3. Graph-search

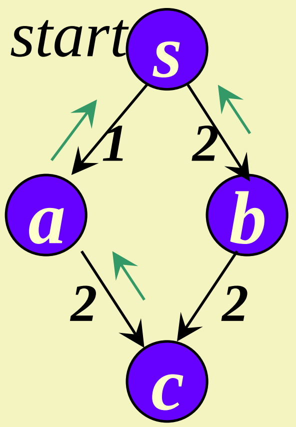
- It is a search system
  - global workspace: stores the discovered paths (the beginning part of all paths driving from the start node: this is the search graph) and separately records the last nodes of all discovered paths (they are called open nodes)
    - initial value: start node
    - termination 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:$OPEN \rightarrow \mathbb{R}$) : helps to select the appropriate open node to be expanded.

Functions of the graph-search

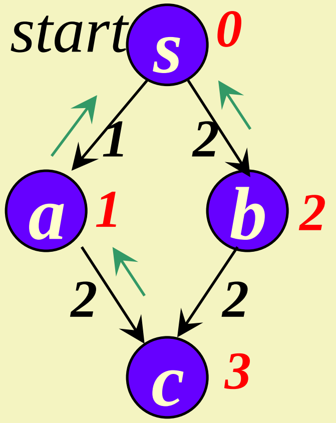
- $\pi: N \to N$ ***parent pointer function***
  - $\pi(m)$ = one parent of m in G, $\pi(start) = nil$
    - π determines a spanning tree in G and helps to take the solution path out from G after successful termination
    - <u>If only</u> the $\pi(m)$ always showed an optimal path $start \to m$ in G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

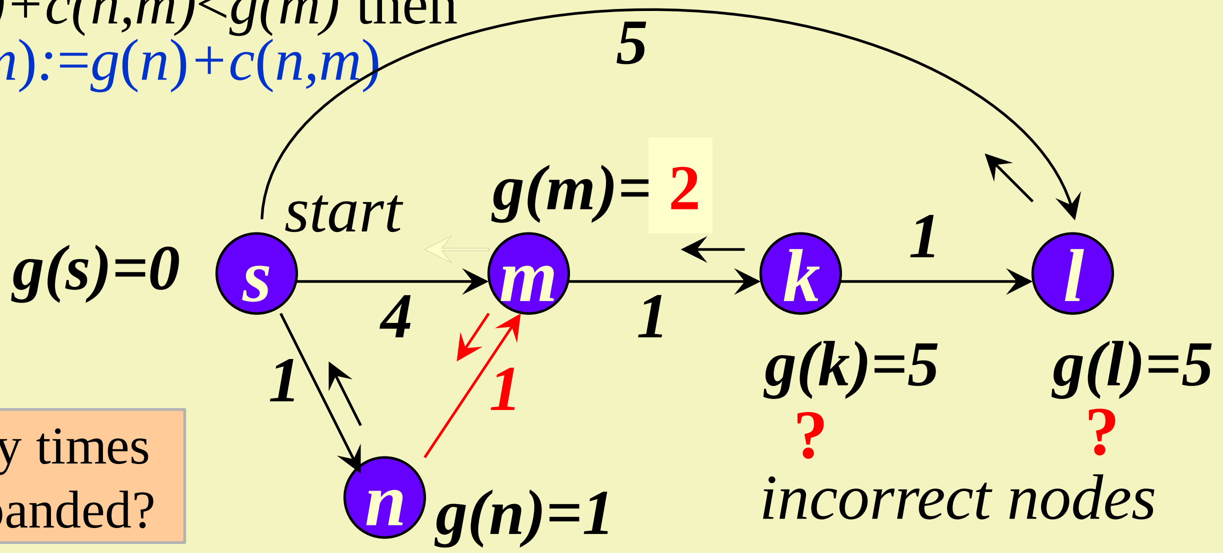
Maintaining the correctness when a node is generated

- 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∉G) then
        π(m) := n,  g(m) := g(n)+c(n,m)
        OPEN := OPEN ∪ {m}
2. if m is an old node to that a cheaper path has been found
   (m∈G and g(n)+c(n,m)<g(m)) then
        π(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∈G and g(n)+c(n,m)≥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*.

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

Algorithm of general graph-search

```
1.  $G := (\{start\}, \emptyset)$ ; $OPEN := \{start\}$ ; $\pi(start) := nil$ ; $g(start) := 0$
2.  loop
3.      if $empty(OPEN)$ then return no solution
4.      $n := min_f(OPEN)$
5.      if $goal(n)$ then return solution $(n, \pi)$
6.      $OPEN := OPEN - \{n\}$
7.      for $\forall m \in \Gamma(n) - \pi(n)$ loop
8.          if $m \notin G$ or $g(n)+c(n,m)<g(m)$ then
9.              $\pi(m) := n$ ; $g(m) := g(n)+c(n,m)$ ; $OPEN := OPEN \cup \{m\}$
10      endloop
11.     $G := G \cup \{(n,m) \in A \mid m \in \Gamma(n)\}$
12. 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.

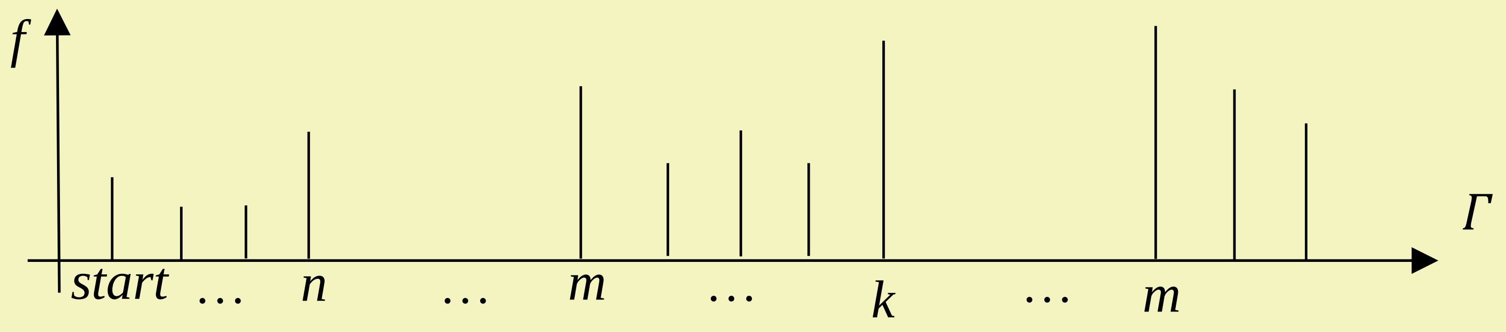
- The general graph-search finds a solution in a finite δ-graph if there exists a solution.

Decreasing evaluation function

- An evaluation function $f:OPEN\rightarrow\mathbb{R}$ 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.

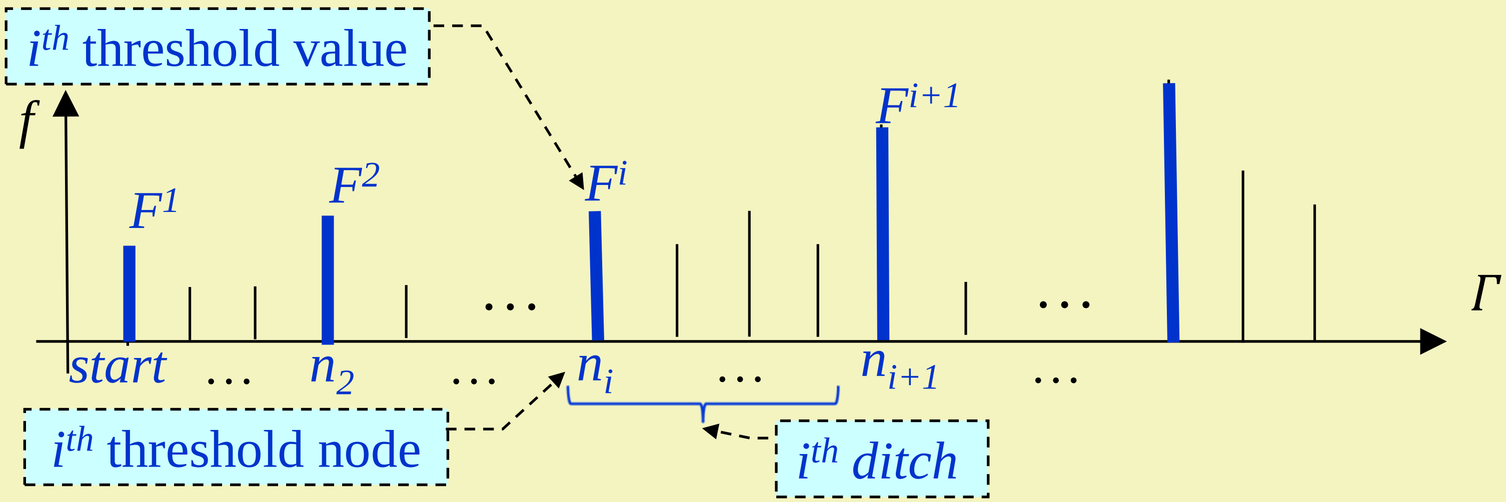
- It can be proved that the correctness of the search graph is re-established automatically over and over again if the graph-search uses a decreasing evaluation function.

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 algorithm

- What kinds of evaluation functions are there?

Non-informed	Heuristic
depth-first graph-search	look-forward graph-search
breadth-first graph-search	algorithm A, A^*, A^c
uniform-cost graph-search	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 $h{:}N \to \mathbb{R}$ estimates the cost of the cheapest path from a node to the goal.

