BEST-FIRST SEARCH AND ALGORITHM A*

JEFFREY L. POPYACK

GRAPHSEARCH REVIEW

Key statement in **Graphsearch**:

Insert(s', OPEN):

- Insert at front: ***Depth-First Search***
- Insert at rear: ***Breadth-First Search***

GRAPHSEARCH WITH HEURISTICS

Key statement in **Graphsearch**:

Insert(s', OPEN): **use heuristics to guide**

- Insert at front: ***Depth-First Search***
  - ***heuristic = constant***
    (just put at front, since all have same value)
- Insert at rear: ***Breadth-First Search***

GRAPHSEARCH WITH HEURISTICS

Key statement in **Graphsearch**:

Insert(s', OPEN): **use heuristics to guide**

- Insert at front: ***Depth-First Search***
  - ***heuristic = constant***
    (just put at front, since all have same value)
- Insert at rear: ***Breadth-First Search***
  - ***heuristic = depth(node)***
    (goes at end, since OPEN is ordered from low to high)

GRAPHSEARCH WITH HEURISTICS

Key statement in **Graphsearch**:

Insert(s', OPEN): **use heuristics to guide**

- Insert at front: ***Depth-First Search***
  - ***heuristic = constant***
    (just put at front, since all have same value)
- Insert at rear: ***Breadth-First Search***
  - ***heuristic = depth(node)***
    (goes at end, since OPEN is ordered from low to high)
- Insert where it *probably belongs*: ***Best-First Search***

GRAPHSEARCH WITH HEURISTICS

Key statement in **Graphsearch**:

Insert(s', OPEN): **use heuristics to guide**

- Insert where it *probably belongs*: ***Best-First Search***
- In general, don't know for sure.
- Use a good heuristic: (low: good; high: bad)
- **OPEN** is ordered from low to high

ALGORITHM A

IDEA: ("Algorithm A")

Let **f(n)** be an *estimate* of the TOTAL COST (path-length, distance) from **start** to a **goal** *if you go through node* ***n***.

ALGORITHM A

<u>IDEA</u>: ("Algorithm A")

Let **f(n)** be an *estimate* of the TOTAL COST (path-length, distance) from **start** to a **goal** *if you go through node **n***.

<u>EXAMPLE</u>:

start = Drexel

goal = Disney World

current location = Charlottesville, VA

Total Distance if going through C'ville:

dist(Drexel,C'ville) + dist(C'ville, DisneyWorld)

= *250 + 800* = *1050*

ALGORITHM A

IDEA: ("Algorithm A")

Let **f(n)** be an *estimate* of the TOTAL COST (path-length, distance) from **start** to a **goal** *if you go through node* ***n***.

