University Mohamed Boudiaf of M'sila
Faculty of Mathematics and Informatics

Introduction to Artificial Intelligence

Chapter IV: Research Strategies

Dr. SAID KADRI

Associate Professor

Department of Computer Science, Faculty of Mathematics and Informatics, University Mohamed Boudiaf of M'sila

E-mail:kadri.said28@gmail.com

Website:https://kadrisaid28.wixsite.com/sgadri

2017 - 2018

Preface

- One of the main components of an intelligent system is the inference engine (plus the BC).
- An inference engine is characterized by:

1. **A basic cycle which includes 04 phases:**
   - The selection phase (gathering, sorting facts and rules of the BC having a specific role)
   - The filtering phase: determining which rules can be applied among those of the BC
   - The conflict resolution phase: choosing a rule to apply according to a choice strategy
   - The execution phase: apply the chosen rule

2. **A chaining method:**
   - Forward chaining
   - Backward chaining
   - Mixed chaining

3. **A research strategy**
   - A blind strategy
   - An informed strategy (heuristic)

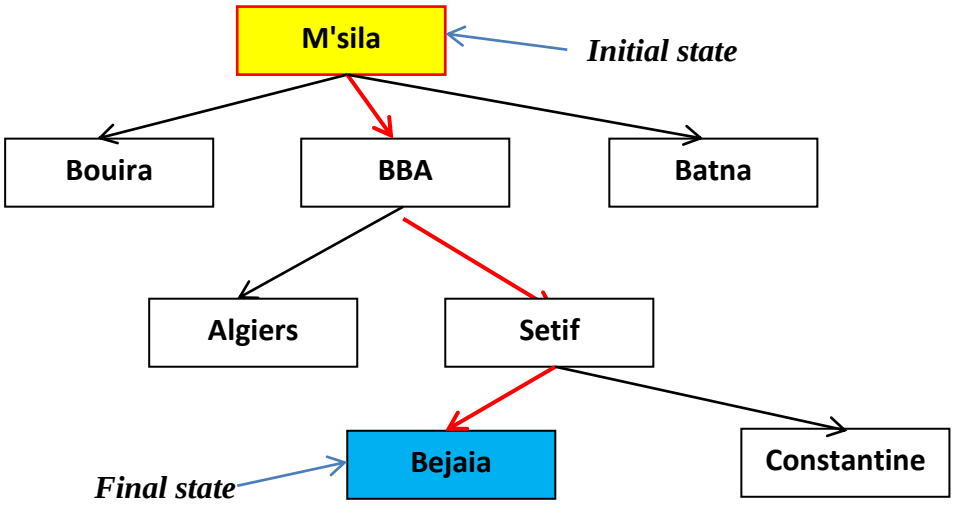
Formulation of a problem in state space

1. ***State:***an abstraction of the real world (a possible configuration).
   - To be in a city.
   - In a room.
   - Hold an object (robot).
   - Position of an object (robot).
   - Switch (open/closed...).

2. ***Actions, state transformation, transition between states:***abstraction of more or less complex real actions.
   - To take or put down an object.
   - Take steps forward.
   - Move from point A to point B
   - Climb, ...

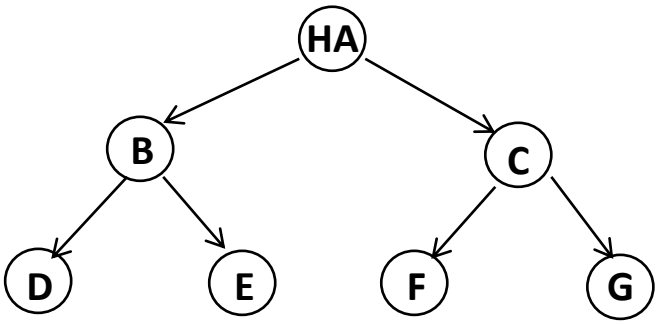
- A research problem is defined by:

  1. ***A starting situation***(initial state e0)
  2. ***A goal to achieve***(final state ef)
  3. ***A research space:***consisting of all possible states.
  4. ***A solution:***is a sequence of actions linking the initial state to a goal state.



- A search space can be represented by a tree, such as:
  - Each node represents a state of this space.

- Two nodes are connected by an arc, if there is a transition between the two states.
- If there is an arc from node B to node D, then B is called the parent of D, and D is the child (son) of B.
- The degree of descent of a node is the number of arcs coming out of that node.



Representation of a search space by a tree

- Node A is the parent of node C
- G is the son of C
- Degree of descent of C is 2

Examples of research problems

1. ***Route search***
   Automatic routes, route guidance, air route planning, routing on computer networks, ...

2. ***Robotics***
   Automatic assembly, autonomous navigation, ...

3. ***Planning and scheduling***
   Schedules, task organization, resource allocation, etc.

Basic Idea of a Research Strategy

Exploration of the state space by generating successor states to the states already explored (also called "expansion" of states). We stop when we have chosen to expand a node that represents a final state.

```
function Tree-Search (problem, strategy) returns a solution, or failure

initialize the search tree using the initial state of problem

loop do
  if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
  if the node contains a goal state then return the corresponding solution
  else expand the node and add the resulting nodes to the search tree
end
```

So, a search algorithm is based on the following concepts:

- Search tree.
- Searching for a node.
- Expansion of a node.
- Search strategy: At each step of the search process, determine which node to "extend".