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

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 :
$c^*(n,m) := \min_{\alpha\in\{n\to m\}} c^\alpha(n,m)$

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

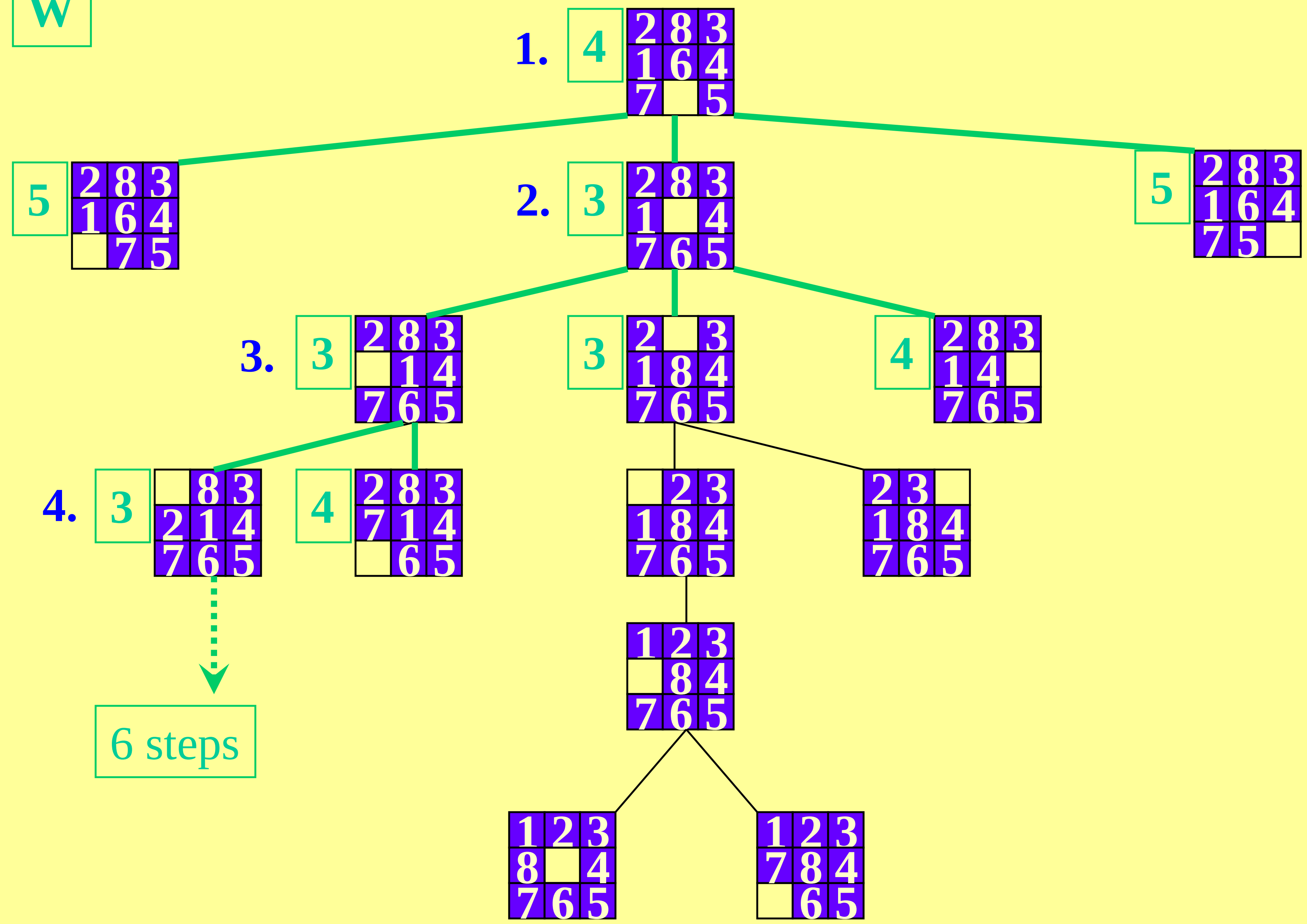
Properties of heuristic function

- Famous 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.

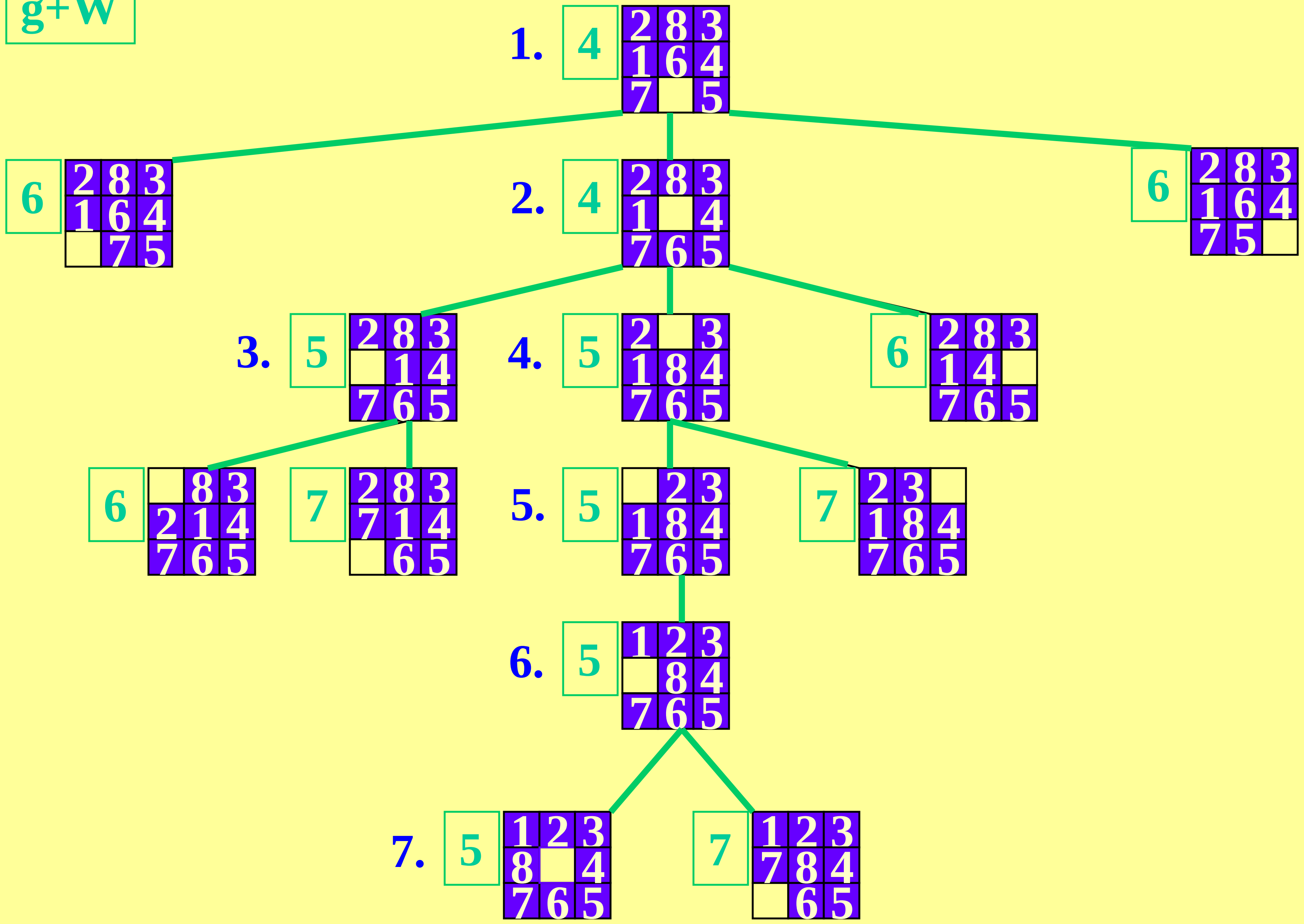
Heuristic graph-search

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

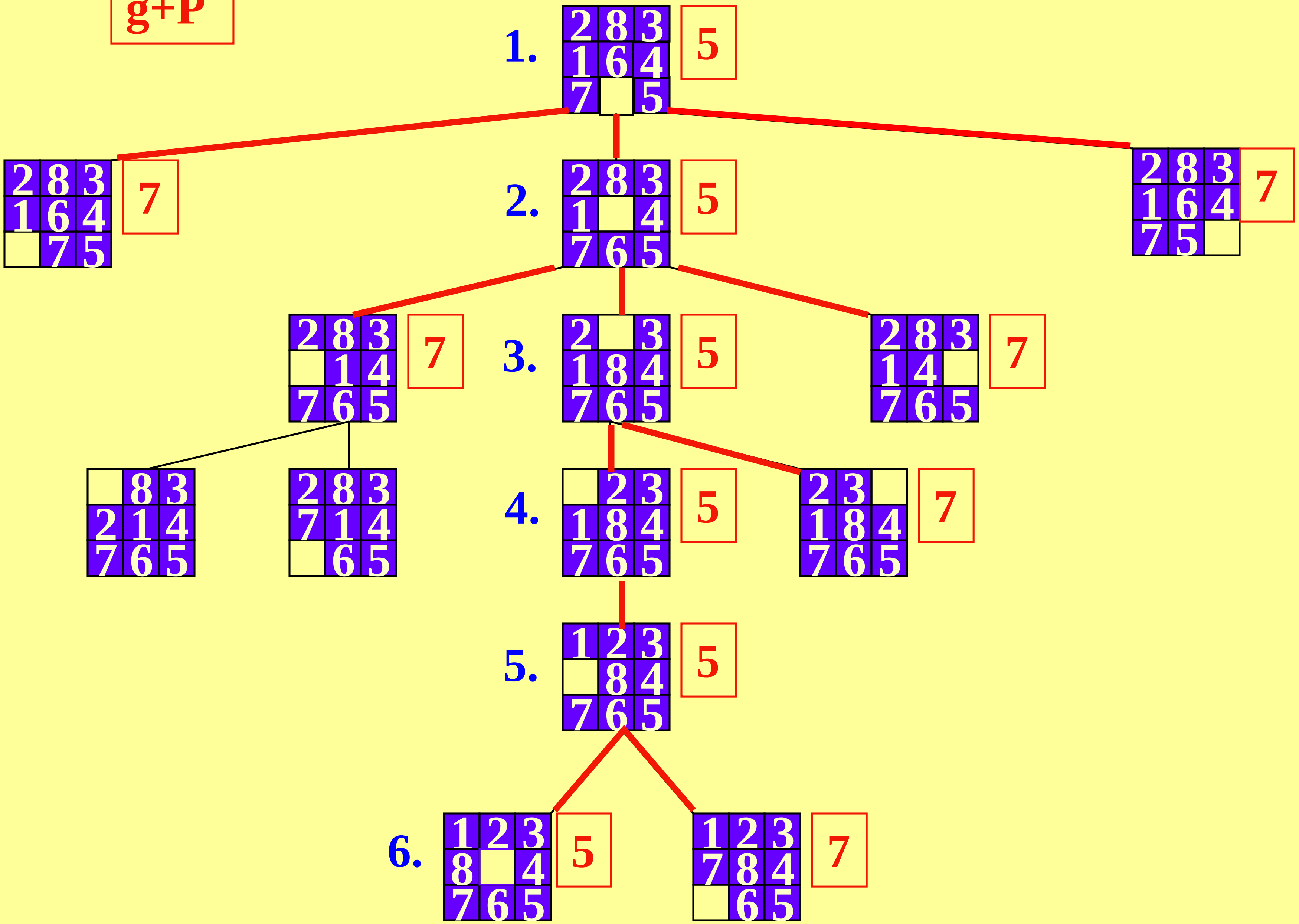

1.
