Graph-search

3. Graph-search

- It is a search system
  - global workspace: stores the discovered paths (the beginning part of all paths driving from the start node: this is the search graph) and separately records the last nodes of all discovered paths (they are called open nodes)
    - initial value: start node
    - termination condition: a goal node must be expanded or there is no open node
  - searching rules: expand open nodes
  - control strategy selects an open node to be expanded based on an evaluation function

3.1. General graph-search

- search graph (G) : the subgraph of the representation graph that has been discovered
- set of open nodes ($OPEN$) : the nodes that are waiting for their expansions because their successors are not known or not well-known
- evaluation function ($f:OPEN\to\mathbb{R}$) : helps to select the appropriate open node to be expanded.

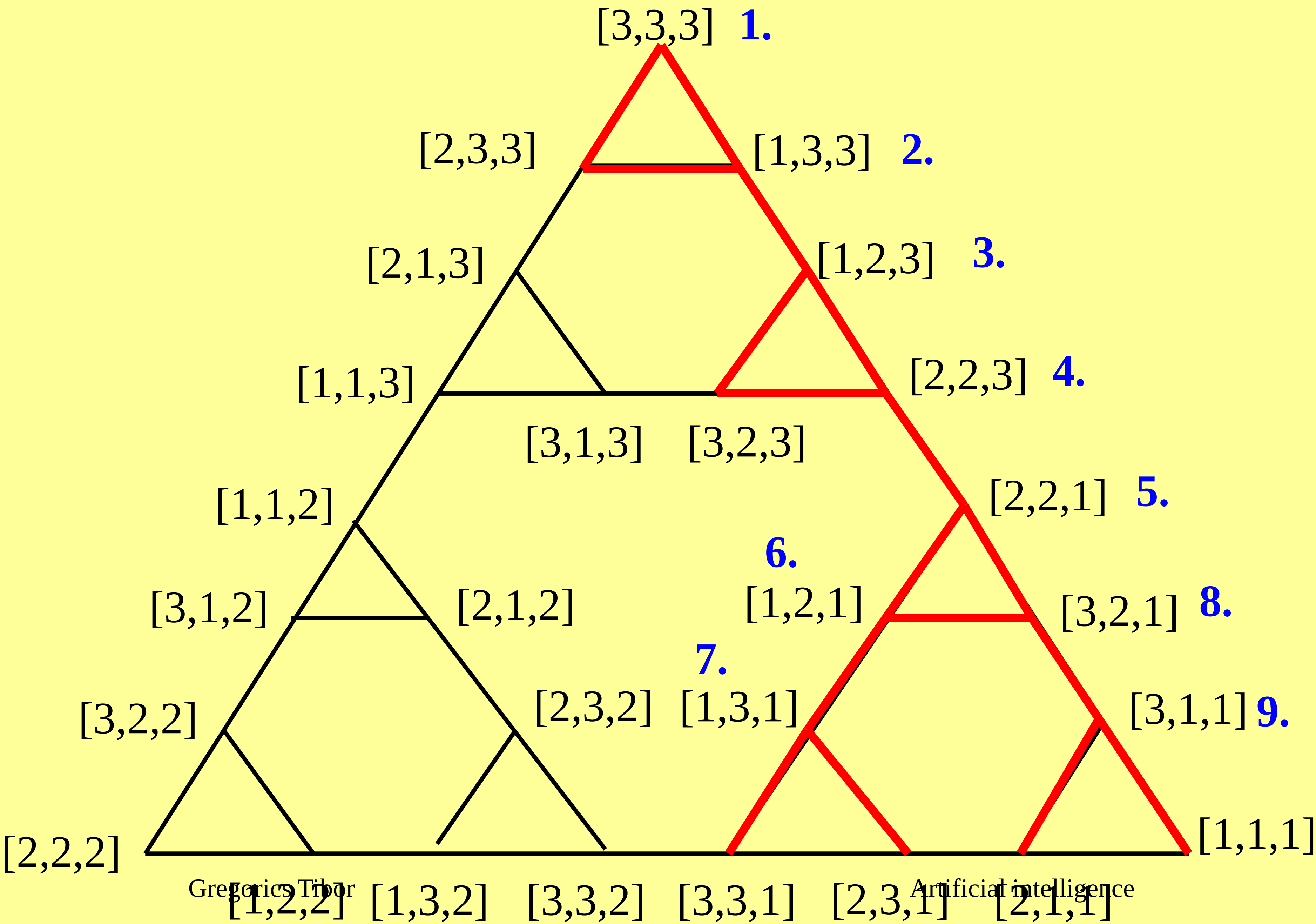
DATA := *initial value*
while ¬ *termination condition*(DATA) **loop**
    SELECT R FROM *rules that can be applied*
    DATA := R(DATA)
endloop

