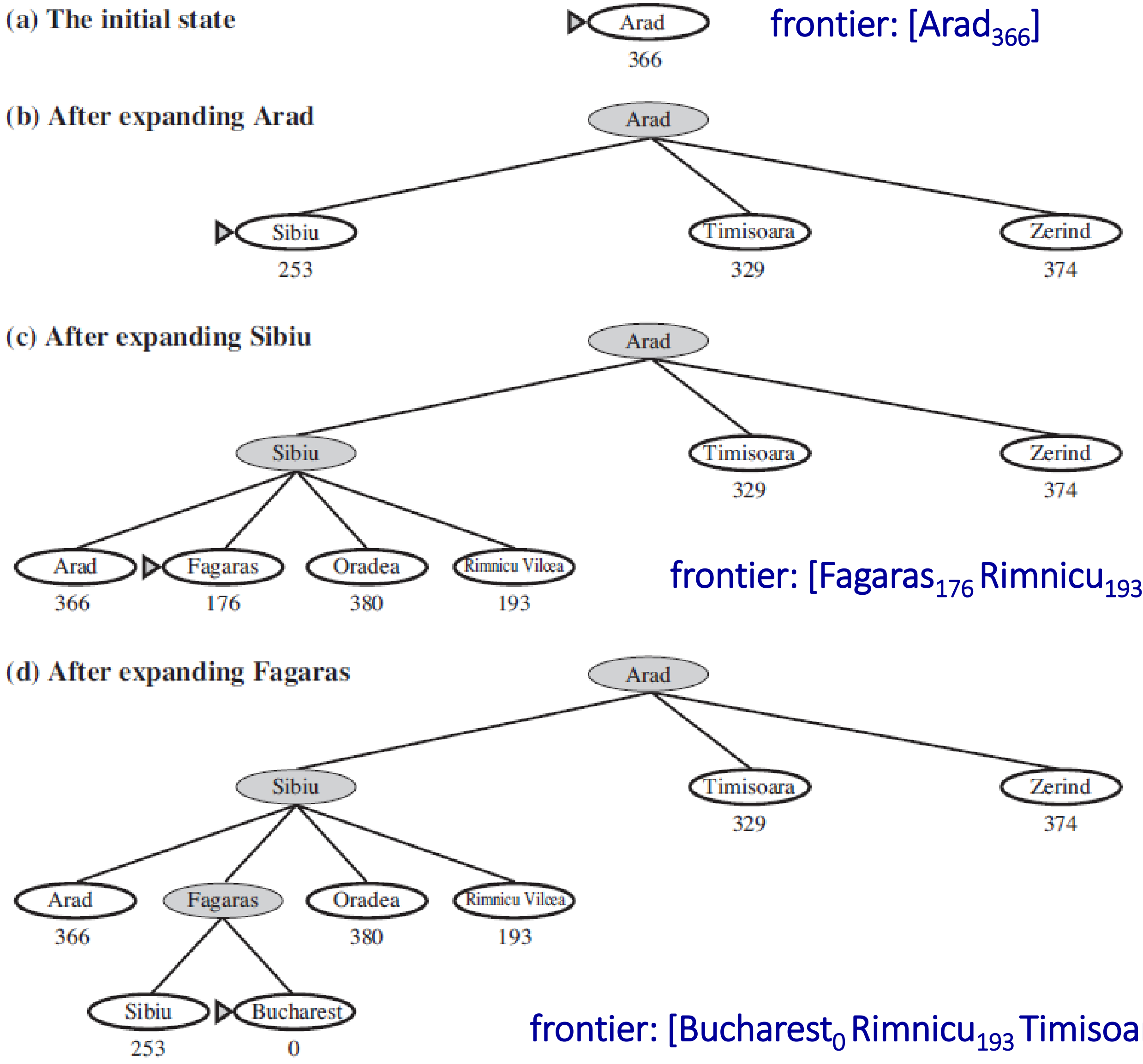
Greedy best-first search



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

Greedy best-first search



(a) The initial state

frontier: [Arad$_{366}$]

(b) After expanding Arad

frontier: [Sibiu$_{253}$ Timisoara$_{329}$ Zerind$_{374}$]

(c) After expanding Sibiu

frontier: [Fagaras$_{176}$ Rimnicu$_{193}$ Timisoara$_{329}$ Zerind$_{374}$ Oradea$_{380}$]

(d) After expanding Fagaras

frontier: [Bucharest$_{0}$ Rimnicu$_{193}$ Timisoara$_{329}$ Zerind$_{374}$ Oradea$_{380}$]

A* search

- Idea: tries to expand the node belonging to the path with the smaller associated cost