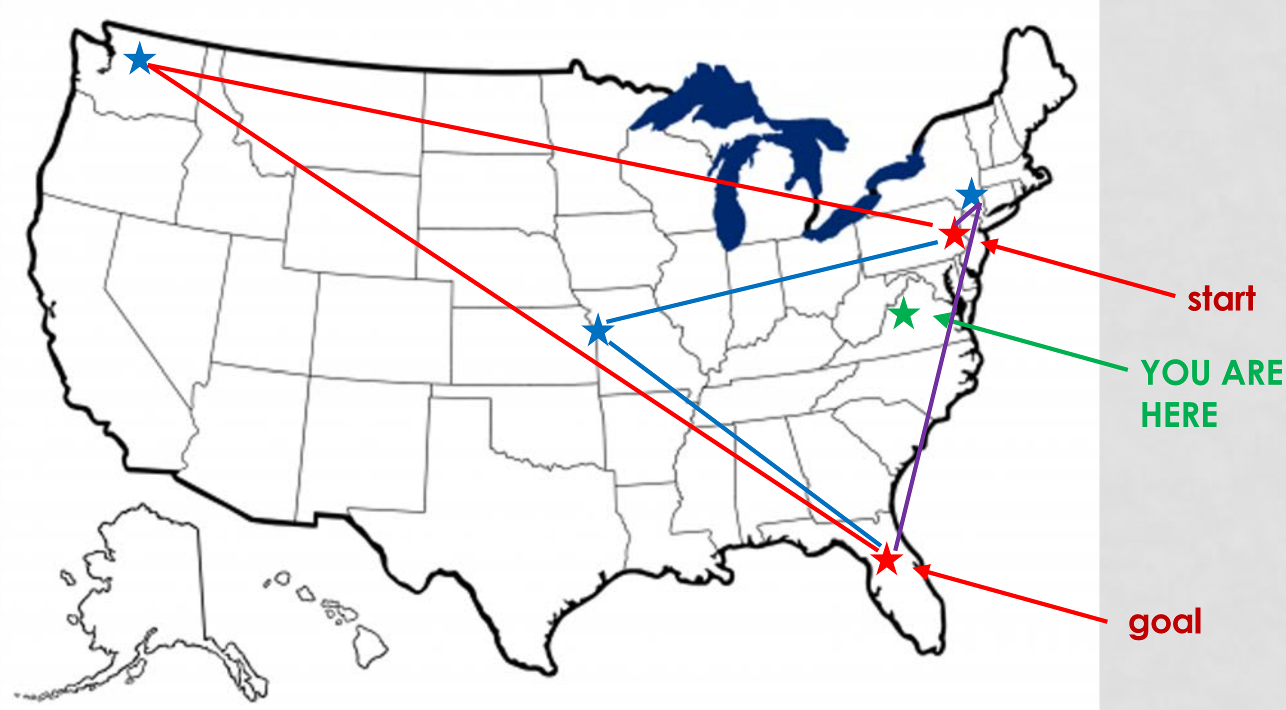
f(n) = Total Distance
if going through n:

f(C'ville) =
250 + 800 =1050

f(Kansas City) =
1125 + 1250 = 2375

f(New York City) =
95 + 1100 = 1195

f(Seattle) =
2800 + 3100 = 5900



f(Seattle) > f(Kansas City) > f(New York City) > f(C'ville)

ALGORITHM A

Algorithm A:

Let **f(n)** be an *estimate* of the TOTAL COST (path-length, distance) from **start** to a **goal** *if you go through node **n***.

f(n) = depth(n) + h(n)

where **h(n)** is an *estimate* of distance from **n** to a goal

and **depth(n)** is the *actual (known) distance traveled so far, from **start** to **n**.*

ALGORITHM A*

Algorithm A*: **("A-Star")**

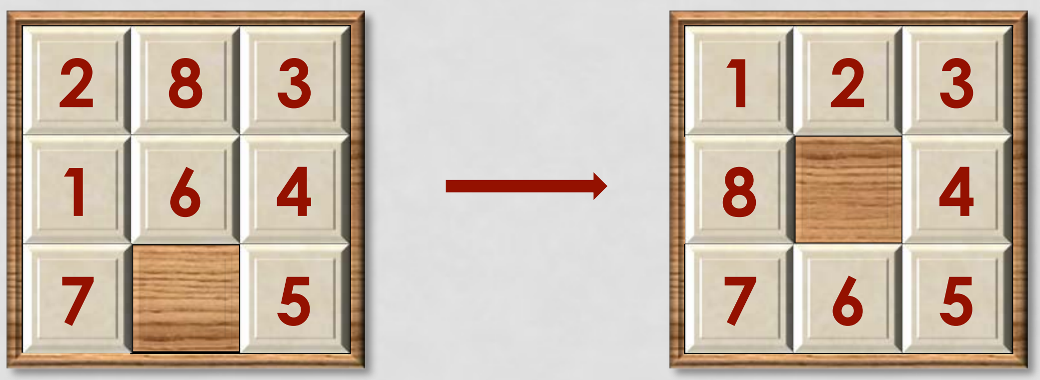
Let **f(n)** be an *estimate* of the TOTAL COST
f(n) = depth(n) + h(n), as in **Algorithm A**,
*but with a guarantee that **h(n)** does not ever over-estimate the distance from **n** to a **goal**.*

Use of Algorithm A* is guaranteed to find a shortest-path solution!