Naive version

Procedure *GK0*

1. $G := (\{start\}, \emptyset)$: $OPEN := \{start\}$
2. **loop**
3.   **if** *empty*($OPEN$) **then return** *no solution*
4. $n := min_f(OPEN)$
5.   ***if*** *goal*(n) **then return** *there is a solution*
6. $OPEN := OPEN - \{n\} \cup (\Gamma(n) - \pi(n))$
7. $G := G \cup \{(n,m) \in A \mid m \in \Gamma(n) - \pi(n)\}$
8. **endloop**

end

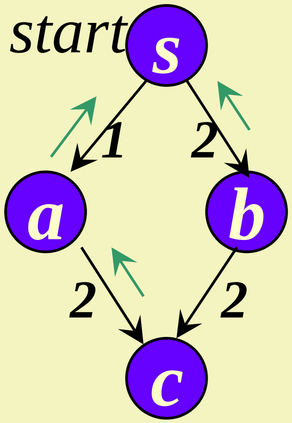

[3,3,3]
1.
[2,3,3]
[1,3,3]
2.
[2,1,3]
[1,2,3]
3.
[1,1,3]
[2,2,3]
4.
[3,1,3]
[3,2,3]
[1,1,2]
[2,2,1]
5.
6.
[3,1,2]
[2,1,2]
[1,2,1]
[3,2,1]
8.
7.
[3,2,2]
[2,3,2]
[1,3,1]
[3,1,1]
9.
[2,2,2]
[1,1,1]
[1,2,2]
[1,3,2]
[3,3,2]
[3,3,1]
[2,3,1]
[2,1,1]

Objections to the naïve version

- A method is needed to take the solution path out from the global workspace after successful termination.
  - A unique path from the start node to each node should be recorded.
- The optimal solution is not guaranteed (neither the solution).
  - The cost of the recorded path should be stored for each discovered node.
- The cycles cause fault.
  - Recording the cheapest path ignores the paths including cycles.

Functions of the graph-search

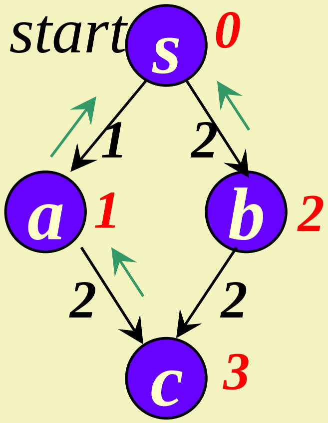
- $\pi: N \to N$ ***parent pointer function***
  - $\pi(m)$ = one parent of m in G, $\pi(start) = nil$
    - π determines a spanning tree in G and helps to take the solution path out from G after successful termination
    - <u>If only</u> the $\pi(m)$ always showed an optimal path $start \to m$ in G when the node m is generated



- $g: N \to \mathbb{R}$ ***cost function***

Functions of the graph-search

- $\pi: N \rightarrow N$ ***parent pointer function***
  - $\pi(m)$ = one parent of m in G, $\pi(start) = nil$
    - π determines a spanning tree in G and helps to take the solution path out from G after successful termination
    - <u>If only</u> the $\pi(m)$ always showed an optimal path $start \rightarrow m$ in G when the node m is generated



- $g: N \rightarrow \mathbb{R}$ ***cost function***
  - $g(m) = c^{\alpha}(start,m)$ − cost of a discovered path $\alpha \in \{start \rightarrow m\}$
  - <u>If only</u> $g(m)$ gave the cost of the path $start \rightarrow m$ that is shown by π when the node m is generated.