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

- $g(n)$ = cost of the path from the root (initial state) to node n

A* search - admissible heuristics

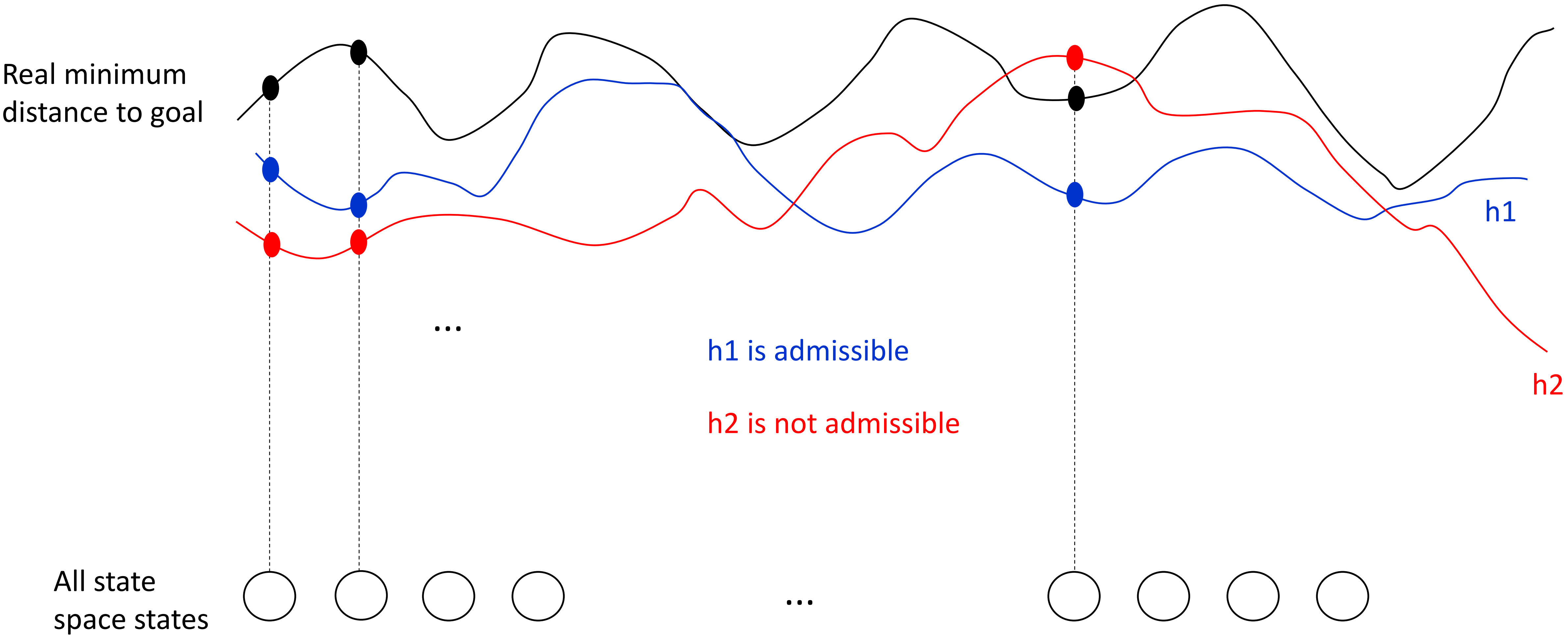
- *h* is said to be admissible if *h*(*n*) <= *h**(*n*), for all *n*, where *h**(*n*) is equal to the real cost of the path with minimum cost from *n* to the goal

- That is, an admissible heuristic never overestimates the cost to reach the goal

- The use of an admissible heuristic guaranties that a node belonging to an optimum path does not seem bad to the point of never being chosen

- A* search is complete and optimal given that *h*(*n*) is admissible

A* search - admissible heuristics



Real minimum distance to goal

All state space states

...

...