4
2 8 3
1 6 4
7 5
5
2 8 3
1 6 4
7 5
2.
3
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
3.
3
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
4.
3
8 3
2 1 4
7 6 5
4
2 8 3
7 1 4
6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
6 steps
1 2 3
8 4
7 6 5
1 2 3
8 4
7 6 5
1 2 3
7 8 4
6 5

g+W



1.
4
2 8 3 / 1 6 4 / 7 _ 5
6
2 8 3 / 1 6 4 / _ 7 5
2.
4
2 8 3 / 1 _ 4 / 7 6 5
6
2 8 3 / 1 6 4 / 7 5 _
3.
5
2 8 3 / _ 1 4 / 7 6 5
4.
5
2 _ 3 / 1 8 4 / 7 6 5
6
2 8 3 / 1 4 _ / 7 6 5
6
_ 8 3 / 2 1 4 / 7 6 5
7
2 8 3 / 7 1 4 / _ 6 5
5.
5
_ 2 3 / 1 8 4 / 7 6 5
7
2 3 _ / 1 8 4 / 7 6 5
6.
5
1 2 3 / _ 8 4 / 7 6 5
7.
5
1 2 3 / 8 _ 4 / 7 6 5
7
1 2 3 / 7 8 4 / _ 6 5

g+P

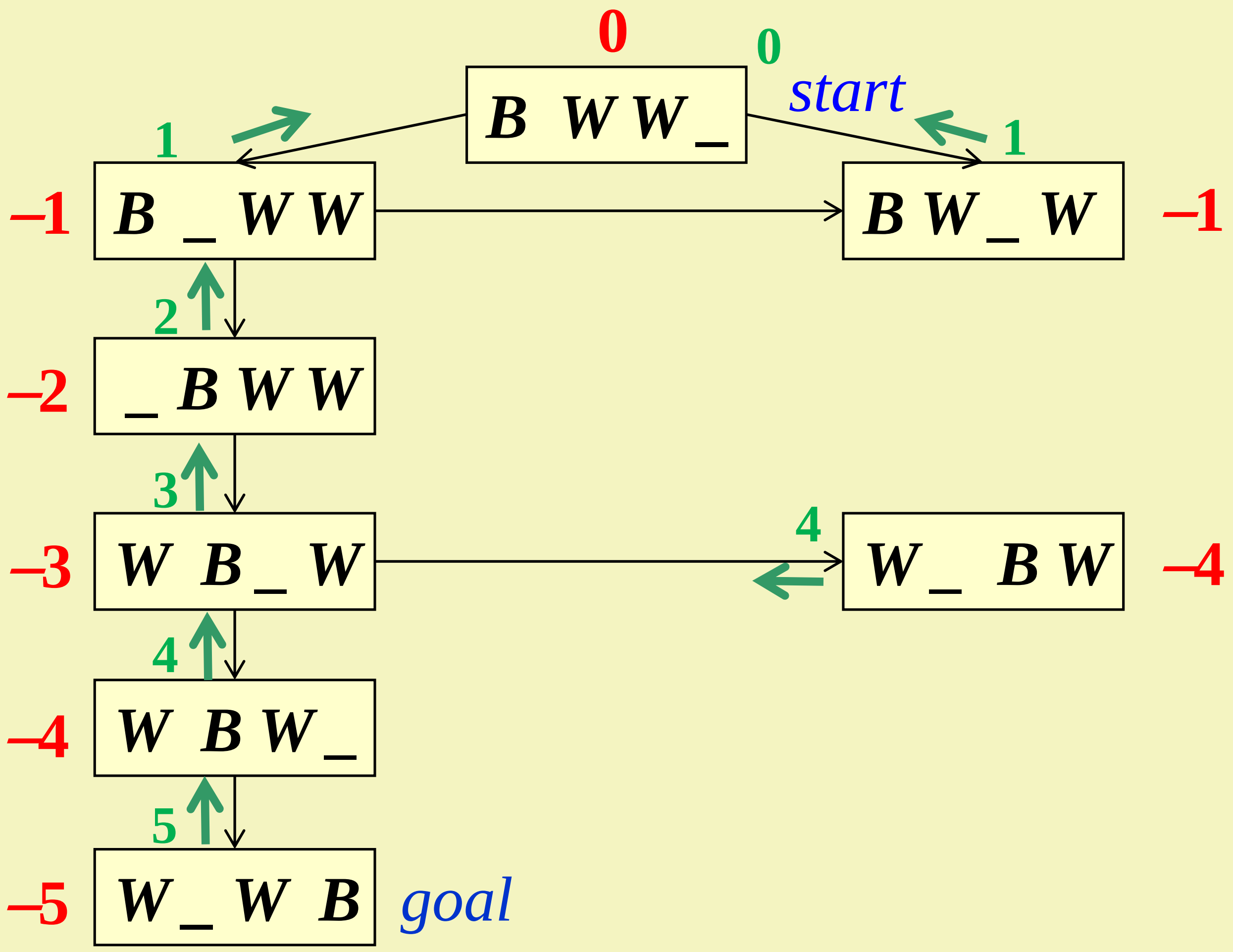

1.
2 8 3 / 1 6 4 / 7 _ 5
5
2 8 3 / 1 6 4 / _ 7 5
7
2.
2 8 3 / 1 _ 4 / 7 6 5
5
2 8 3 / 1 6 4 / 7 5 _
7
2 8 3 / _ 1 4 / 7 6 5
7
3.
2 _ 3 / 1 8 4 / 7 6 5
5
2 8 3 / 1 4 _ / 7 6 5
7
_ 8 3 / 2 1 4 / 7 6 5
2 8 3 / 7 1 4 / _ 6 5
4.
_ 2 3 / 1 8 4 / 7 6 5
5
2 3 _ / 1 8 4 / 7 6 5
7
5.
1 2 3 / _ 8 4 / 7 6 5
5
6.