The node m is ***correct*** if $g(m)$ and $\pi(m)$ are consistent and optimal,
i.e. $g(m) = c^{\pi}(start, m)$ and $c^{\pi}(start, m) = \mathbf{min}_{\alpha \in \{start \rightarrow m\} \cap G}\, c^{\alpha}(start, m)$.
G is correct if its nodes are correct.

Maintaining the correctness when a node is generated

- Initially: $\pi(start) := nil$, $g(start) := 0$
- for all $m \in \Gamma(n)$ (after expansion of the node n):
  - 1. **if** m is a new node ($m \notin G$) **then**

    $\pi(m) := n$, $g(m) := g(n)+c(n,m)$

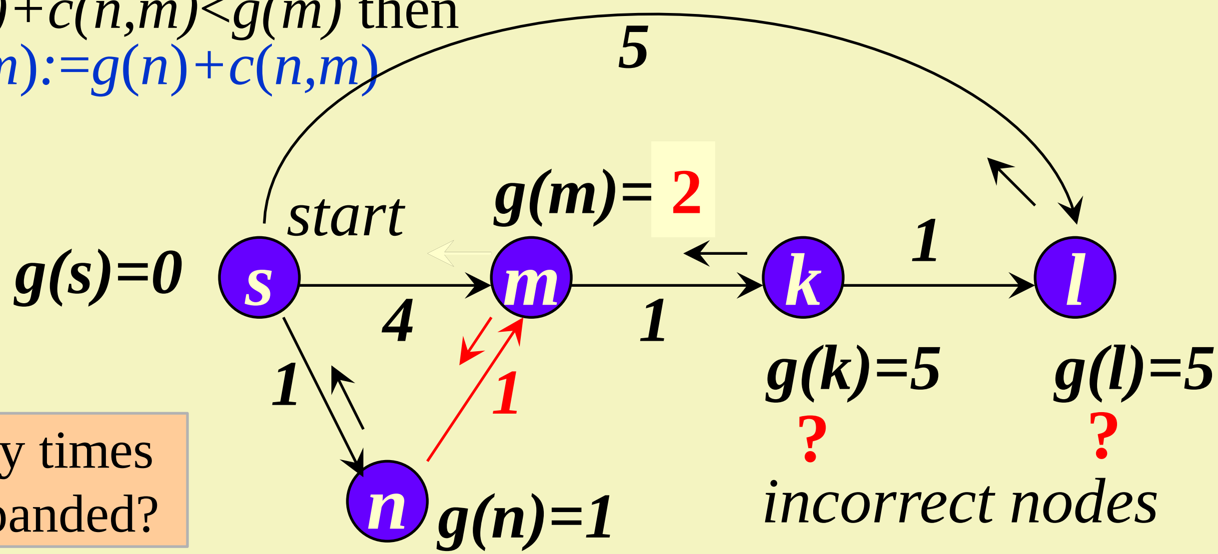
    $OPEN := OPEN \cup \{m\}$
  - 2. **if** m is an old node to which a cheaper path has been found ($m \in G$ and $g(n)+c(n,m)<g(m)$) **then**

    $\pi(m) := n$, $g(m) := g(n)+c(n,m)$
  - 3. **if** m is an old node to that a not cheaper path has been found ($m \in G$ and $g(n)+c(n,m) \geq g(m)$) **then**

    DO NOTHING

The correctness of the search graph is not even ensured

If $m \in G$ and $g(n)+c(n,m)<g(m)$ then
$\pi(m):=n, \; g(m):=g(n)+c(n,m)$



Danger: how many times will a node be expanded?

- What should we do with the descendants of the node to which a better path has been found?
  1. The pointers and costs of all descendants of the node *m* might be modified using some traversal method.
  2. Such a case could be avoided with a good evaluation function.
  3. Do not care of the correctness just put the node *m* back into *OPEN*.