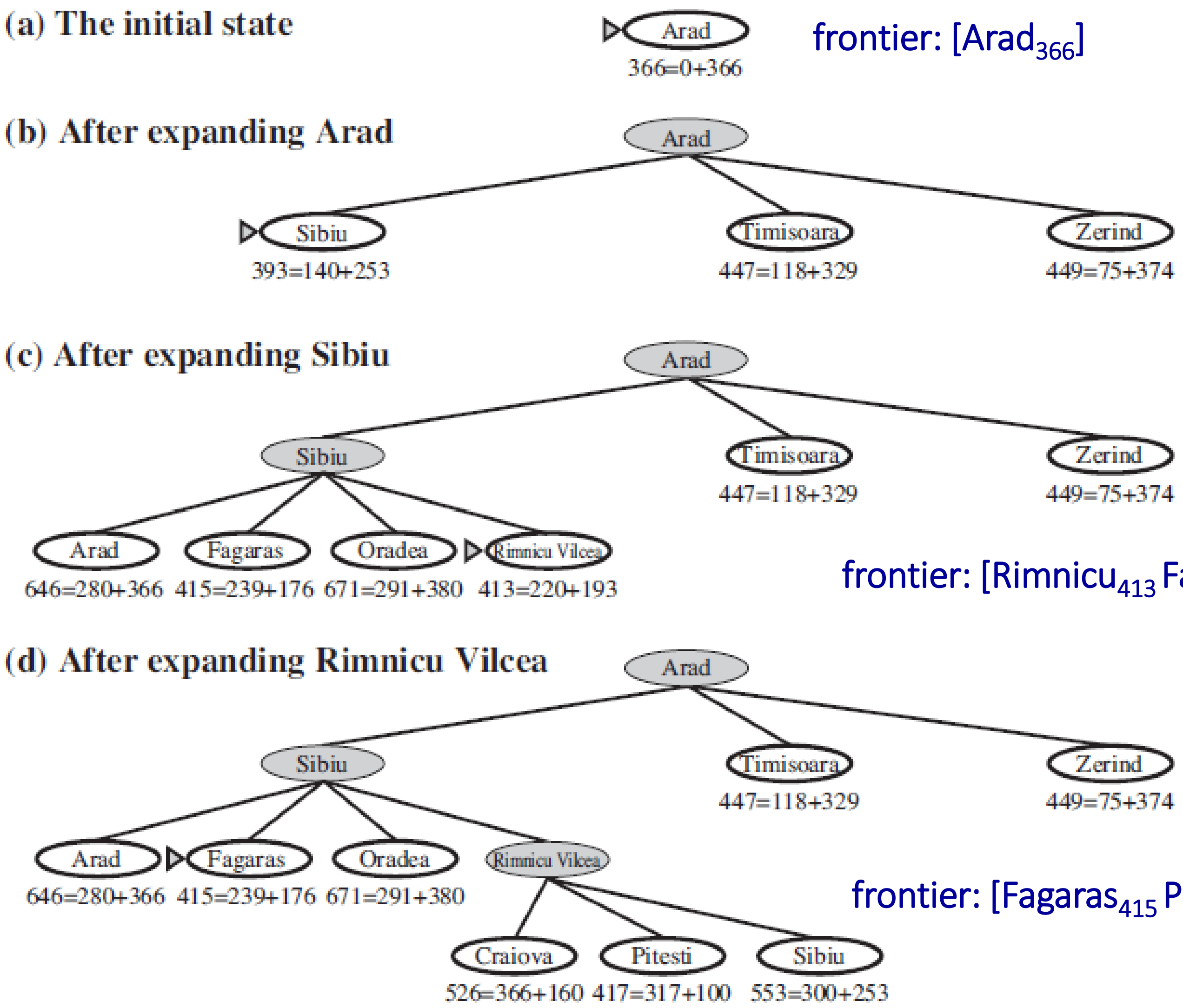
h1

h2

h1 is admissible

h2 is not admissible

A* search



(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Rimnicu Vilcea

frontier: [Arad$_{366}$]

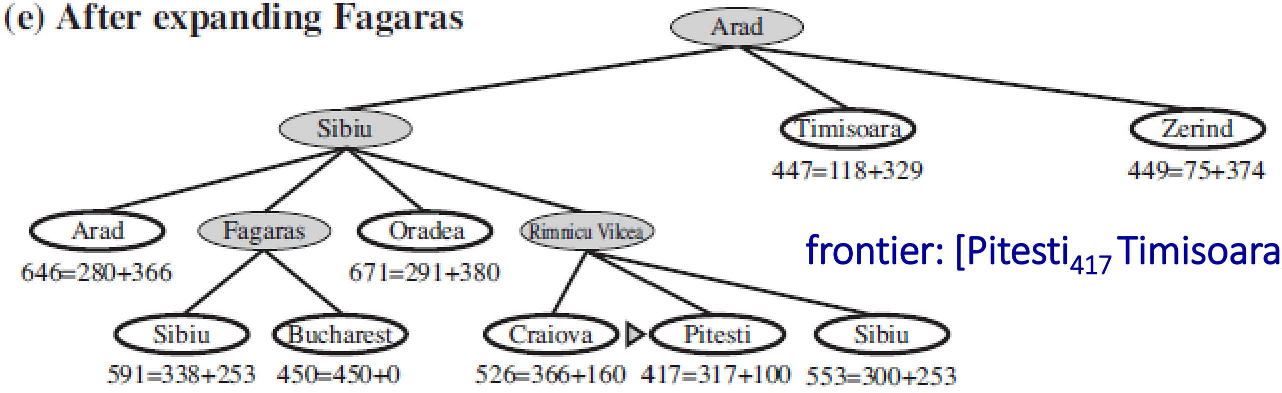
frontier: [Sibiu$_{393}$ Timisoara$_{447}$ Zerind$_{449}$]

frontier: [Rimnicu$_{413}$ Fagaras$_{415}$ Timisoara$_{447}$ Zerind$_{449}$ Oradea$_{671}$]

frontier: [Fagaras$_{415}$ Pitesti$_{417}$ Timisoara$_{447}$ Zerind$_{449}$ Craiova$_{526}$ Oradea$_{671}$]

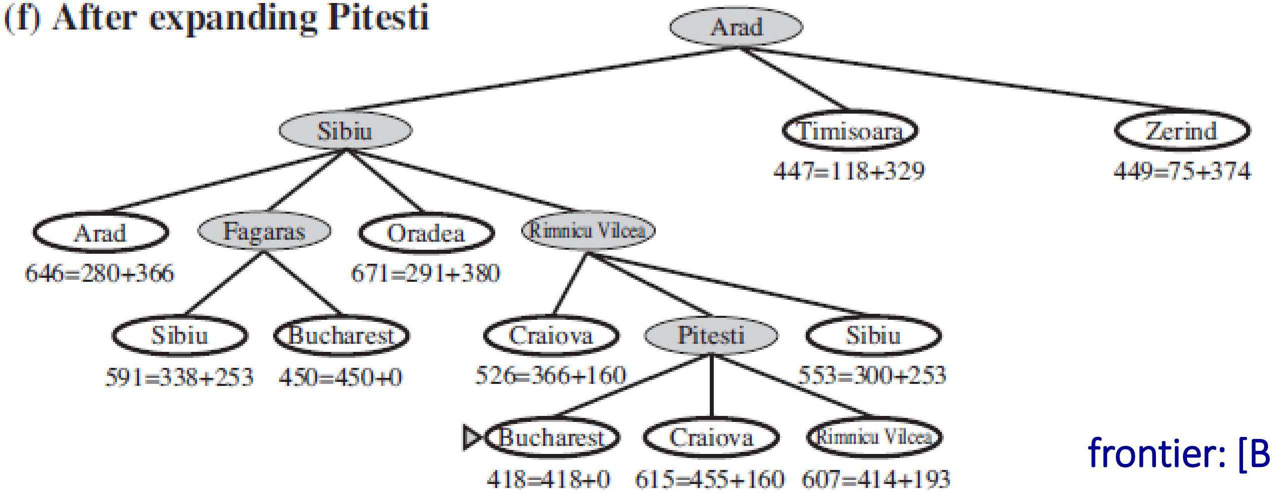
A* search

(e) After expanding Fagaras



frontier: [Pitesti$_{417}$ Timisoara$_{447}$ Zerind$_{449}$ Bucharest$_{450}$ Craiova$_{526}$ Oradea$_{671}$]