1 2 3 / 8 _ 4 / 7 6 5
5
1 2 3 / 7 8 4 / _ 6 5
7

State graph of Black&White problem

$$\begin{tikzcd}
 & B\ W\ W\ \_ \arrow[dl, leftrightarrow] \arrow[drr, leftrightarrow] & \text{start} & & \\
B\ \_\ W\ W \arrow[rrr, leftrightarrow] \arrow[d, leftrightarrow] & & & B\ W\ \_\ W \arrow[d, leftrightarrow] & \\
\_\ B\ W\ W \arrow[d, leftrightarrow] & & & \_\ W\ B\ W \arrow[d, leftrightarrow] & \\
W\ B\ \_\ W \arrow[rrr, leftrightarrow] \arrow[d, leftrightarrow] & & & W\ \_\ B\ W \arrow[d, leftrightarrow] & \\
W\ B\ W\ \_ \arrow[d, leftrightarrow] & & & W\ W\ B\ \_ \arrow[d, leftrightarrow] & \text{goal} \\
W\ \_\ W\ B \arrow[rrr, leftrightarrow] \arrow[dr, leftrightarrow] & & & W\ W\ \_\ B \arrow[dll, leftrightarrow] & \text{goal} \\
\text{goal} & \_\ W\ W\ B & \text{goal} & &
\end{tikzcd}$$

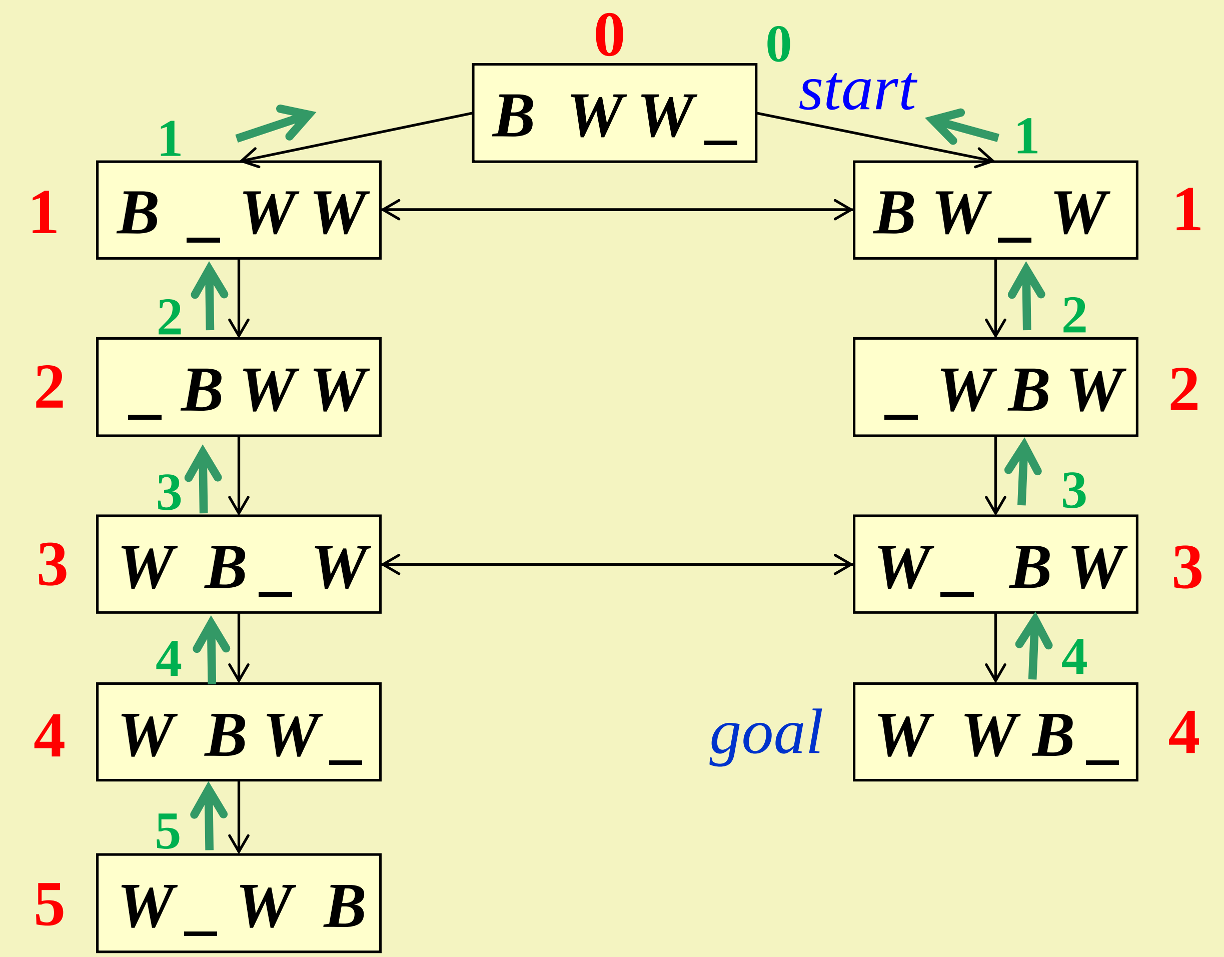
Depth-first graph-search

$f = -g$



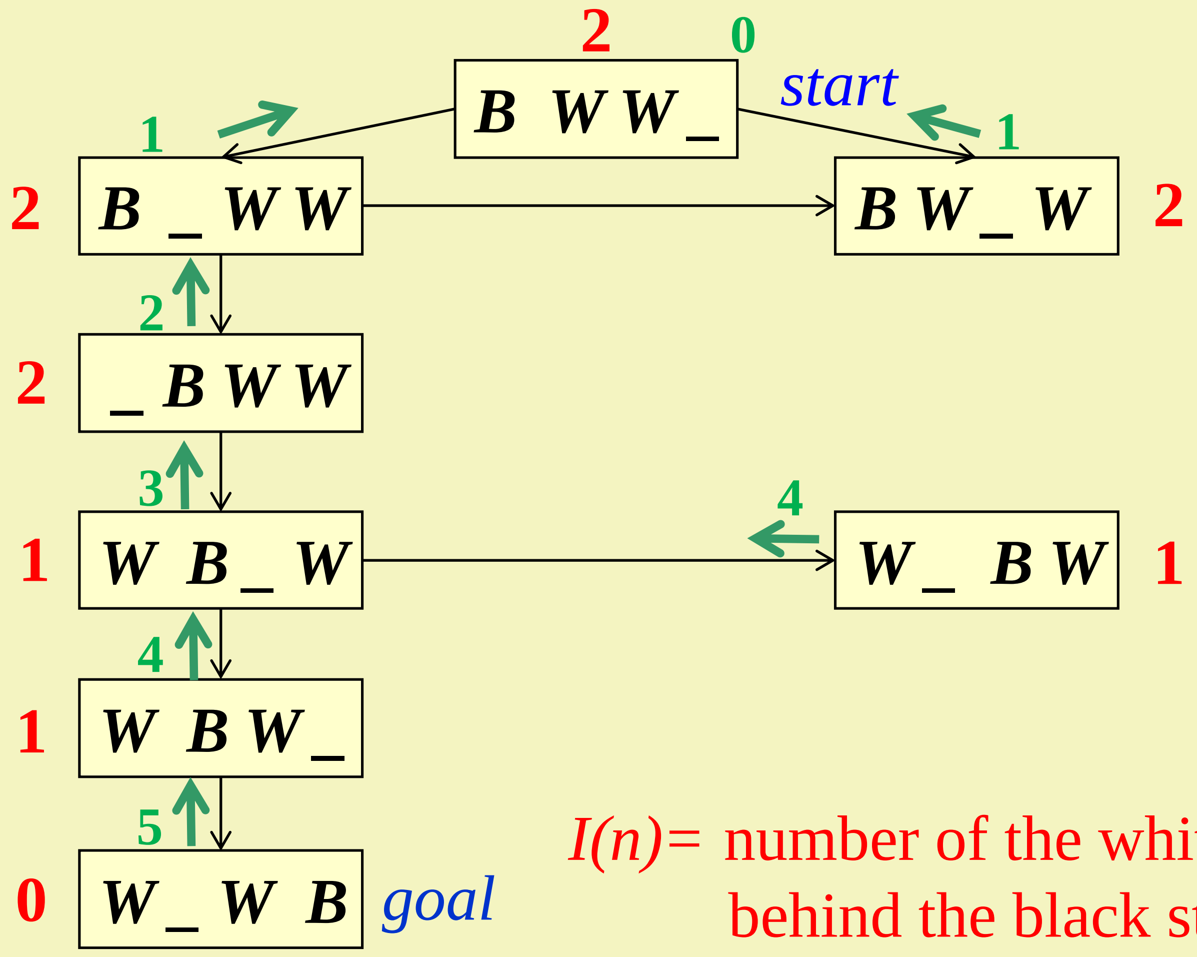
Breadth-first graph-search

f = *g*



Look-forward graph-search

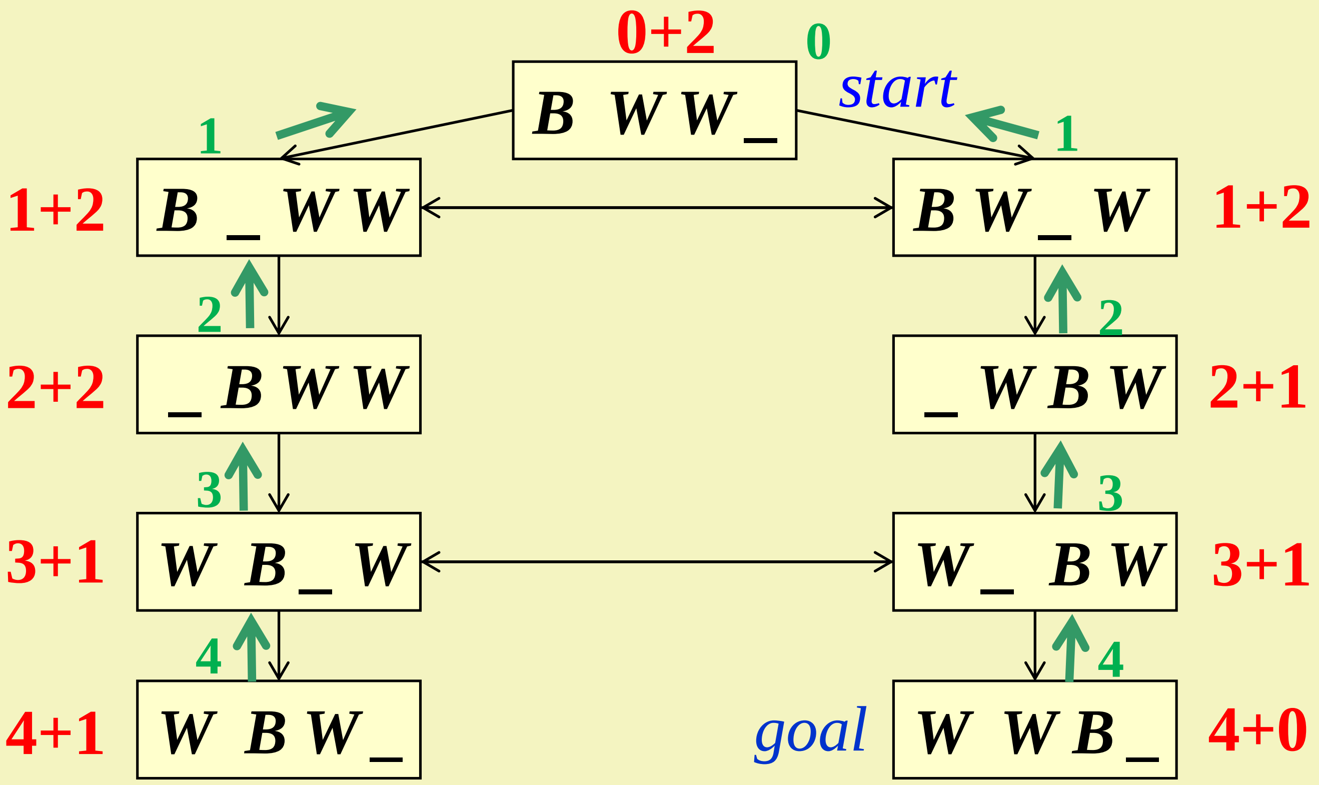
f = *I*



I(*n*)= number of the white stones behind the black stone

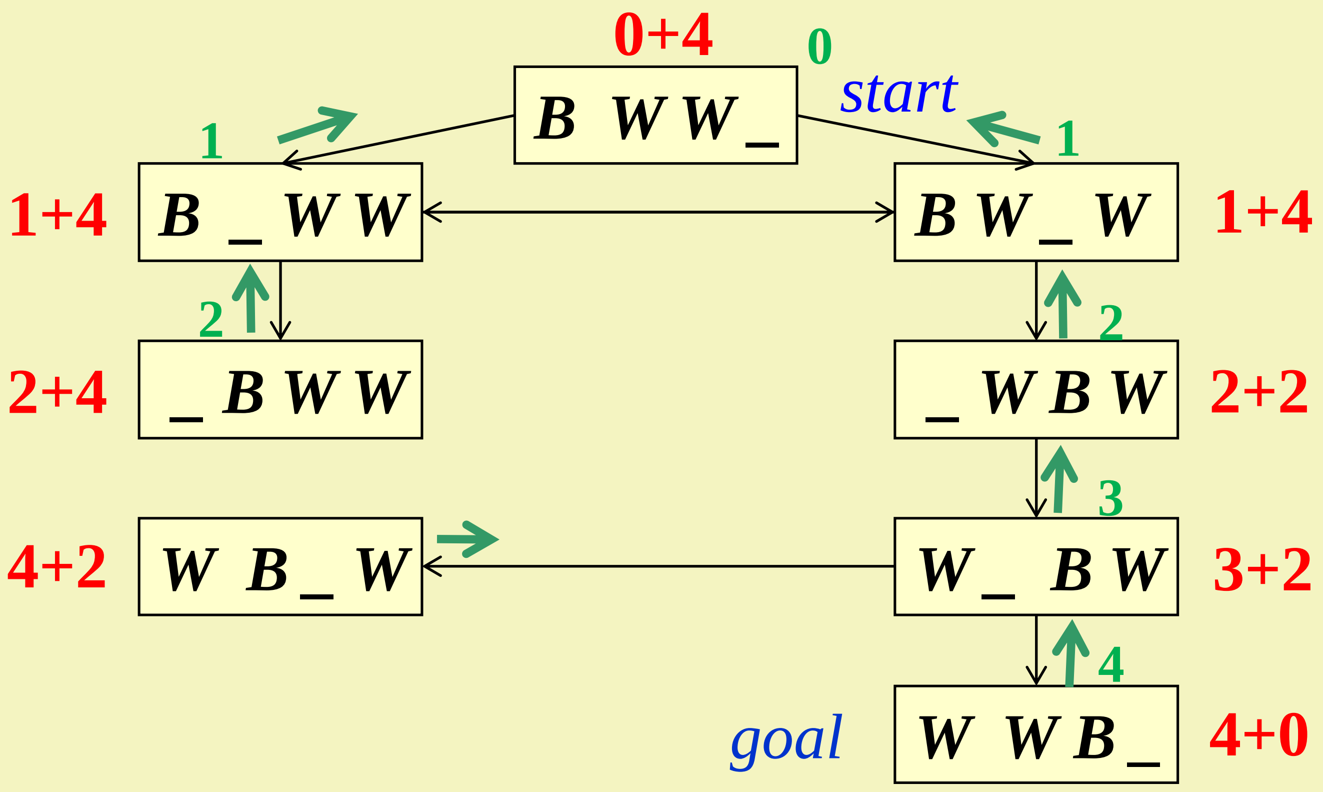