Managing search tree nodes

- The management of the nodes of the search tree is based on the use of a queue.
- The queue contains search nodes that have not yet been "extended".
- The queue uses the following two procedures:

1. ***INSERT (node, file)***: insert a new node into the queue.
2. EXTRACT (queue): extract the node that occupies the first position in the queue (FIFO strategy).

- The order in which nodes are arranged in the queue defines the search strategy.

Two main families (classes) of search strategies:

1. Blind search strategies

Characterized by:

- Do not use any information about the location of the goal in the search space or the node to be extended.
- Check for each explored state whether it is the desired goal or not.

Blind search strategies include:

1. Breadth-first search
2. Uniform cost research
3. In-depth research
4. Limited depth search
5. Iterative depth-first search (IDS).
6. Two-way search

2. Informed search strategy (heuristics)

- Widely used for problems whose state space is characterized by a combinatorial explosion.
- Based on rules for assessing the probability that one path to the goal is better than the others.
- These rules allow you to estimate the benefits of different paths before taking them.➔Improve the search process.

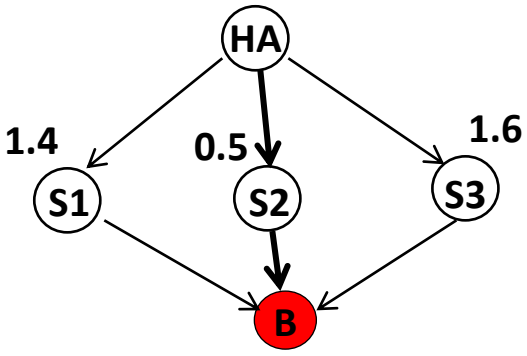
- A heuristic search strategy uses a function called the heuristic function h.

Hey ~~***R***~~

With :

- E: the state space.
- A: the set of real numbers.
- h(n): generally estimates the ratio (cost/benefit) of the chosen path from the current state to the desired goal, i.e. choose the node n which has the value h(n) [Minimum/Maximum] according to the chosen heuristic (the most promising node in the search).

Example:



- ***h(S1)=1.4; h(S2)=0.5; h(S3)= 1.6***

- We note that the search continuation according to S2 is heuristically better than S1, S3.

- Under this family of strategies, several algorithms can be considered:

1. Best First Search BFS
2. Greedy search (greedy/gluttonous search).
3. Hill climbing.
4. Simulated annealing.
5. A* and variants: IDA*, SMA*

Evaluation criteria for a research method

1. ***Completeness:***Does the method guarantee to find a solution if it exists?
2. ***Optimality:***Does the method find the best solution if there are several?
3. ***Time complexity:***How many nodes do you need to produce to find the solution?
4. ***Space complexity:***maximum number of nodes to keep in memory to find the solution?

Noticed :

Time and space complexities are measured in terms of the following parameters:

- ✓ b: branching factor of the search tree (maximum number of successors for a state)
- ✓ d: depth at which the (best) solution node is located
- ✓ m = maximum depth of the search space (can be ∞)

States vs. Nodes

- A state is a representation of an actual problem configuration
- A node is a basic element of the search tree or a structure consisting of the following fields:
  - Parent state.
  - Children's state.
  - Depth
  - Path cost g(x)

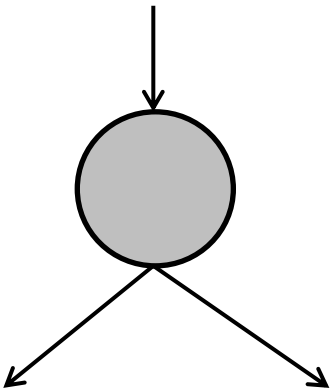
NB: These properties do not exist for states.

State (PUZZLE-8)