(f) After expanding Pitesti



frontier: [Bucharest$_{418}$ Timisoara$_{447}$ Zerind$_{449}$ Craiova$_{526}$ Oradea$_{671}$]

A* search – consistency condition

- In general, the state space can be represented as a graph where there can be more than one path to reach a node

- So, we may need to verify if the cost $g(n)$ of a node generated in step 8 of the graph-search algorithm is smaller than the one of an equivalent node already present in the frontier and, if that is the case, to replace it, as happens in the uniform-cost search method

A* search – consistency condition

- There is a condition, named consistency condition, that we can impose to *h* that, when satisfied, guaranties that the first path to be generated to reach some state is already the optimum path to that state

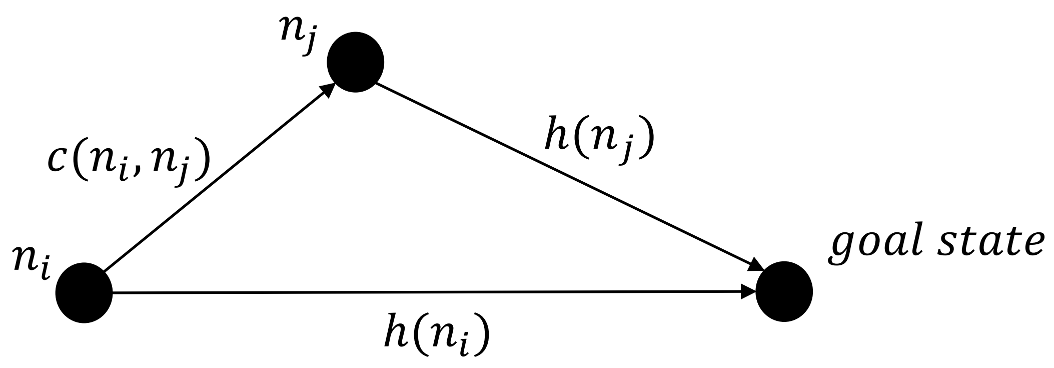
- If this condition is satisfied, there is no need to verify if the node already present in the frontier has a cost *g*(*n*) larger than the one of the new node

A* search – consistency condition

- Let us consider a pair of nodes such that n_j is a successor of n_i; Heuristic h is said to verify the consistency condition if, for all pairs in the search graph

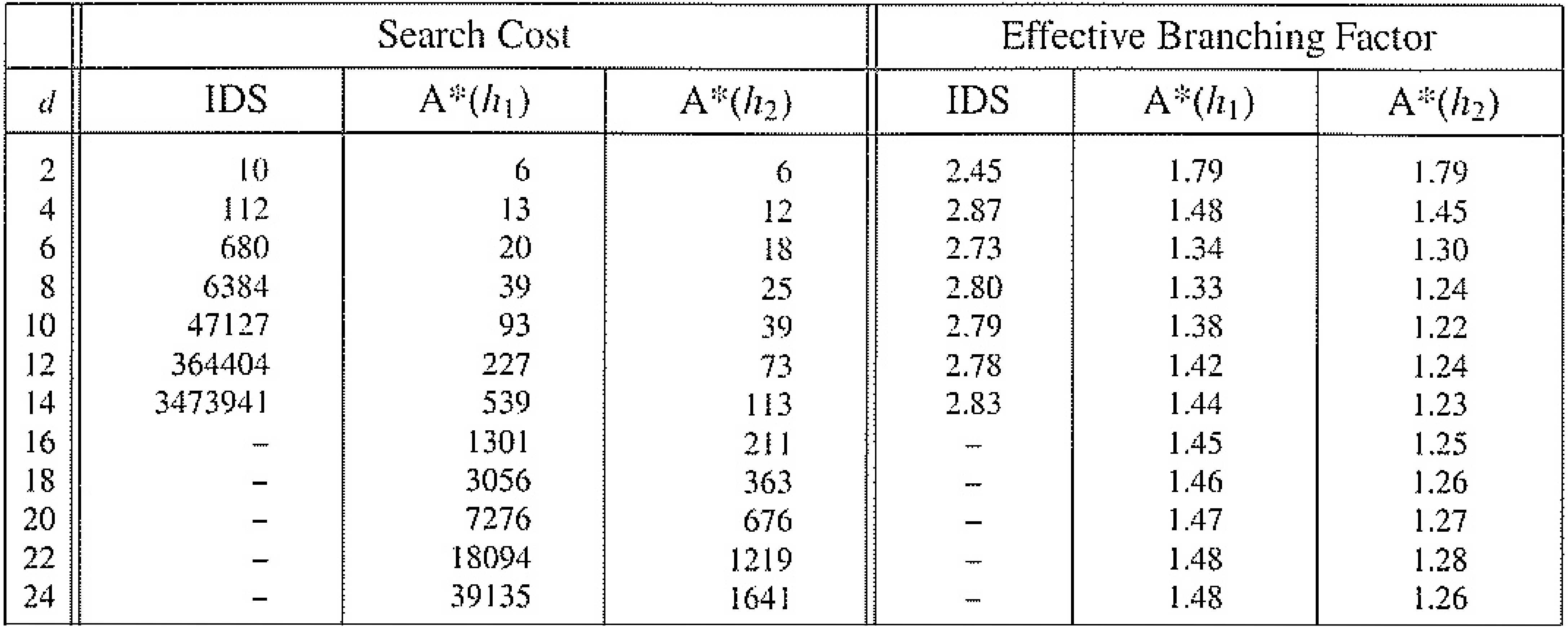
Intuition: a consistent heuristic becomes more precise as we get deeper in the search tree