EXAMPLE: 8-PUZZLE

Recall: for the 8-Puzzle, the rules are:

- <u>actions</u>: move the *empty spot* UP, DOWN, LEFT or RIGHT
- <u>preconditions</u>: the destination spot is on the board.



For Best-First Search, we will use **Algorithm A**, with

f(n) = depth(n) + h(n),

and the heuristic

h(n) = number of tiles out of place (not including empty cell).

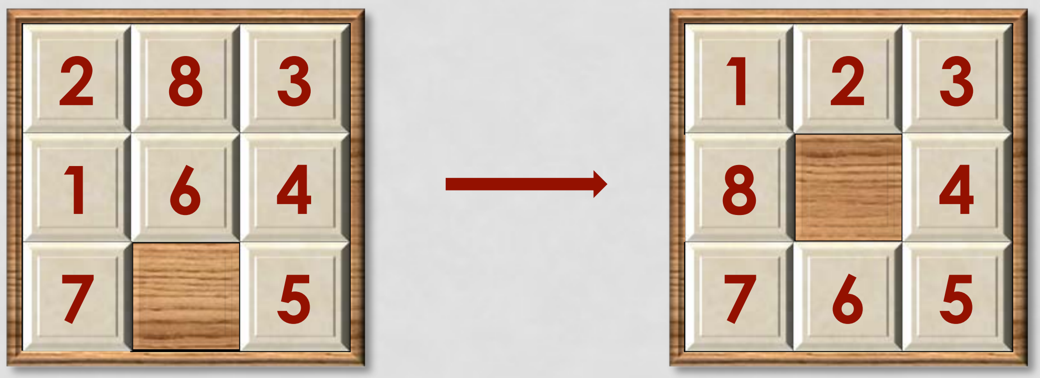
EXAMPLE: 8-PUZZLE

For Best-First Search, we will use **Algorithm A**, with

f(n) = depth(n) + h(n),

and the heuristic

h(n) = number of tiles out of place (not including empty cell).



h(start) = 4 (Since 1,2,6,8 are out of position, others OK).

It will take at least 4 moves to get all tiles in position (goal!) because on a single move, at most 1 tile can move into position. *In short, **h()** never overestimates the distance to the goal.*