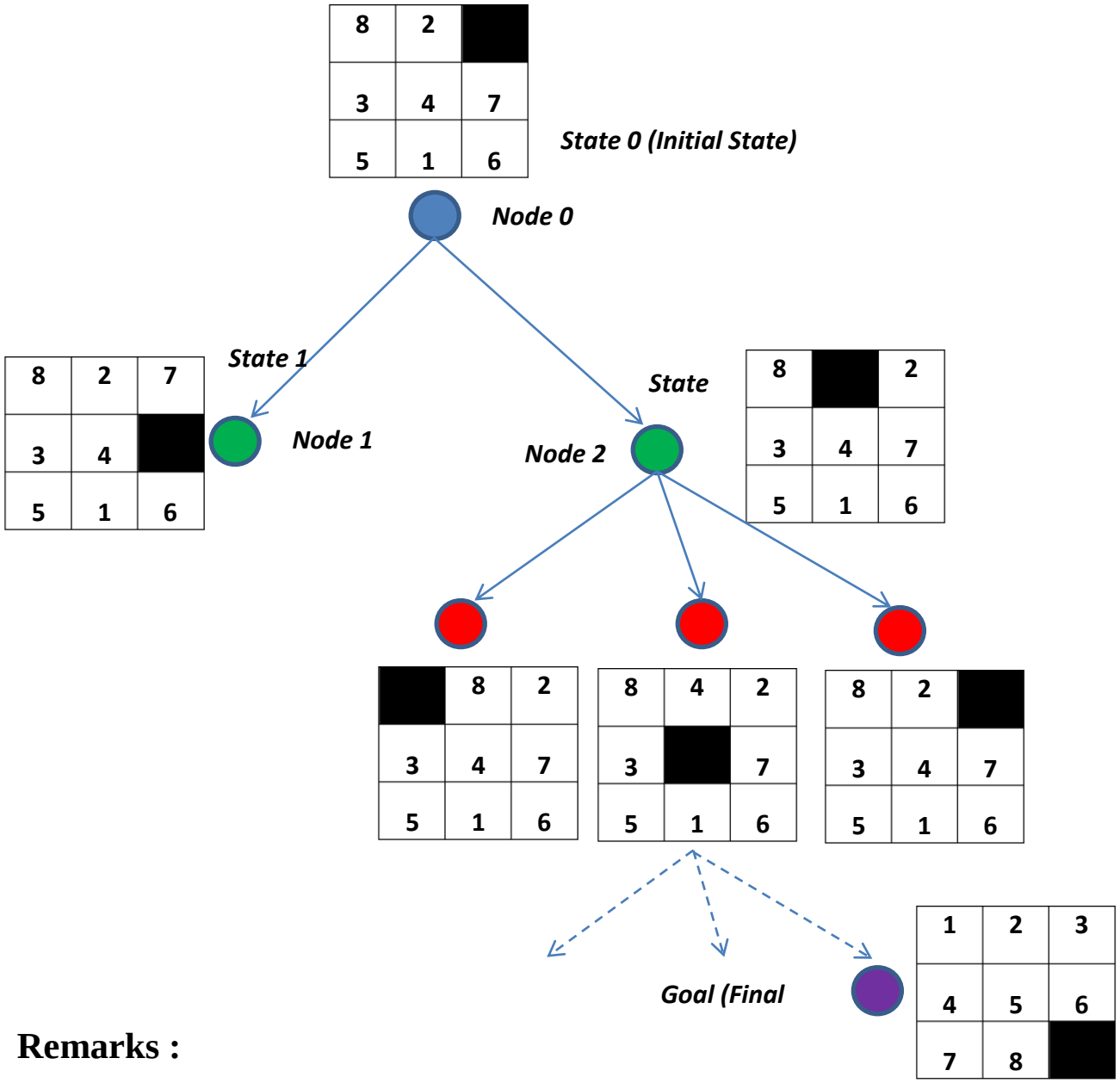
5	4	
6	1	8

Parent

Node Depth=6



Complete example (PUZZLE-8 / Puzzle)



Remarks :

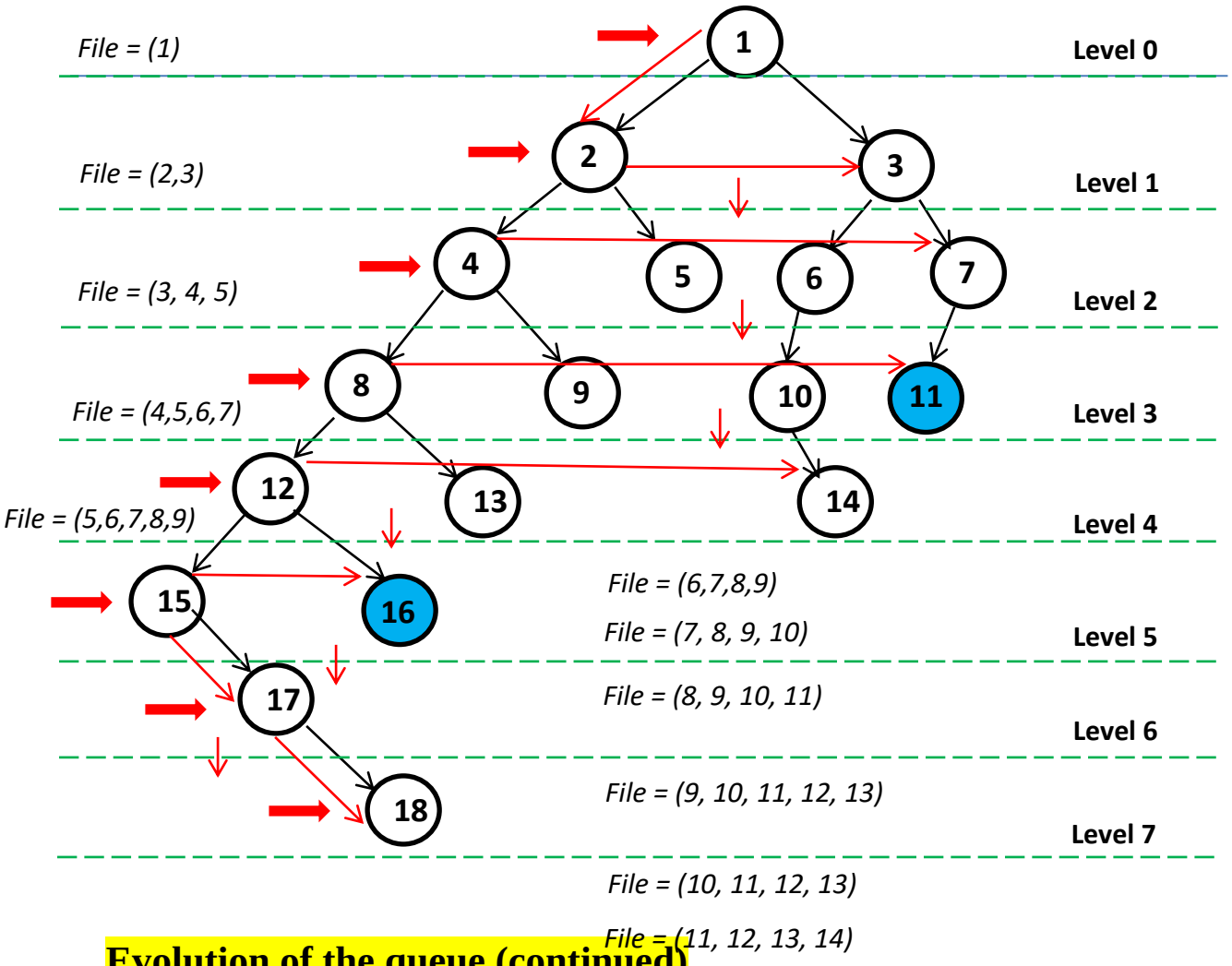
- The search tree can be infinite, while the state space is finite.
- Problems formulated as search problems are NP-hard, e.g., puzzle-(n2-1). They cannot be expected to be solved in less than exponential time in the worst case.