Note: all consistent heuristics are also admissible



$$h(n_i) \leq c(n_i, n_j) + h(n_j)$$

The effect of heuristics



	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Beam search

- It is similar to A* search but establishes a maximum size to the frontier

- After adding the successors of a node to the frontier, the last nodes are removed until the frontier as the established maximum

Beam search

- It allows the search to concentrate only on the most promising states, ignoring the less promising ones

- It is not complete nor optimal

- It can be a good choice if there are many possible solutions

IDA* (iterative deepening)

- Idea: it is similar to the iterative deepening depth-first search but, instead of imposing limits on the search depth, it imposes successive limits (contours) to the cost of the solutions that can be computed

- The limit starts to be the *f* value for the initial node and, in the following iterations, it takes the smaller *f* value of all the nodes that were not expanded in the previous iteration

- The algorithm stops when it finds a solution with a cost smaller then the current limit value

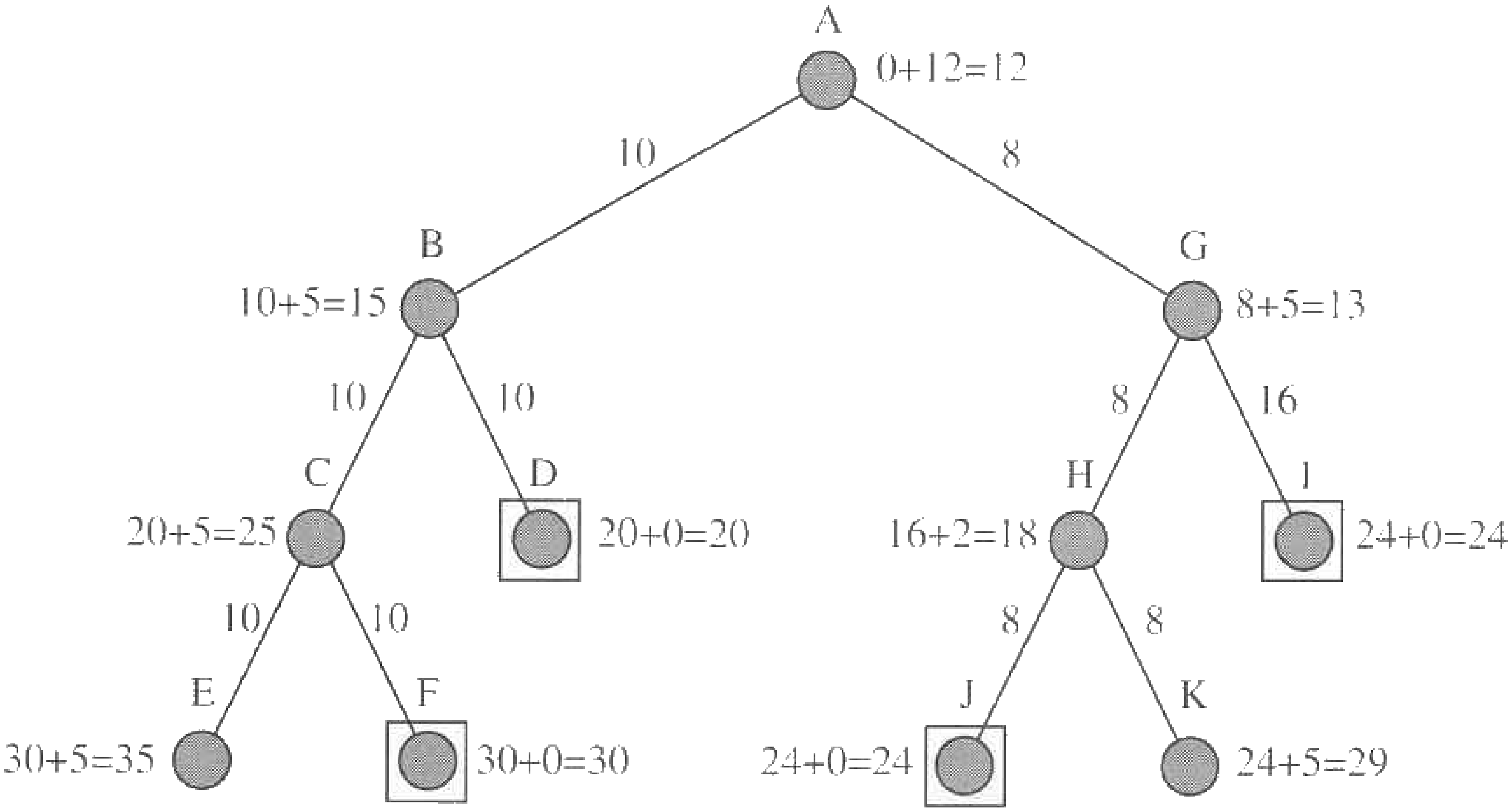
IDA* (iterative deepening)

- It is complete and optimal

- Observations:
  - It only retains the value of the smaller *f* from an iteration to the next

- If the heuristic’s value is different for all states, in each new iteration only one more node is explored

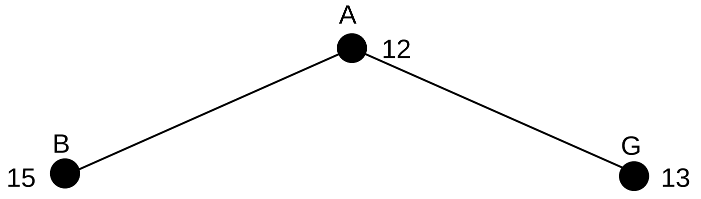
IDA* (iterative deepening)