Algorithm of general graph-search

DATA := *initial value*
while $\neg$ *termination condition*(DATA) **loop**
    SELECT R FROM *rules that can be applied*
    DATA := R(DATA)
endloop

$G := (\{start\}, \emptyset)$; $OPEN := \{start\}$; $\pi(start) := nil$; $g(start) := 0$
loop
    if $empty(OPEN)$ **then return** *no solution*
    $n := min_f(OPEN)$
    if $goal(n)$ **then return** solution (n, π)
    $OPEN := OPEN - \{n\}$
    for $\forall m \in \Gamma(n) - \pi(n)$ **loop**
        if $m \notin G$ or $g(n)+c(n,m)<g(m)$ **then**
            $\pi(m) := n$; $g(m) := g(n)+c(n,m)$; $OPEN := OPEN \cup \{m\}$
    endloop
    $G := G \cup \{(n,m) \in A \mid m \in \Gamma(n)\}$
endloop

Execution and outcomes

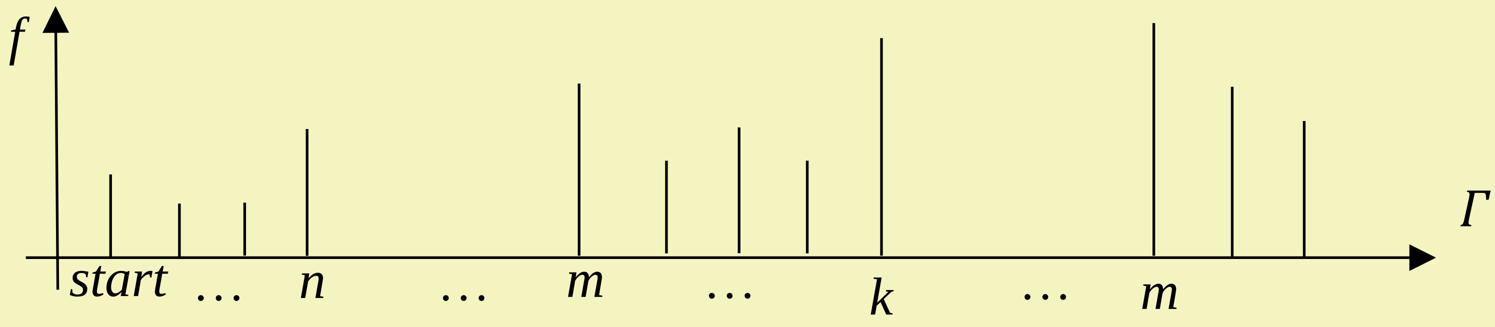
It can be proved:

- Each node is expanded only finite times in a δ-graph.
- The general graph-search always terminates in a finite δ-graph.
- The general graph-search finds a solution in a finite δ-graph if there exists a solution.

Decreasing evaluation function

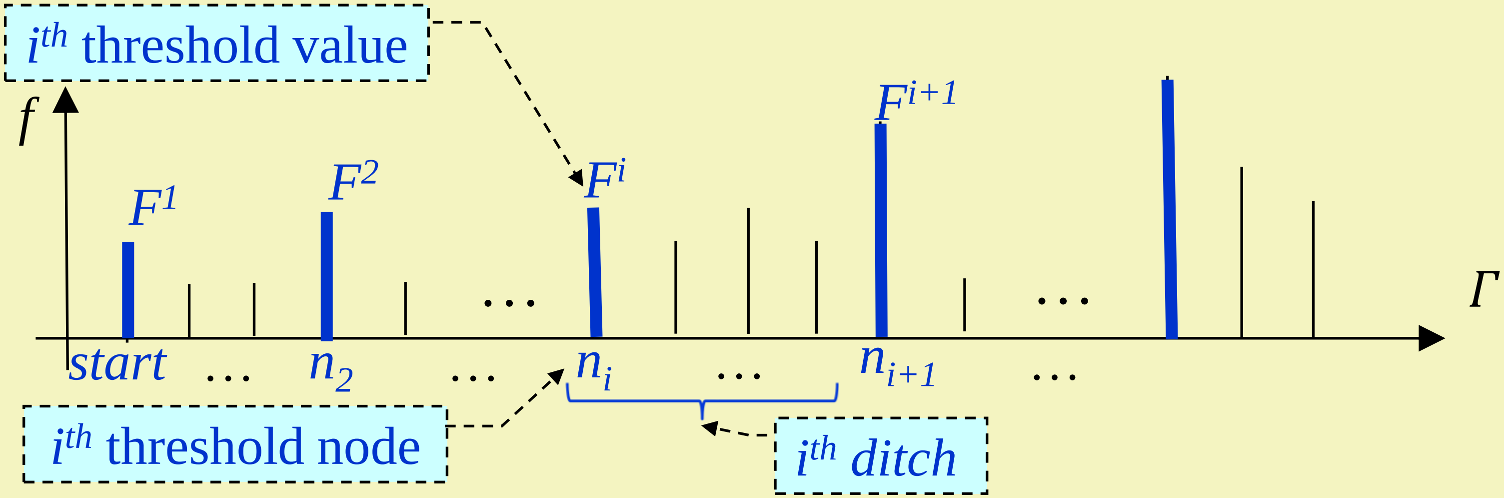
- An evaluation function f:$OPEN \to \mathbb{R}$ is decreasing if for all nodes n ($n \in N$) the $f(n)$ never increases but always decreases when a cheaper path has been found to the node n.
  - For example the function g has got this property.
- It can be proved that the correctness of the search graph is re-established automatically over and over again if the graph-search uses a decreasing evaluation function.