This is an ***admissible heuristic*** for **Algorithm A***

OPEN: { A }
CLOSED: { }

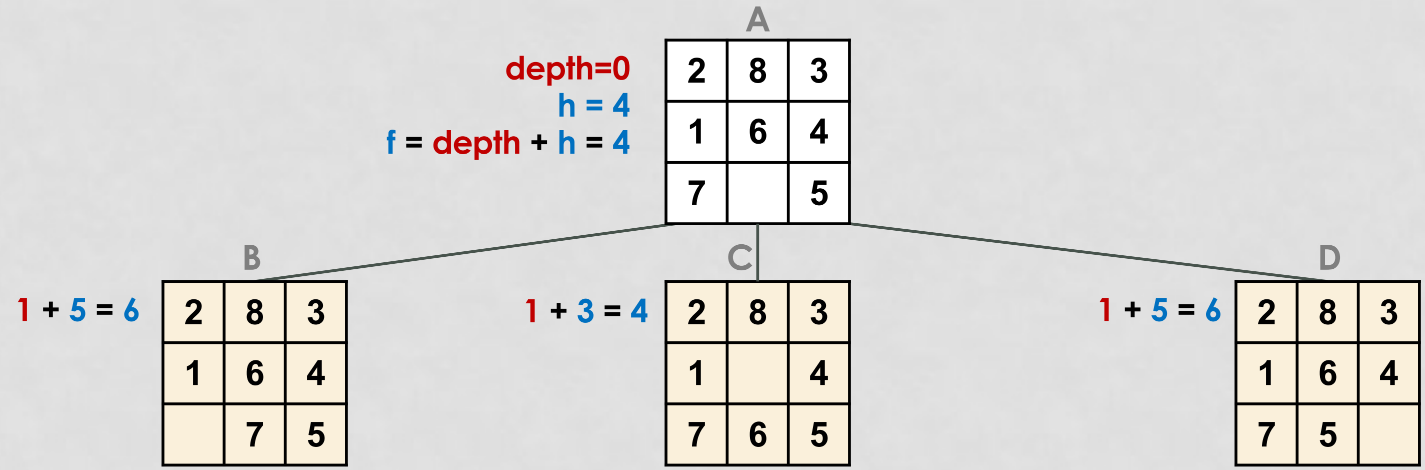
depth=0
h = 4
f = depth + h = 4

A

2	8	3
1	6	4
7		5

OPEN: { A }
CLOSED: { }

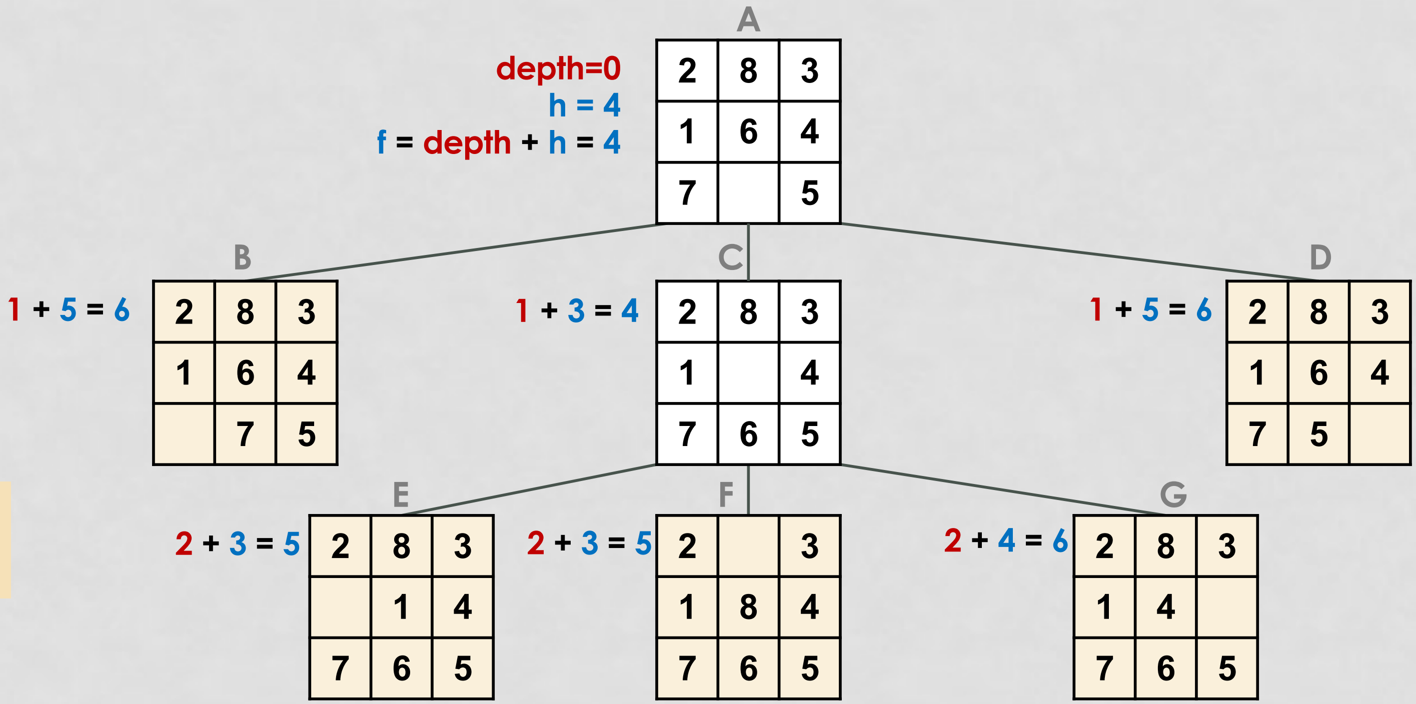
OPEN: { C B D }
CLOSED: { A }