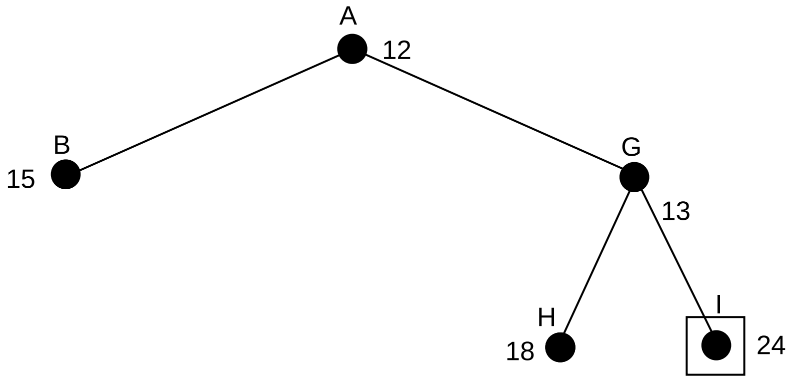
IDA* (iterative deepening)

- Example (cont.)

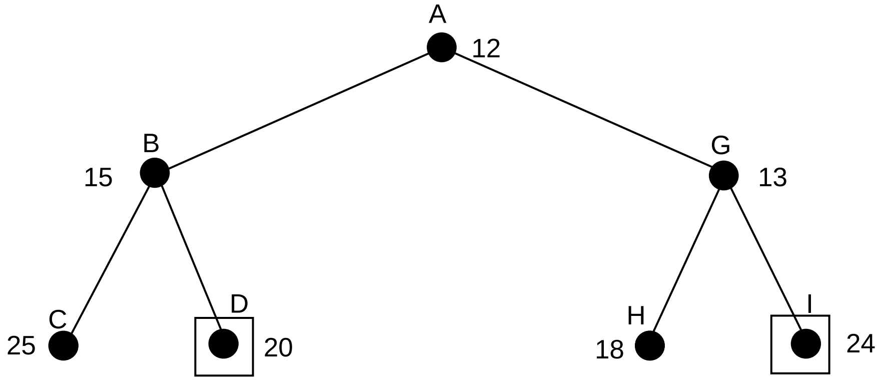
1st iteration (limit = 12)



2nd iteration (limit = 13)

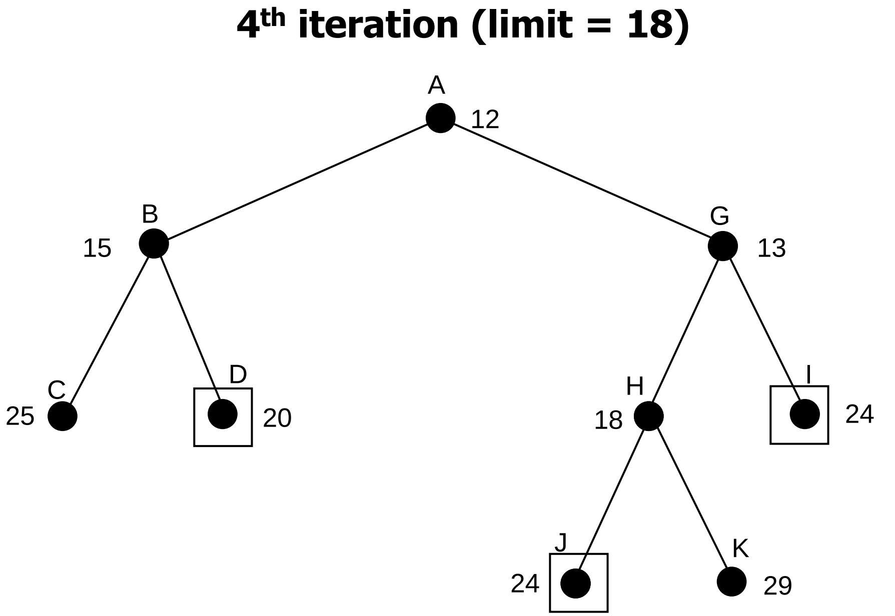


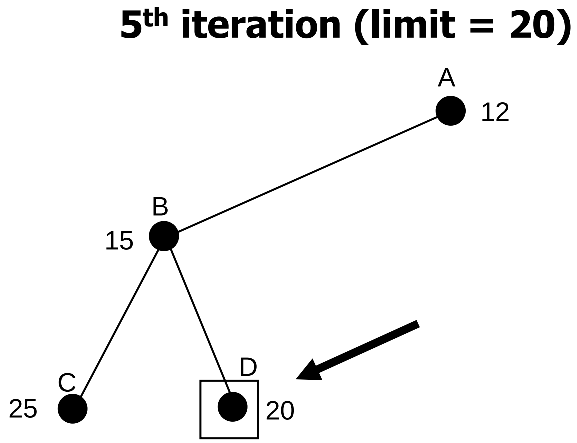
3rd iteration (limit = 15)



IDA* (iterative deepening)

- Example (cont.)