Execution diagram



- The expanded nodes with their evaluation function values are enumerated in order of their expansions (the same node can occur several times).

About the correctness of the search graph with decreasing evaluation function



- A monotone increasing subsequence F^i ($i=1,2,\ldots$) can be selected from the values of the diagram so that it starts with the first value and each value is followed by the closest non smaller value.
- The graph-search with a decreasing evaluation function
  - records a correct search graph at expansion of a threshold node
  - never expands incorrect nodes

3.2. Famous graph-search algorithms

- What kinds of evaluation functions are there?

Non-informed

- depth-first graph-search
- breadth-first graph-search
- uniform-cost graph-search

Heuristic

- look-forward graph-search
- algorithm A, A^*, A^c
- algorithm A^{**}, B

- The tie-breaking rules (secondary evaluation functions) may contain heuristics even in non-informed graph-search.

Non-informed graph-search

Algorithm	Definition	Results
depth-first graph-search	$f = -g$, $c(n,m) = 1$	no special property in infinite δ-graphs a depth bound is needed
breadth-first graph-search	$f = g$, $c(n,m) = 1$	• finds the shortest (not the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once
uniform-cost graph-search	$f = g$	• finds optimal (the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once

Non-informed graph-search

not identical to the backtracking search that is called as depth-first search

Algorithm	**Definition**	**Results**
depth-first graph-search	$f = -g$, $c(n,m) = 1$	no special property in infinite δ-graphs a depth bound is needed
breadth-first graph-search	$f = g$, $c(n,m) = 1$	• finds the shortest (not the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once
uniform-cost graph-search	$f = g$	• finds optimal (the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once

Non-informed graph-search

not identical to the backtracking search that is called as depth-first search

Algorithm	**Definition**	**Results**
depth-first graph-search	$f = -g$, $c(n,m) = 1$	no special property in infinite δ-graphs a depth bound is needed
breadth-first graph-search	$f = g$, $c(n,m) = 1$	• finds the shortest (not the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once
uniform-cost graph-search	$f = g$	• finds optimal (the cheapest) solution if there exists one even in infinite δ-graph • each node is expanded at most once

similar to Dijkstra's shortest path algorithm

Heuristics in graph-search

- The heuristic function $h\!:\!N \rightarrow \mathbb{R}$ estimates the cost of the cheapest path from a node to the goal.

- $h(n) \approx h^*(n)$ $\quad h^*\!:\!N \rightarrow \mathbb{R}$

remaining optimal cost from n to any goal node of T :
$h^*(n) = c^*(n,T)$
optimal cost from n to any node of M:
$c^*(n,M) := \min_{m \in M} c^*(n,m)$
optimal cost from n to m :
$c^*(n,m) := \min_{\alpha \in \{n \rightarrow m\}} c^{\alpha}(n,m)$

- Examples:
  - *8*-puzzle : *W*, *P*
  - *0* (zero function) ~ fake heuristic function