Blind search strategies

1. Breadth-first search

- Also called level search.
- We traverse the search tree from the root, from left to right, from top to bottom (level by level)
- Extend the shallowest knot.

Example :Let the tree represent the search space of a problem



Evolution of the queue (continued)

File = (12, 13, 14) ➔File = (13, 14, 15, 16) ➔File = (14, 15, 16) ➔ File = (14, 15, 16)

➔*File = (15, 16)* ➔*File = (16, 17)* ➔*File = (17)* ➔*File = (18)* ➔*File = ()*

Course sequencing

The order of traversing the nodes: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, 12, 13, 14, 15,**16**, 17, 18

Algorithm performance

1. Completeness: Yes (if b is finite).
2. Time complexity

   $1 + b + b2 + b3 + \ldots + bd + b(bd - 1) = (bd+1 - 1)/(b -1)$

   $= O(bd+1)$ (exponential in d)

3. Space complexity O(bd+1) (all nodes must be kept in memory)
4. Optimality: Yes (if unit cost for each step is 1), but generally not optimal.

Disadvantages

- Greedy in memory space.
- Greedy for time.

Example :

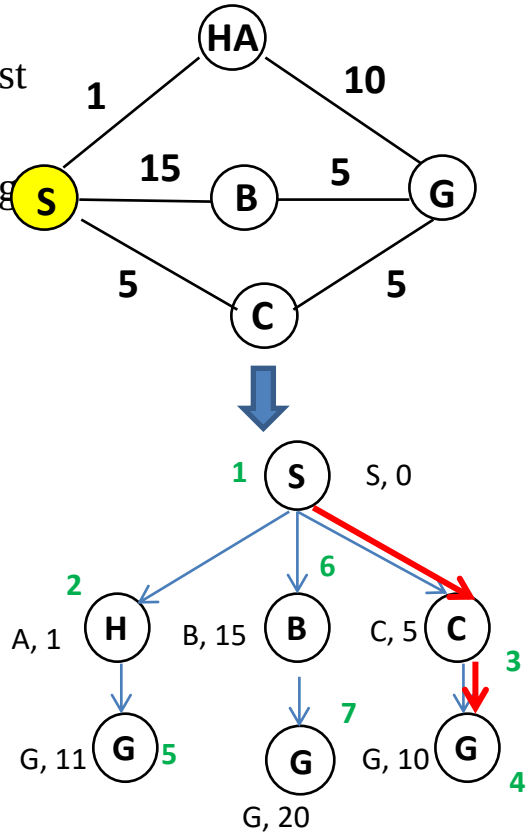
- B (branching factor) = 10; d (tree depth) = 8
- Perform 10000 knots/second
- 1000 bytes/node in memory

depth	Nodes O(bd+1)	Time O(bd+1)	Memory
8	109	109/104 s = 31 h	109x103=1012=1 TB
12	1013	109 s = 35 years	1016bytes=10 Peta.O

2. Uniform cost research

It has the same principle as breadth-first search. Except here that the insertion of the next node in the queue is done in the order of the cost g(x) associated with each transition. That is, take the node with the lowest cost.

1. Insert the initial state with zero cost.
   {(S, 0)} in the queue and building the first Tree node
2. {(A, 1), (C, 5), (B, 15)} always according to the order of cost.
3. {(C, 5), (G, 1+10), (B, 15)}
4. {(G, 5+5), (G, 11), (B, 15)}
5. {(G, 11), (B, 15)}
6. {(B, 15)}
7. {(G, 20)}
8. { }



NB: Oorder of construction of nodes in green

Properties of the algorithm:

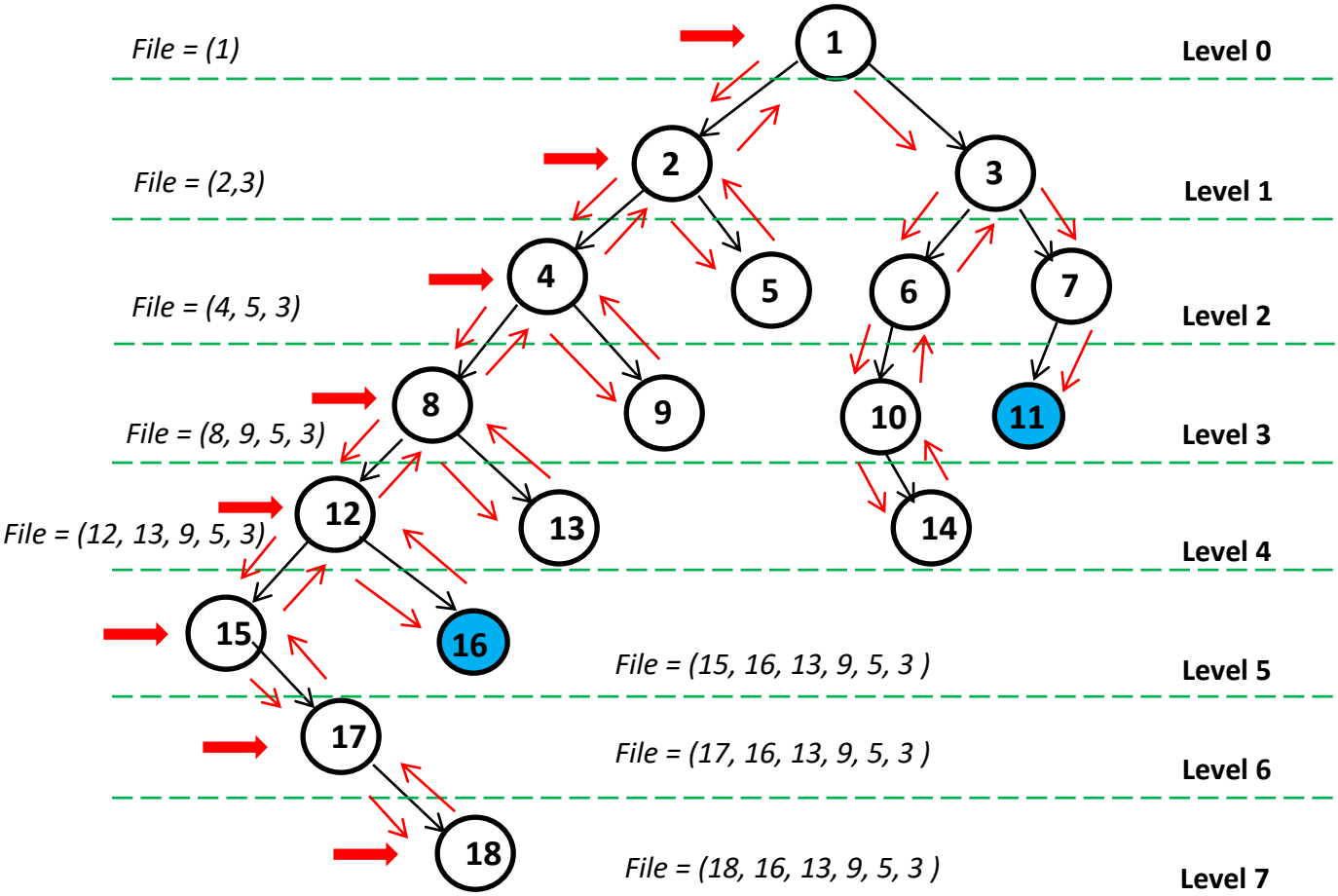
- Equivalent to the previous one (width first) if the cost is still the same.
- Completeness: Yes
- Time complexity: O (bd)
- Space complexity: O (bd)
- Optimality: Yes

3. Depth-first search

Principle:extend the deepest knot each time.

And for the queue, we always insert the new nodes at the beginning, and we remove the accessed node.

Example :



Course sequencing

The order of traversing the nodes: 1, 2, 4, 8, 12, 15, 17, 18,**16,**13, 9, 5, 3, 6, 10, 14, 7,**11**

Algorithm performance

1. Not complete in infinite or loop state spaces
   - It is possible to add a test to detect repetitions
2. Time complexity: O (bm)
   - Very bad if m is much larger than b
3. Space complexity: O (b*m)
   - Linear!
4. Not optimal

Benefits

- Less expensive in memory space.

Disadvantages

1. If the goal is on the right, the search will be long.
2. If there is an infinite branch before a branch containing a goal, the search enters an infinite loop.

Comparison :

For b = 10, d = 12 and 100 bytes/node:

- Search indepthneeds space E=10*12*100=**12 Kbytes (**modest space requirements)
- Search inwidthneeds E=1013*100=1015 =**1000 Terabytes = 1PB**

4. Limited depth search

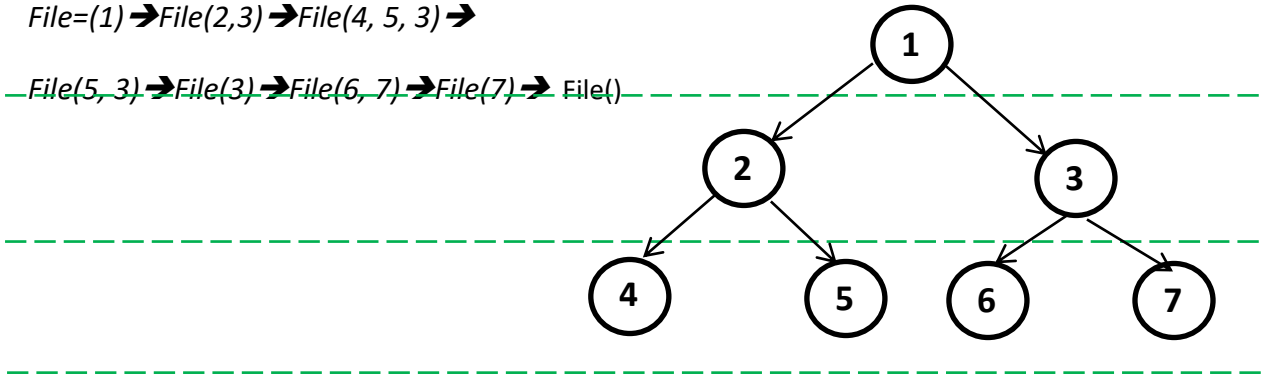
Principle:the same principle as the depth-first search algorithm, but with a limit L on the exploration depth => (1). Cut the tree into slices of L levels.