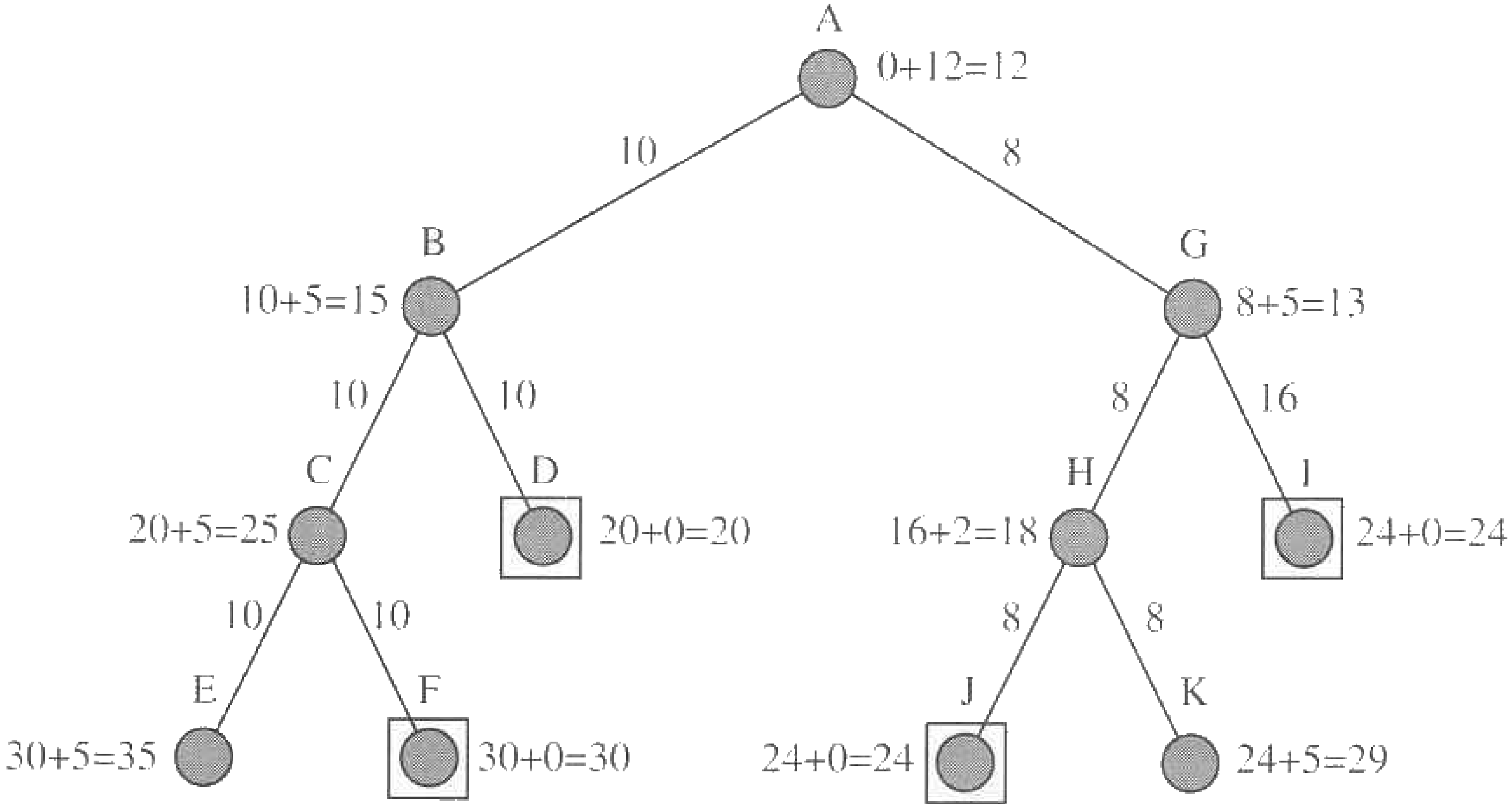
SMA*

- It allows to do a A* search with memory limitations
  - It uses all available memory
  - Avoids repeated states (given that there is enough memory)
  - Complete (given that there is enough memory)
  - Optimal (given that there is enough memory)