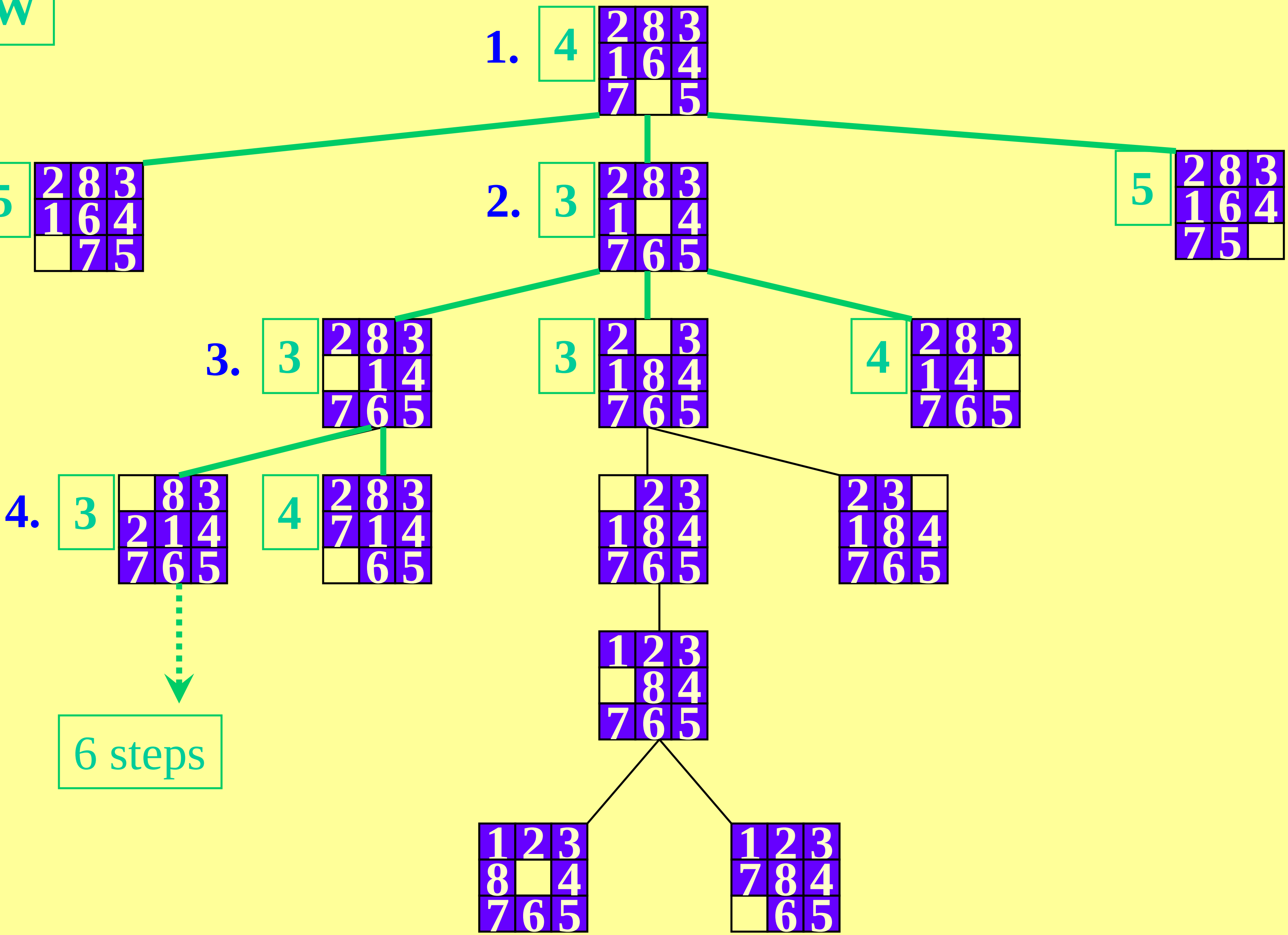
Properties of heuristic function

- Properties:
  - ***Non-negative***: $h(n) \geqq 0$ $\quad \forall n \in N$
  - ***Admissible***: $h(n) \leqq h^*(n)$ $\quad \forall n \in N$
  - ***Monotone restriction***: $h(n) - h(m) \leqq c(n,m)$ $\quad \forall (n,m) \in A$ (consistent)
- Remarks
  - *8*-puzzle : *W*, *P* are non-negative, admissible and monotone.
  - Zero function is non-negative, admissible and monotone.
  - If *h* is monotone and gives zero on goal, then it is admissible.