Algorithm A

$f = g + I$



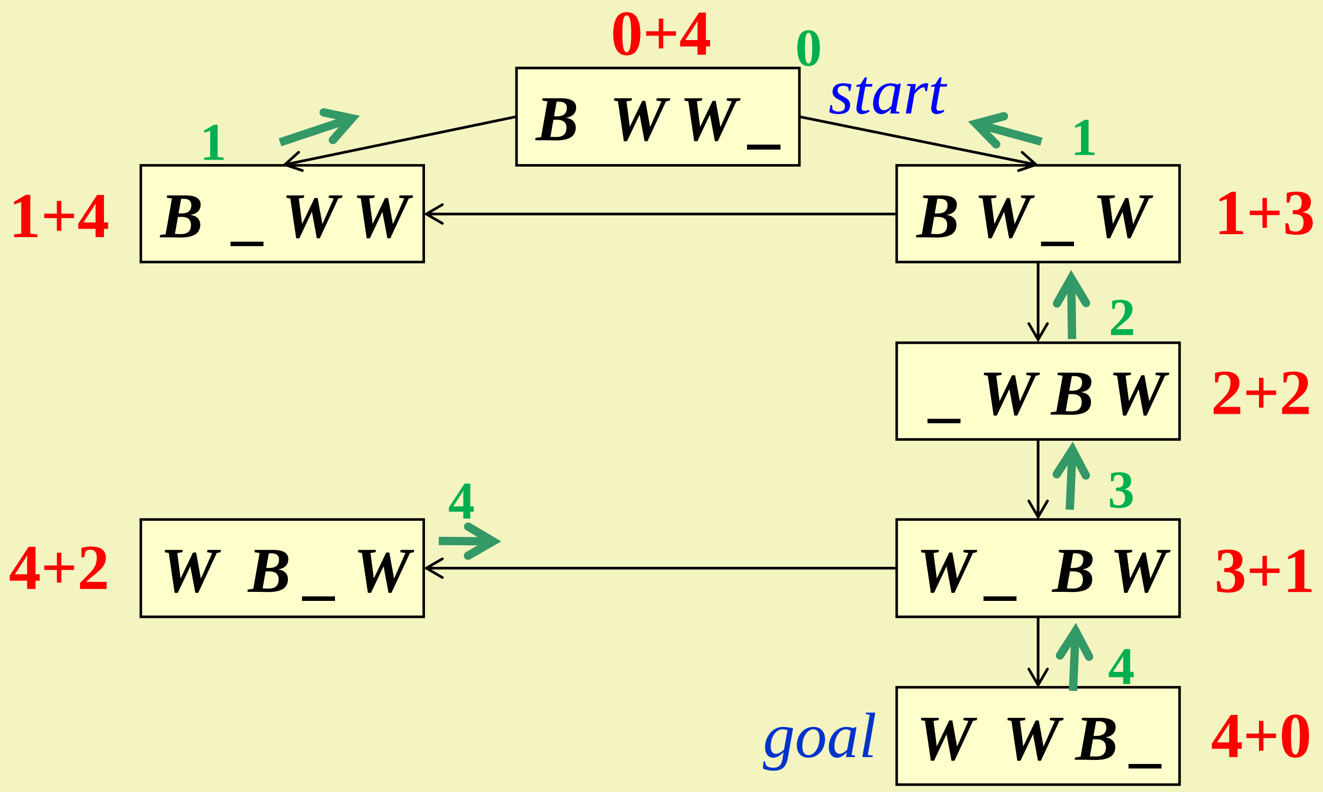
Algorithm A

*f = g + 2*I*



Algorithm A

f = *g* + 2**I* – (1 if there is either *BW_* or *_BW*)



Analyzing

algorithm A^c

f	***Graph-search***	***solution***	***G***	***Γ***
-g	*depth-first*	5	8	5
g	*breadth-first*	4	10	8
I	*look-forward*	5	8	5
g+I	*algorithm A*	4	9	7
*g+2*I*	*algorithm A*	4	8	5
*g+2*I–1(if…)*	*algorithm A*	4	7	4

algorithm A^c

algorithm A^c

3.3. Efficiency of *algorithm A**

Computational cost

Memory complexity

Number of closed nodes can estimate the size of the global workspace at termination

Time complexity

Number of expansions with respect to the number of the closed nodes shows the number of iterations

Admissible problems have got a solution and have got an admissible nonnegative heuristic function. (only these conditions guarantee that *algorithm A** finds optimal solution)

3.3.1. Analysis of memory requirement

- $CLOSED_S$ ~ the set of the closed nodes which are expanded by the graph-search algorithm S until its termination

- Fix a problem. Let X and Y be two graph-search algorithms.