(2). The search in each slice is a in-depth research.

Example for L = 2

File=(1)➔File(2,3)➔File(4, 5, 3)➔

File(5, 3)➔File(3)➔File(6, 7)➔File(7)➔ File()



Properties

- Completeness: Yes (if L < d)
- Time complexity: O (bL)
- Space complexity: O(b * L)
- Optimality: No

Benefits

1. Avoid the infinite branch problem by limiting a maximum depth.
2. Less expensive in memory space.

5. Iterative depth-first search

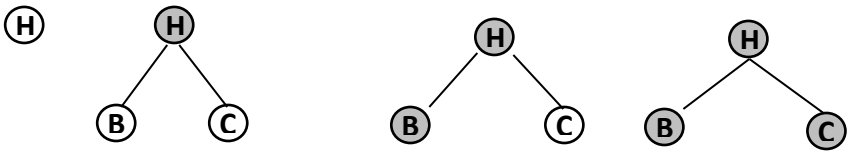
Principle:

- Limited depth, but trying all depths:
  0, 1, 2, 3, etc.
- Avoids the problem of finding a limit for limited depth search.
- Combines the advantages of breadth-first (complete and optimal), but has the space complexity of depth-first

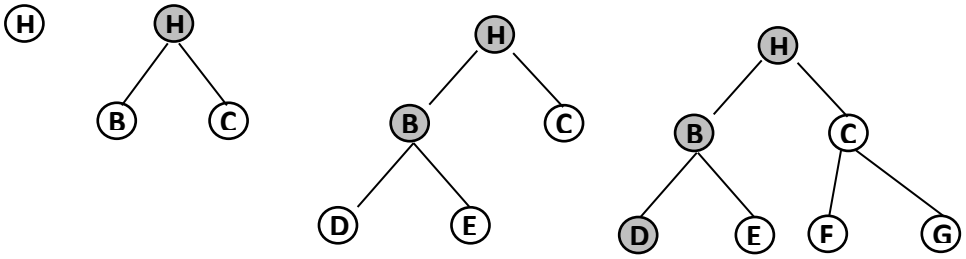
Example :

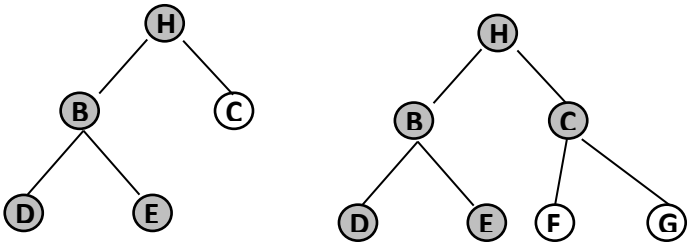
- For L = 0

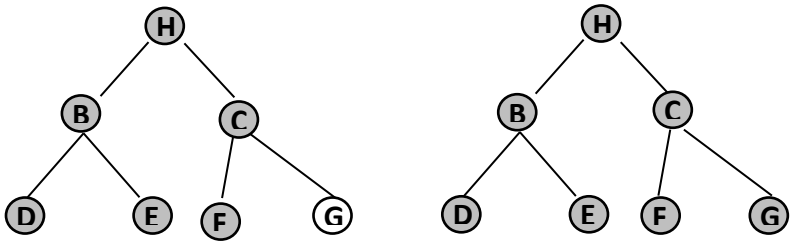
- For L = 1



- For L = 2







Properties

- May seem wasteful as many nodes are extended multiple times.

- But since most of the new nodes are at the lowest level, it is not important to expand the nodes at higher levels several times.
- Complete
- Time complexity:
  (d + 1)b0 + db1 + (d - 1)b2 + ……. + bd = O(bd)
- Space complexity: O(b*d)
- Optimal: Yes (if the cost of each action is 1). Can be modified for a uniform cost strategy.

Summary of Blind Algorithms

Criteria	Width First	Uniform Cost	Depth First	Limited Depth	Iterative Depth
Completeness	Yes If b is finite	Yes	Yes Finite space, no loop	Yes if L < d	Yes
Time	O(bd+1)	O(bd)	O(bm)	O(bL)	O(bd)
Space	O(bd+1)	O(bd)	O(b*m)	O(b*L)	O(b*d)
Optimality (cost of an action = 1)	Yes	Yes	No	No	Yes
Optimality (general case)	No	Yes	No	No	No

Informed search strategies (heuristics)