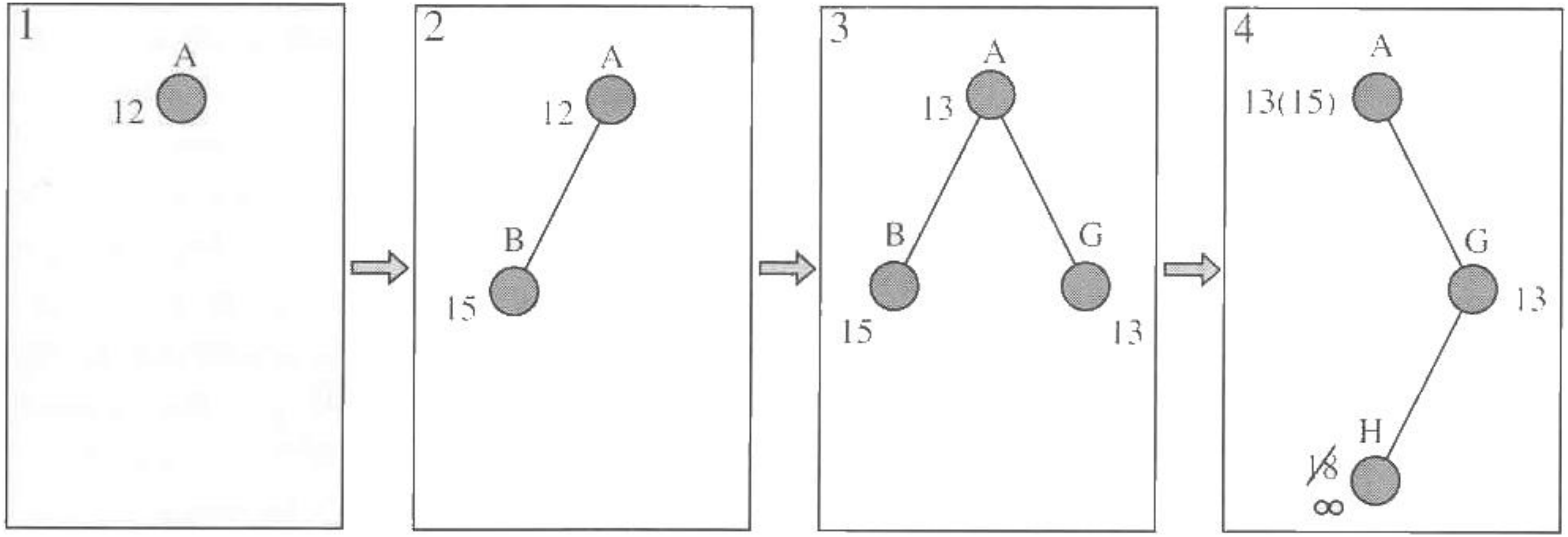
SMA*

- When it needs memory:
  - Eliminates nodes with higher *f* values
  - Information about the “quality” of the eliminated node is saved in its parent node
  - An eliminated node is regenerated only if all other available paths seem worst than that node

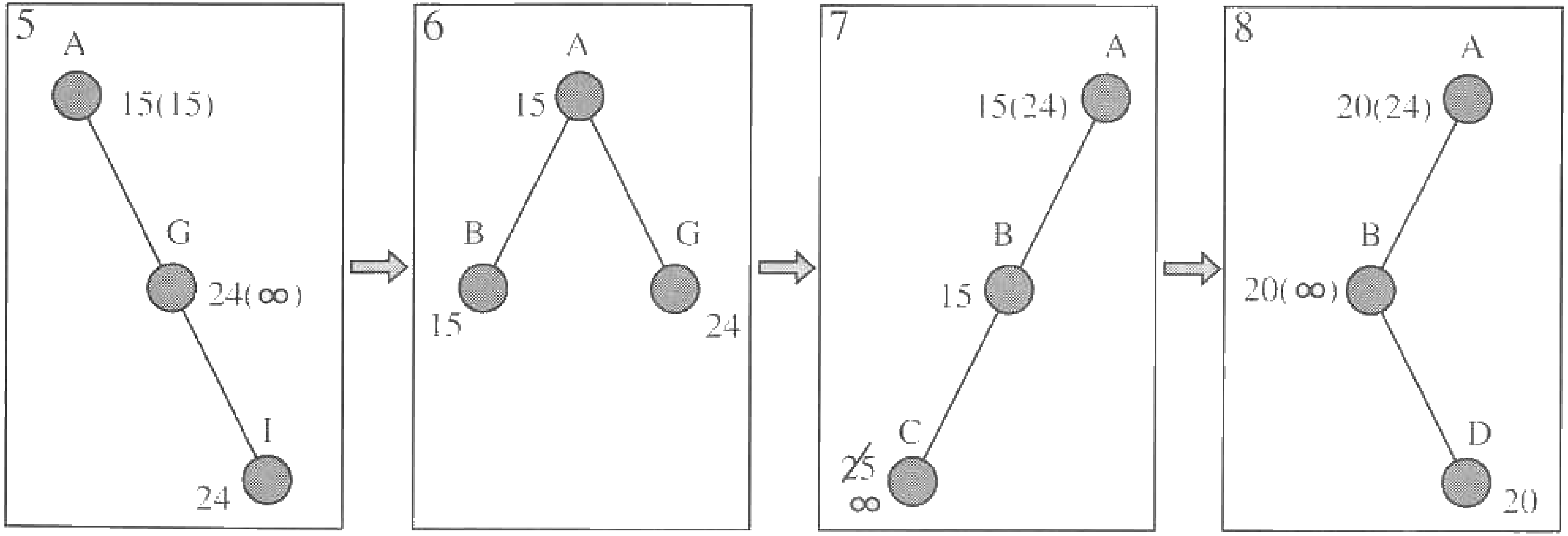
SMA*


A
0+12=12
10
8
B
10+5=15
G
8+5=13
10
10
8
16
C
20+5=25
D
20+0=20
H
16+2=18
I
24+0=24
10
10
8
8
E
30+5=35
F
30+0=30
J
24+0=24
K
24+5=29

SMA*



SMA*



Heuristic functions

- A good heuristic should be computed quickly; It may be preferable to expand more nodes using a faster heuristic

- Sometimes, it is possible to consider the characteristics of a state to formulate the heuristic
    - Ex. 1: 8 Puzzle - number of misplaced tiles
    - Ex. 2: Missionares and Canibals: number of persons in the departing margin

Heuristic functions

- Sometimes, a good heuristic can be formulated by computing the exact cost of a solution for a simplified version of the problem
  - Ex. 1: 8 Puzzle - number of misplaced tiles: this would be the exact solution cost in a version of the problem where we could just pick up a misplaced tile and put it directly in it goal position
  - Ex. 2: 8 Puzzle – (Manhattan) distance of each misplaced tile to its goal position: this would be the exact solution cost in a version of the problem where we could move each tile over other tiles up to their goal position (through a “Manhattan path”)

Heuristic functions

- If an optimal solution is not demanded, the use of an heuristic that occasionally overestimates the real cost but that is usually close to that value, leads to much less generation of nodes than if an admissible heuristic were used

Heuristic functions

- Statistical heuristics (ex: the average time that people take between places): this may lead to non admissible heuristics but which, still, can lead to good solutions

- We can use a group of admissible heuristics where none of them dominates another; then we always chose the higher value

Search algorithms

- They can only be applied to the following types of environments:
  - Fully observable
  - Deterministic
  - Static
  - Discrete

- Informed search algorithms use a heuristic function… that is given to them!
  - But, where the real intelligence is?