  X is not worse than Y if $CLOSED_X \subseteq CLOSED_Y$
  X is better than Y if $CLOSED_X \subset CLOSED_Y$

- Using these definitions we can compare

1. two *algorithms* A^* (with different admissible heuristics) over the same problem.
2. the *algorithm* A^* with another *graph-search algorithm* over a given problem with the same heuristics.

Comparing two algorithms A^* using different heuristics

- Let A_1 (with heuristics h_1) and A_2 (with heuristics h_2) be *algorithms* A^*.
  A_2 is more informed than A_1 if h_2 gives a better estimation on the remaining optimal cost than h_1 for all non goal nodes i.e. $\forall n \in N \setminus T$: $h_1(n) < h_2(n)$.

  $\boxed{h_1(n) < h_2(n) \leqq h^*(n)}$

- It can be proved that a more informed algorithm (A_2) is not worse than a less informed one (A_1),
  i.e. $CLOSED_{A_2} \subseteq CLOSED_{A_1}$
  - In practice, however, many times $CLOSED_{A_2} \subset CLOSED_{A_1}$ even if $h_1(n) \leqq h_2(n)$ for $n \in N$

Closed nodes of the 15-puzzle

f =	*g+0*	*g+W*	*g+P*
6 steps solution	*167*	*9*	*8*
13 steps solution	*32389*	*119*	*13*
21 steps solution	*too many*	*3343*	*145*
30 steps solution	*too many*	*too many*	*1137*
34 steps solution	*too many*	*too many*	*3971*

Comparing algorithm A^* with other graph-search algorithms

- Our aim is to show that algorithm A^* does not require much more memory than the other graph-search algorithm over admissible problems.

- A graph-search algorithm is admissible if it can find an optimal solution over each admissible problem.

- Examples:
  - *Uniform-cost graph-search:* $f(n)=g(n)+0$
  - *Algorithm* A^* : $f(n)=g(n)+h(n)$
  - *Algorithm* A^{**} : $f(n)=\mathbf{max}_{m\in start\to n}(g(m)+h(m))$
    tie-breaking rule: prefers the goal node

Provable results

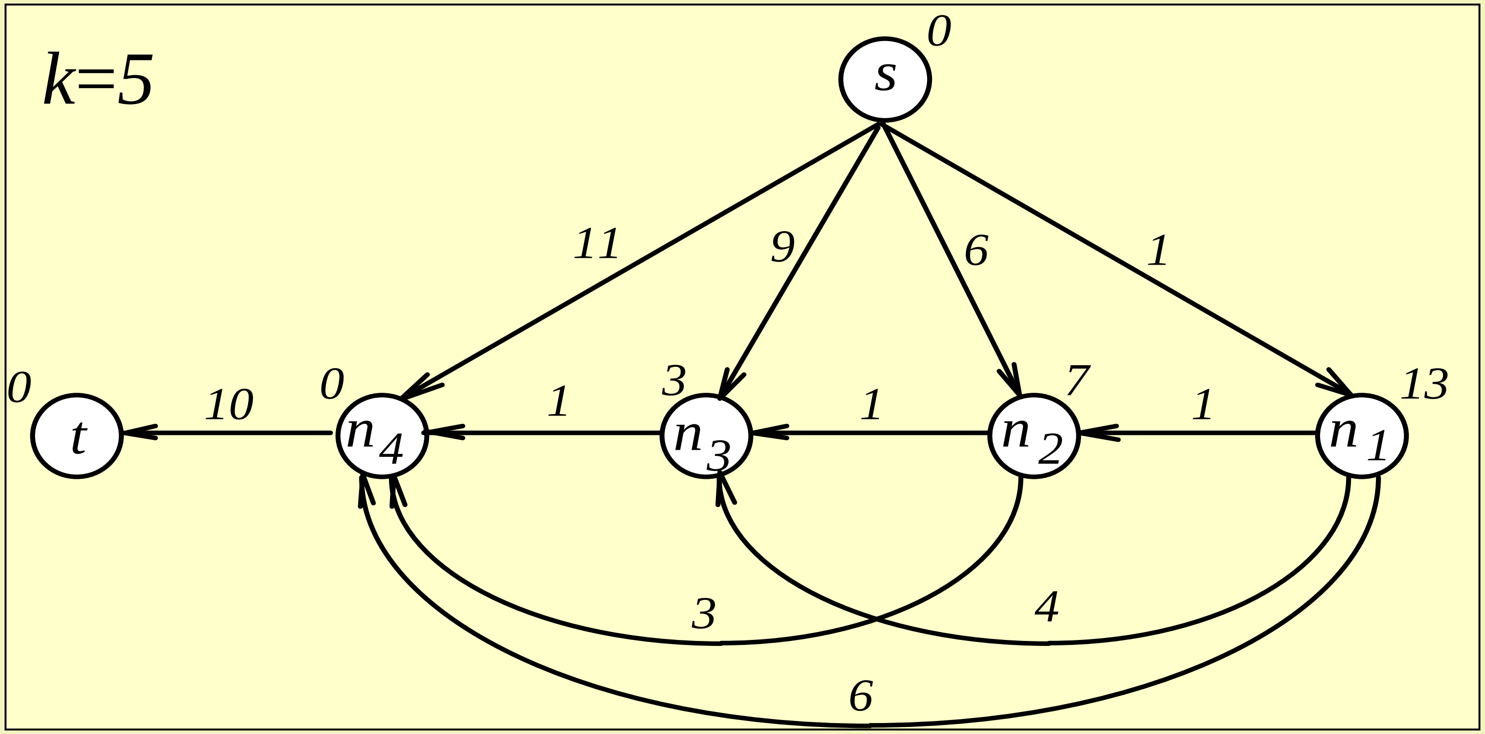
- *Algorithm A** might be worse than another admissible graph-search algorithm over a given problem. But
  - Each admissible graph-search algorithm might be worse than another admissible graph-search algorithm over a given problem.
  - *Algorithm A** is never worse than any other admissible graph-search algorithms over the problems with monotone heuristics.
  - No better admissible graph-search algorithm than *algorithm A** over the problems which contain at least one optimal solution path where $h(n)<h^*(n)$ for all nodes n except the goal node.

3.3.2. Analysis of running time

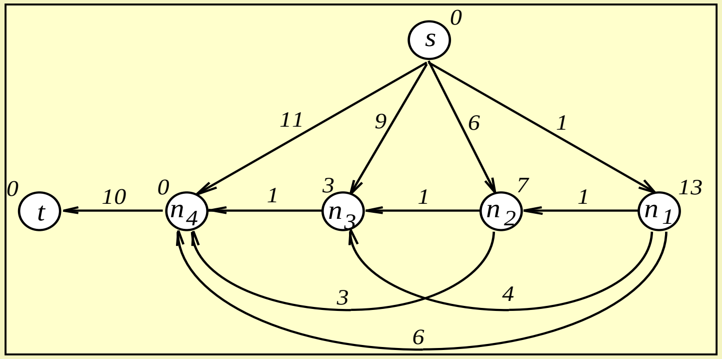
- Denote the number of the closed nodes of an algorithm A^* at termination as k.

- Lower limit on the number of the expansions (iterations): k
  - If algorithm A^* uses a monotone heuristics, (it is an algorithm A^c) it expands a node at most once thus the number of the closed nodes is equal to the number of their expansions.