1. **Seeking the best first BFS**

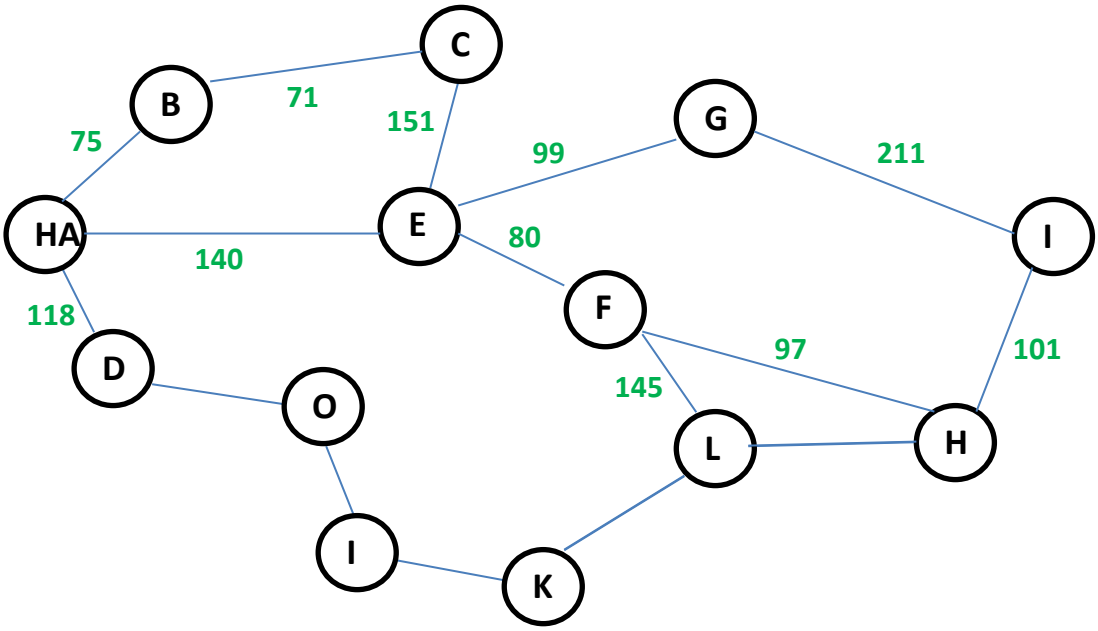
Principle: This algorithm uses a function that measures the utility of a node (using a heuristic function)➔we always take the node which has the best utility value h(n).

Let us also note that:

- The function h(n) is usually an estimate of a cost/benefit ratio.
- For queue management, the most useful node is inserted first.

Example :

Consider the following example with:



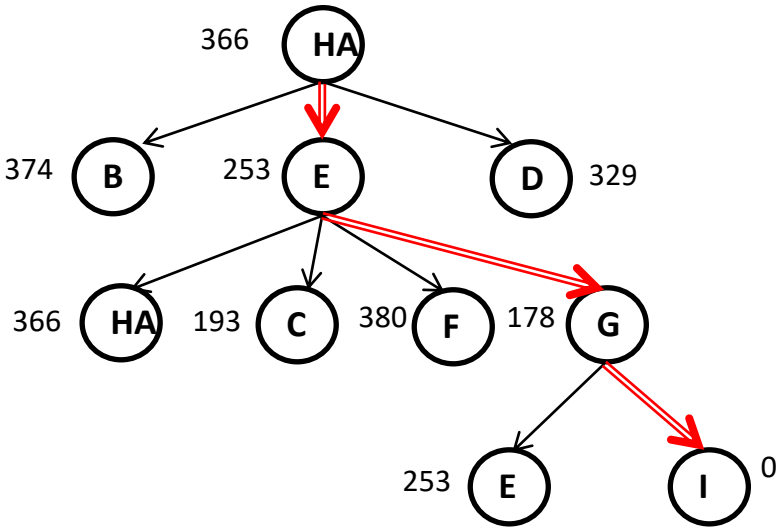
Node	HAS	B	C	D	E	F	G	H	I	I	K	L
h(n)	366	374	193	329	253	380	178	98	0	500	602	615

- The nodes represent cities.
- The values on the arcs represent distances between these cities.
- The values recorded in the table indicate the values of the heuristic function h.
- City A is the initial state and City I is the final state.

- Each time we move towards the node which has the minimum heuristic value h(n)

Issue : How to move from city A (the initial state) to state I (the final state)?

The solution search tree



So the solution is: {A, E, G, I}

Special cases of the best-first search algorithm

a) Greedy-Search

- The simplest variant of the BFS algorithm.

- Uses a heuristic function h(n) that estimates the minimum total cost to move from one state to another.
  (ex: h(n): the minimum direct distance between city n and the destination city).
- The greedy algorithm favors the node that appears to be closer to the final state according to the chosen function.

Properties :

- Incomplete (can go into loops)
- Complete if we add repeated state testing (neglect them).
- Time complexity: O(bm) (good heuristics can improve a lot).
- Space complexity: O(bm).
- Not optimal.

Disadvantages:

- Costly in time.
- Expensive in memory space (keep all nodes in memory).

b) The A* algorithm

Principle:

- Avoid high cost paths.
- Uses a compound heuristic function f(n) = g(n) + h(n)
  With :