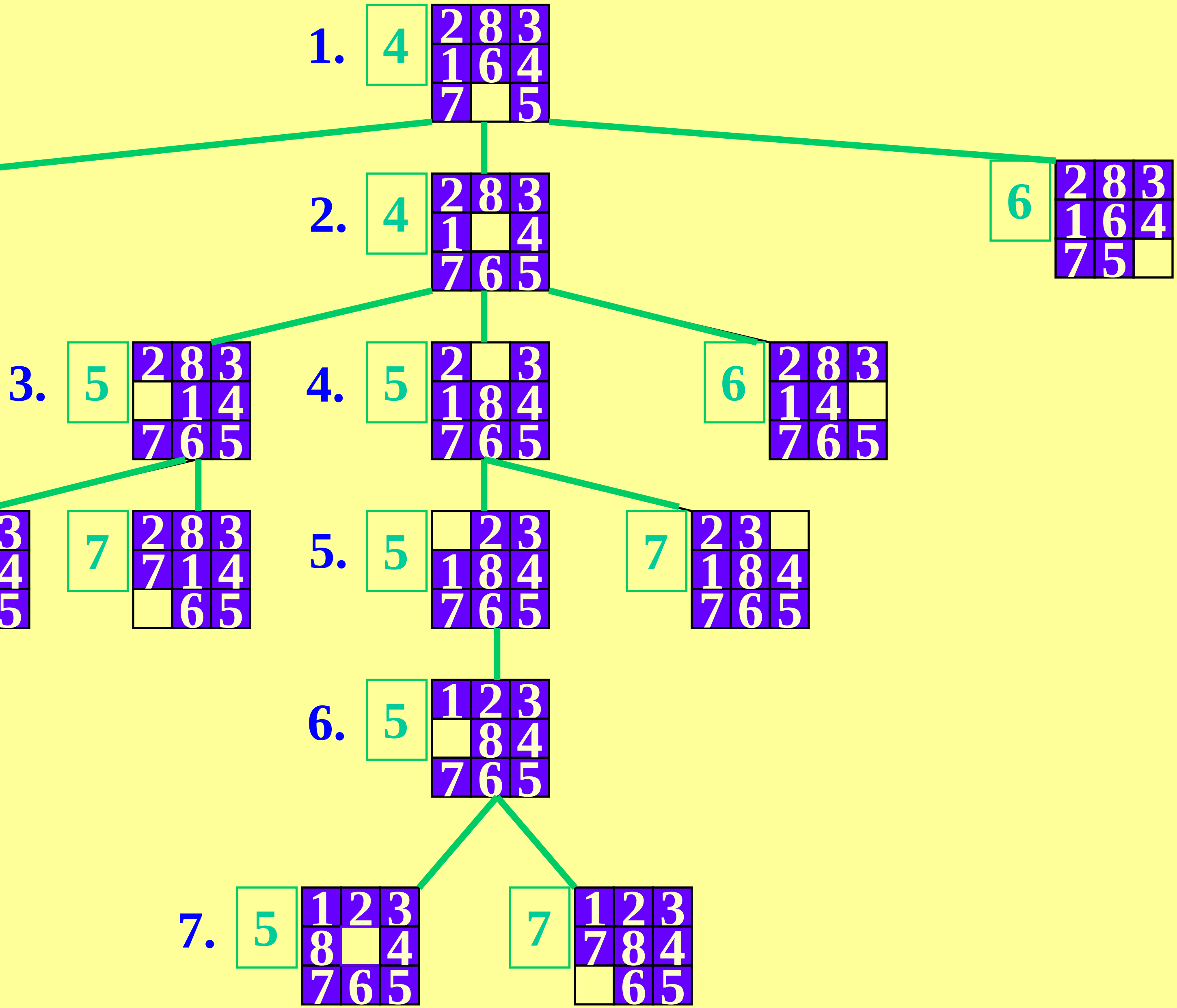
Heuristic graph-search

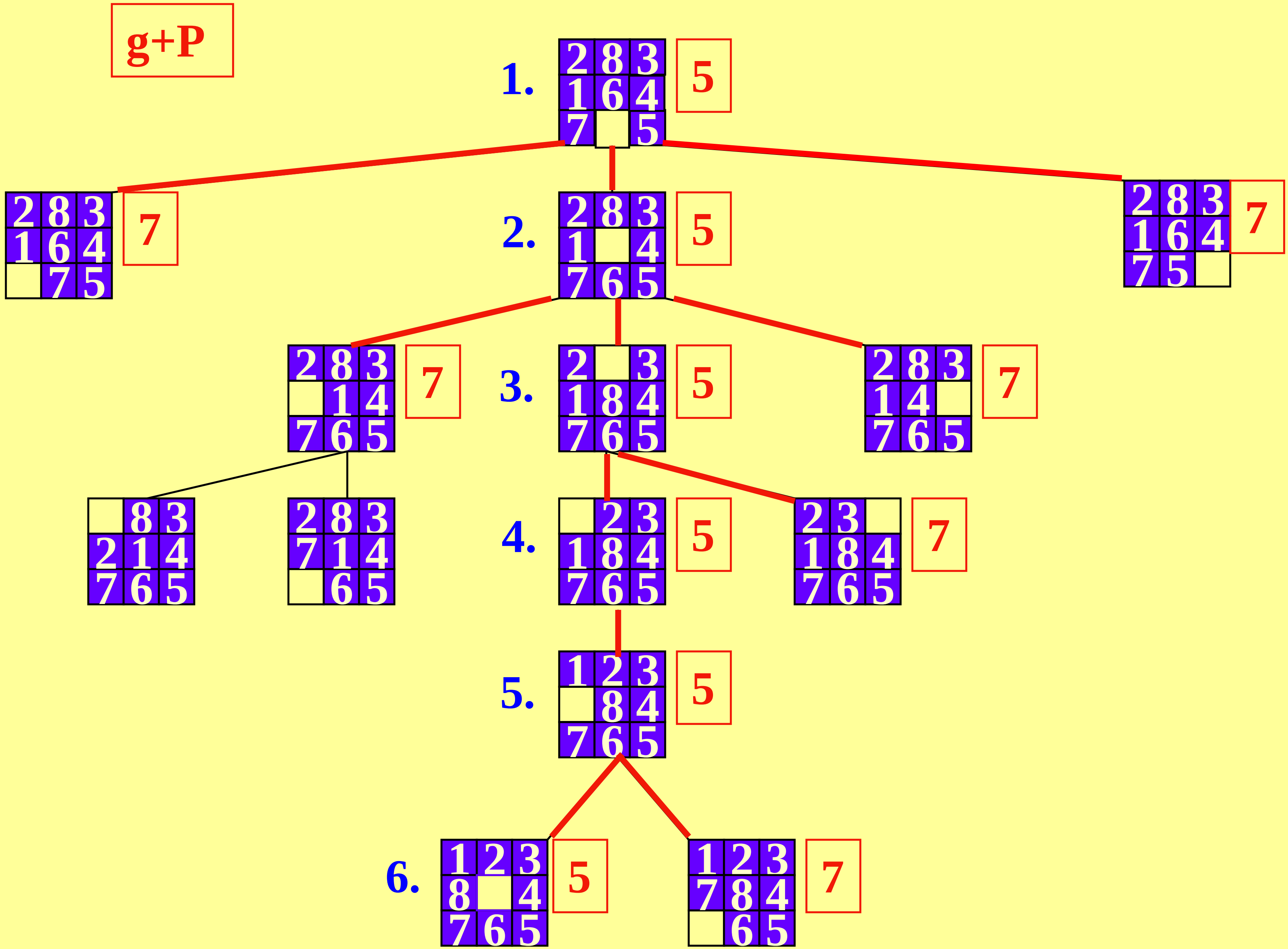
Algorithm	Definition	Results
look-forward graph-search	$f = h$	no special property
algorithm A	$f=g+h$, $h \geqq 0$	• finds solution if there exists one (even in infinite δ-graph)
algorithm A^*	$f=g+h$, $h \geqq 0$, $h \leqq h^*$	• finds optimal solution if there exists one (even in infinite δ-graph)
algorithm A^c	$f=g+h$, $h \geqq 0$, $h \leqq h^*$, $h(n)-h(m) \leqq c(n,m)$	• finds optimal solution if there exists one (even in infinite δ-graph) • expands a node at most once

W



g+W




g+P
1.
2.
3.
4.
5.
6.