- Upper limit on the number of the expansions : 2^{k-1}
  - See Martelli’s problem

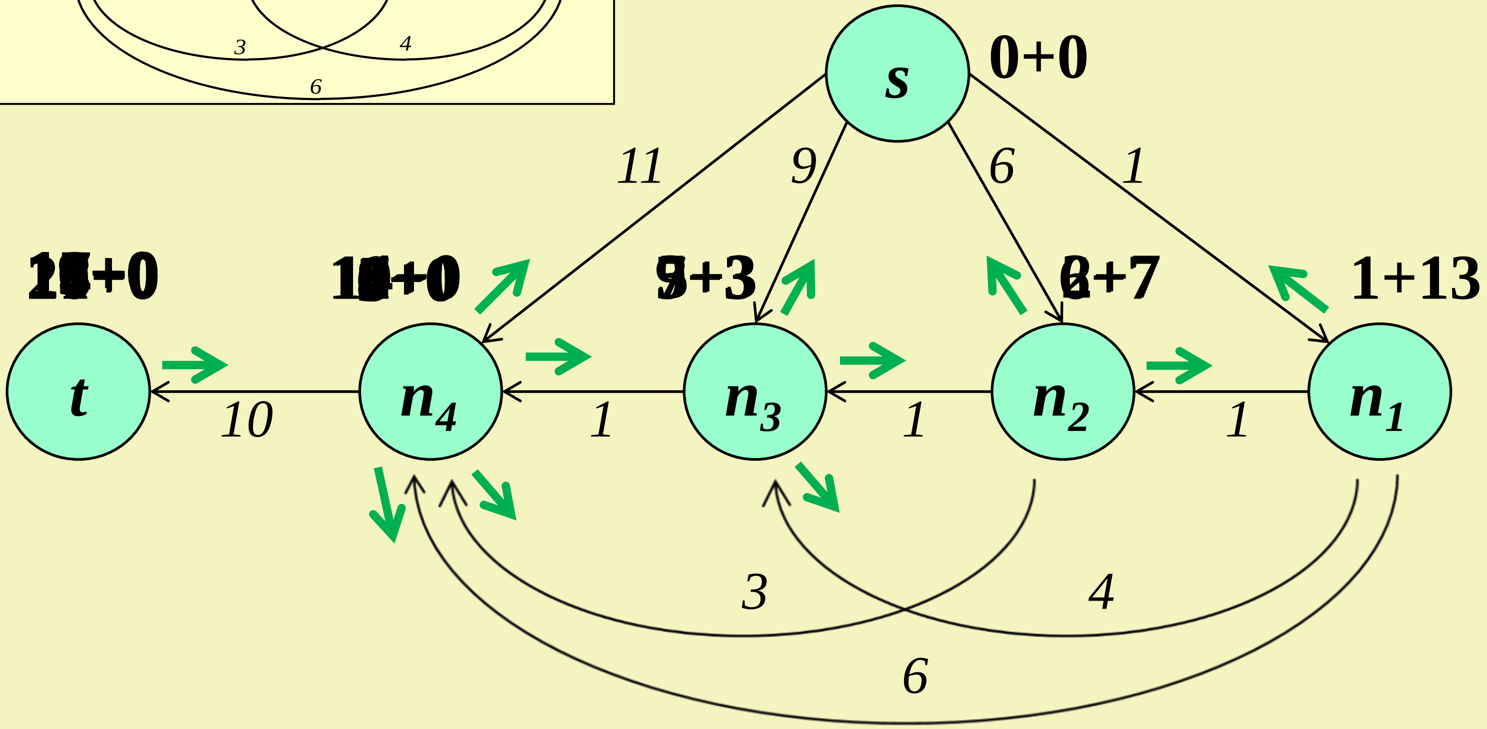
Martelli's example



Execution

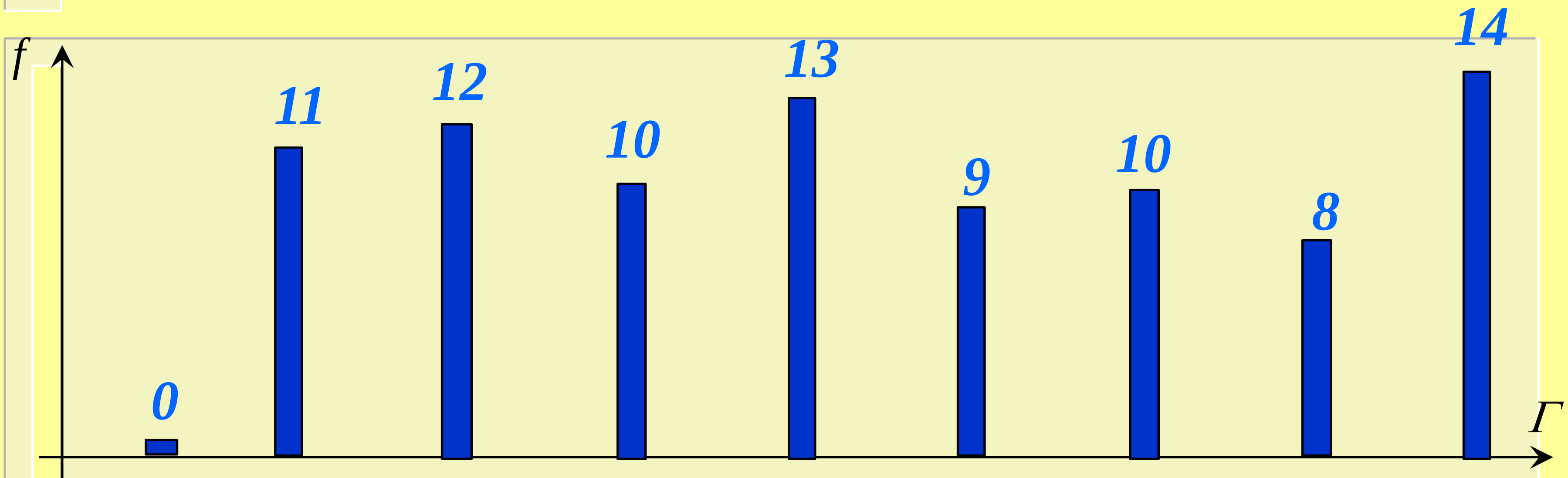


s
0
11
9
6
1
t
0
10
n4
0
1
n3
3
1
n2
7
1
n1
13
3
4
6



s
0+0
11
9
6
1
t
n4
n3
n2
n1
1+13
10
1
1
1
3
4
6

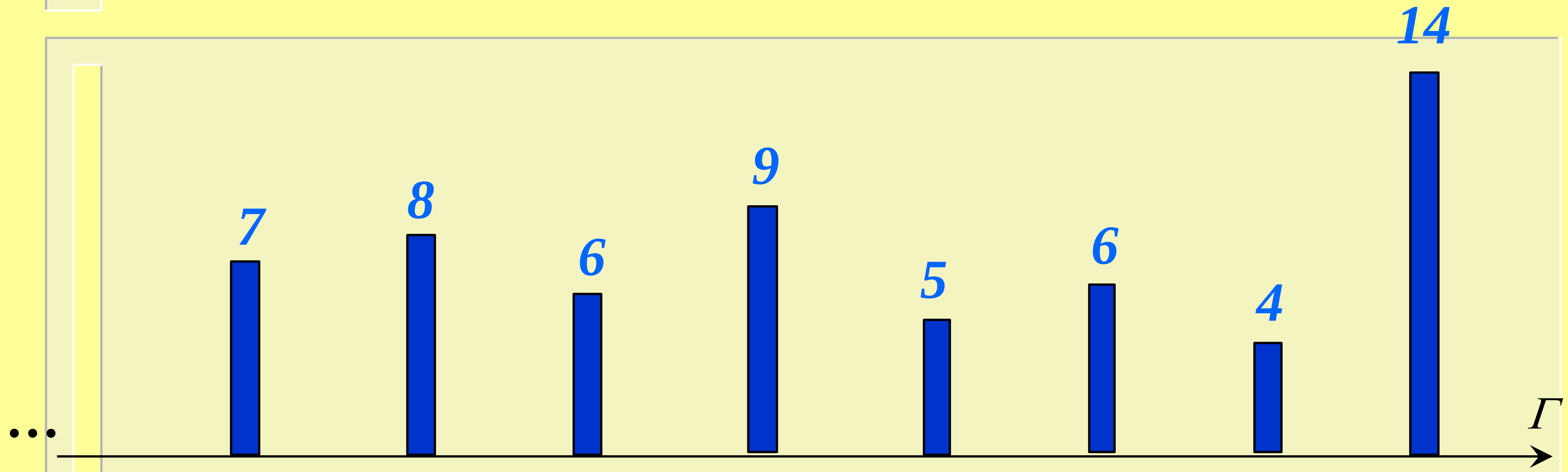
Execution diagram



OPEN	s	n_4	n_3	n_4	n_2	n_4	n_3	n_4	n_1
s	nil,0,0	-	-	-	-	-	-	-	-
n_1	-	s, 1,14	s, 1,14	s,1,14	s, 1,14	s, 1,14	s, 1,14	s, 1,14	s, 1,14
n_2	-	s, 6,13	s, 6,13	s, 6, 13	s, 6, 13	-	-	-	-
n_3	-	s, 9,12	s, 9,12	-	-	n_2,7,10	n_2,7,10	-	-
n_4	-	s, 11,11	-	n_3,10,10	-	n_2,9,9	-	n_2,8,8	-
t	-	-	n_4,21,21	n_4,21,21	n_4,20,20	n_4,20,20	n_4,19,19	n_4,19,19	n_4,18,18

π,g,f

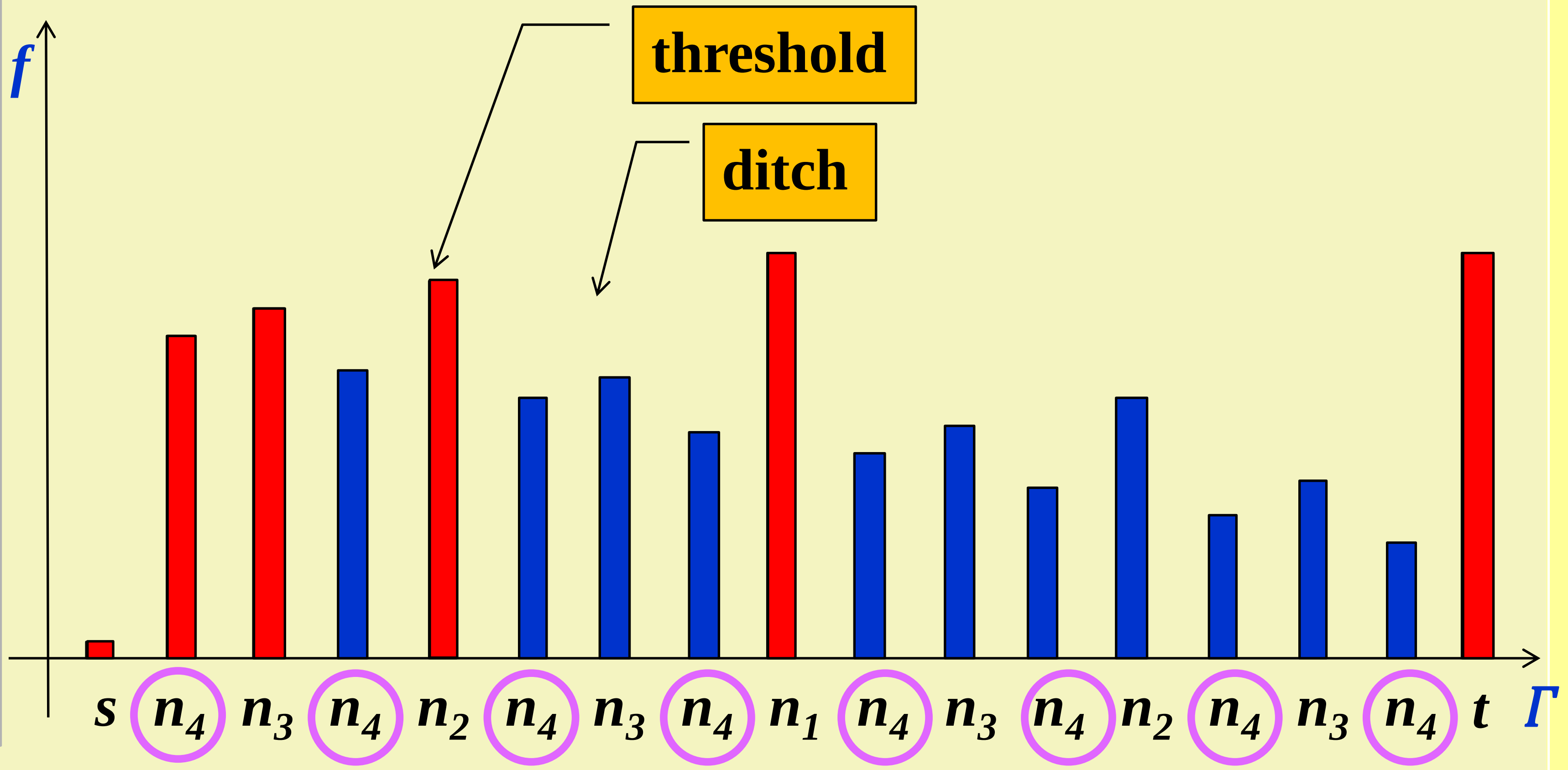
Execution diagram



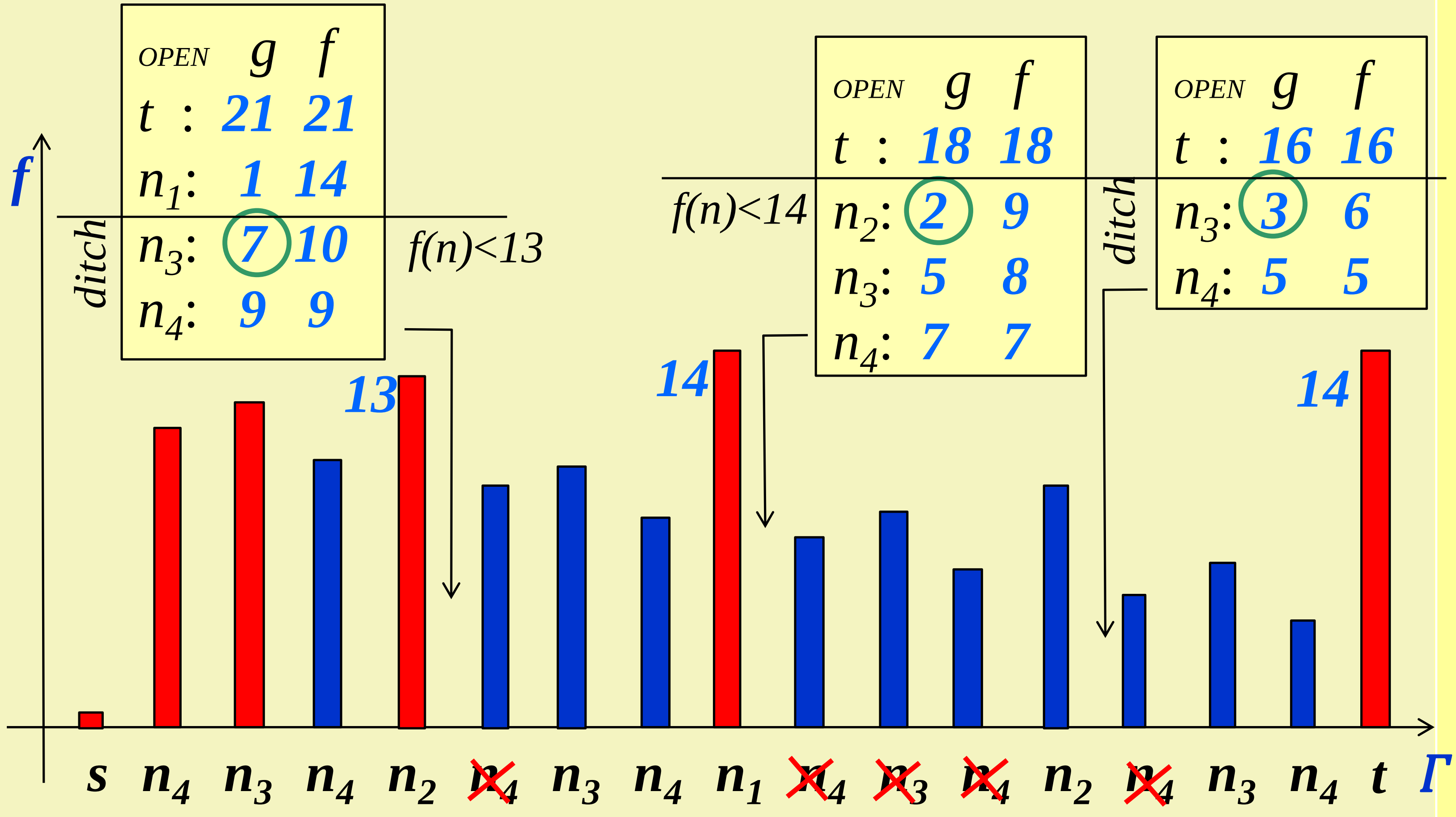
OPEN	n_4	n_3	n_4	n_2	n_4	n_3	n_4	t
s	-	-	-	-	-	-	-	-
n_1	-	-	-	-	-	-	-	-
n_2	n_1,2,9	n_1,2,9	n_1,2,9	n_1,2,9	-	-	-	-
n_3	n_1,5,8	n_1,5,8	-	-	n_1,3,6	n_1,3,6	-	-
n_4	n_1,7,7	-	n_3,6,6	-	n_2,5,5	-	n_3,4,4	-
t	n_4,18,18	n_4,17,17	n_4,17,17	n_4,16,16	n_4,16,16	n_4,15,15	n_4,15,15	n_4,14,14

π,g,f

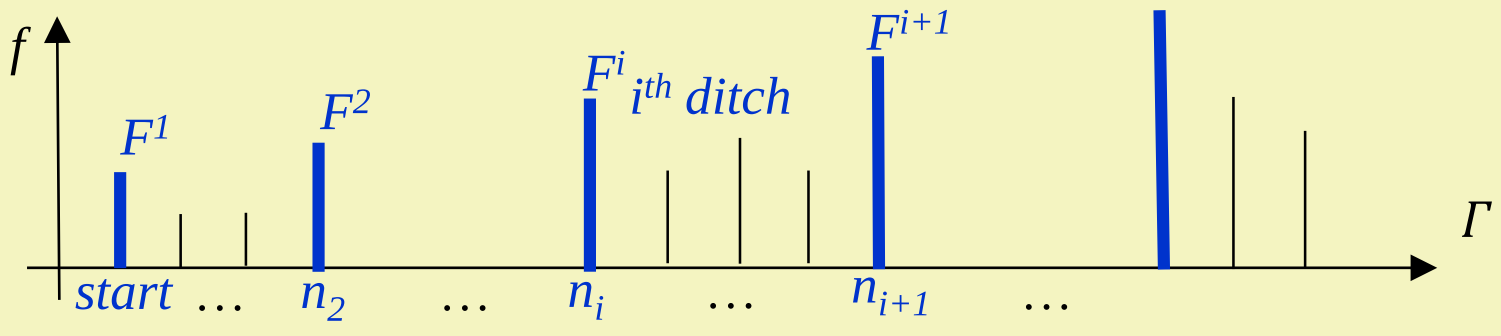
*Number of executions of algorithm A**



Reduce the number of expansions inside a ditch



Discussion



- A secondary (inner) evaluation function may be introduced in the ditches to select the best node among open nodes having less value then the current threshold value.
- It can be proved that this inner function does not influence the thresholds (neither the threshold nodes nor their values nor their order). It can change only the order and the number of the expansions of the nodes belonging to a ditch.

Algorithm B

- Martelli suggested the function *g* as an inner evaluation function.
- Introduce the variable *F* to store the current threshold value. Change the step 1 and step 4 of the general graph-search:

```
– step 1. + $F := f(start)$
– step 4. if $min_f(OPEN)<F$
              then  $n := min_g(m \in OPEN \mid f(m)<F)$
              else  $n := min_f(OPEN); F := f(n)$
          endif
```

Running time of algorithm B

- *Algorithm B* works in the same way as *algorithm A** except that it expands a node in a ditch only once.
  - In the worst case
    - the first expansion of each closed node might be as a threshold. (Under decreasing evaluation function one node may be threshold only once at its first expansion.)
    - the i^{th} ditch may contain at most the previous $i-1$ threshold nodes (except for the start node)
  - It follows that the number of expansions may be at most $\frac{1}{2}\cdot k^2$.