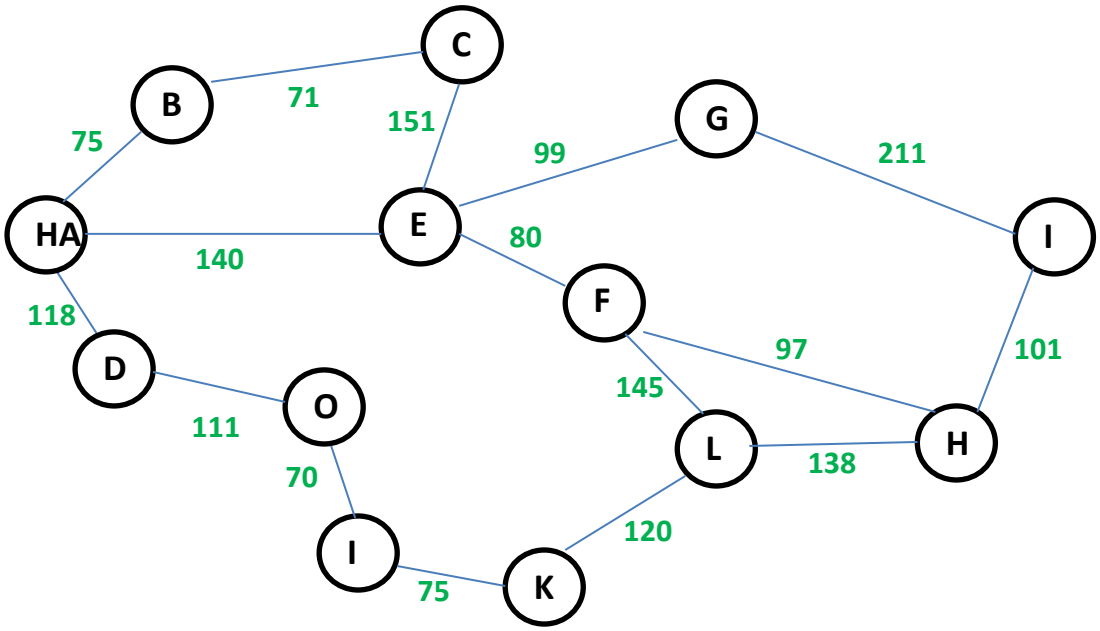
- *g(n)*: the estimated cost so far (from the initial state to the current node, i.e. the cost of the part already explored)
- *h(n)*: estimated cost to go from the current node n to a final state. (the cost of the part not already explored)
- *F(n)*: the total cost

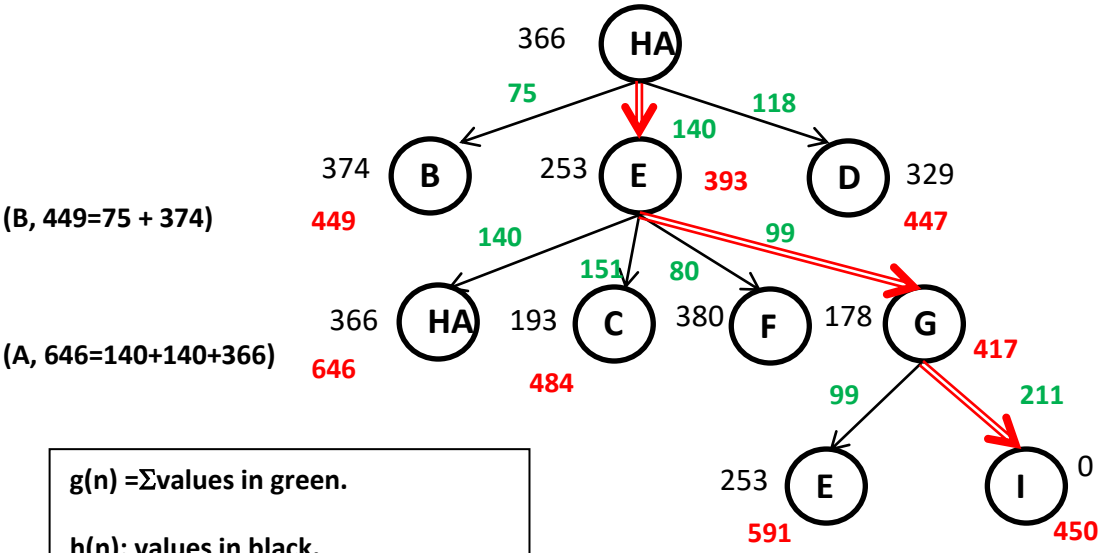
Noticed :A* is always optimal

Properties :

- Complete.
- Time complexity: exponential.
- Space complexity: all nodes are loaded.
- The A* algorithm isoptimal.

Example 1: Reaching city I from city A





g(n) =Σvalues in green.

h(n): values in black.

F(n) = g(n) + h(n)

How to choose between two admissible heuristics?

Let's take the example of PUZZLE-8

Initial state S

5	4	
6	1	8
7	3	2

Final state F

1	2	3
8		4
7	6	5

Two admissible heuristics h1, h2:

- *h1(n)*: number of misplaced pieces
- *h2(n)*:Σdistance from each part to its final position

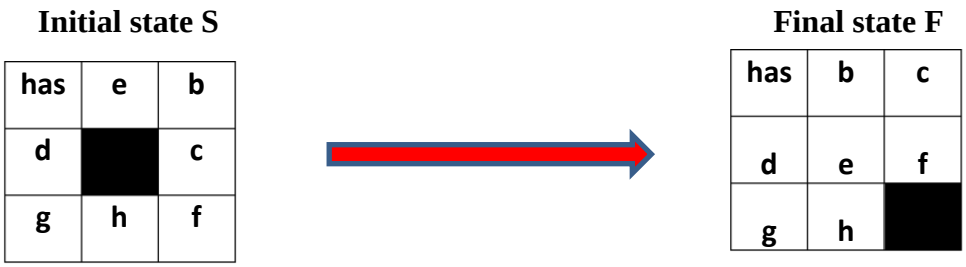
In the previous example we have:

h1(S)= 7; (by comparison between states S and F)
h2(S)= d(1) + d(2) + d(3) + d(4) + d(5) + d(6) + d(7) + d(8)
= 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18;

Remarks:

- We say that h2 dominates h1 if only if h2(n) ≥ h1(n) for all states n.
- A dominant heuristic can significantly reduce the search space.

Complete example



1. Possible operators: move void left L, right R, up H, down B.
2. The same configuration (state) should not be regenerated.
3. We use a compound heuristic function f.
   With f(n) = g(n) + h(n)
   g(n): the length of the path from the initial state e0 to the current state in
   h(n): the number of squares misplaced compared to the final configuration.