OPEN: { A }
CLOSED: { }

OPEN: { C B D }
CLOSED: { A }

OPEN: { E F G B D }
CLOSED: { A C }

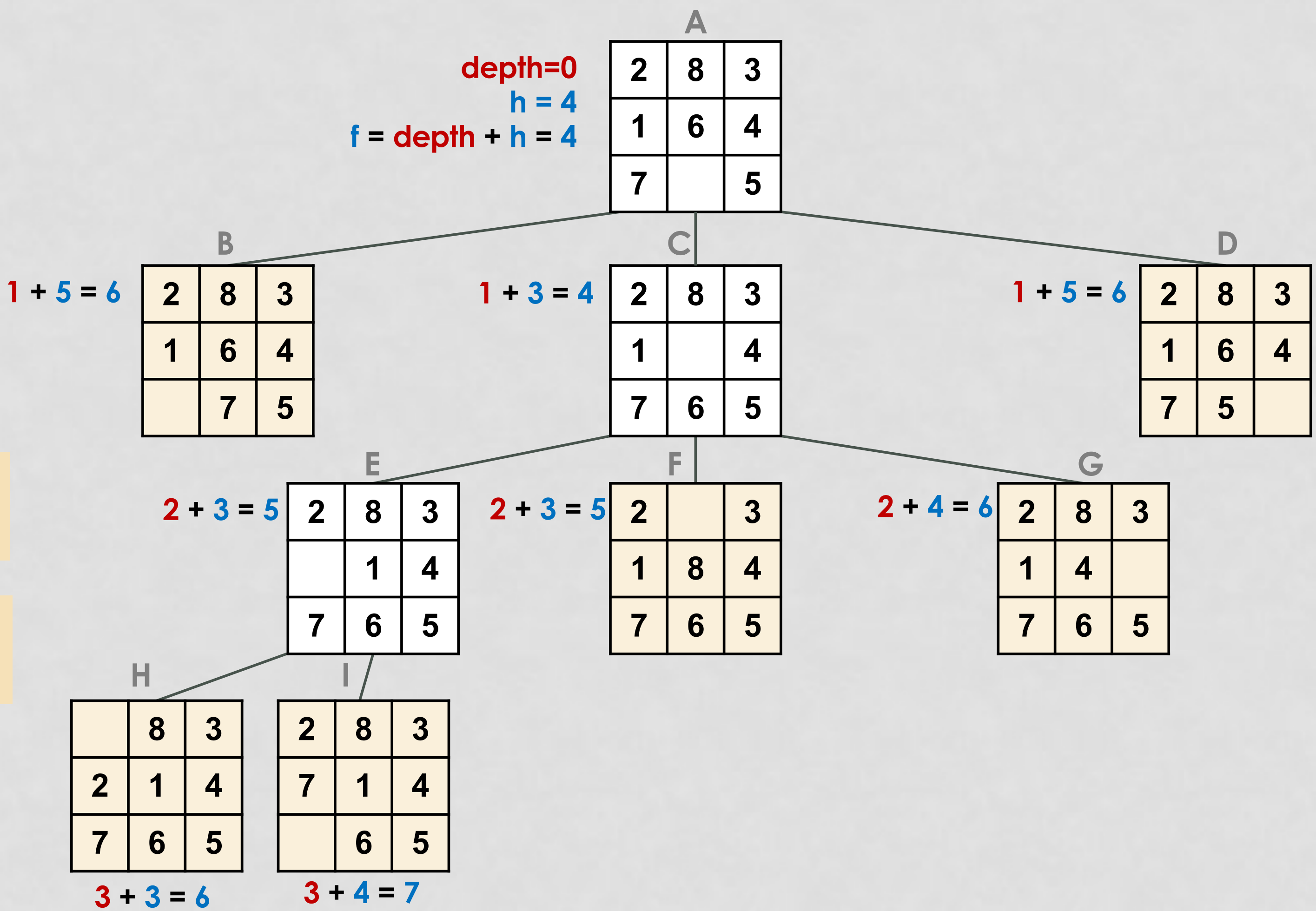


OPEN: { A }
CLOSED: { }

OPEN: { C B D }
CLOSED: { A }

OPEN: { E F G B D }
CLOSED: { A C }

OPEN: { F H G B D I }
CLOSED: { A C E }



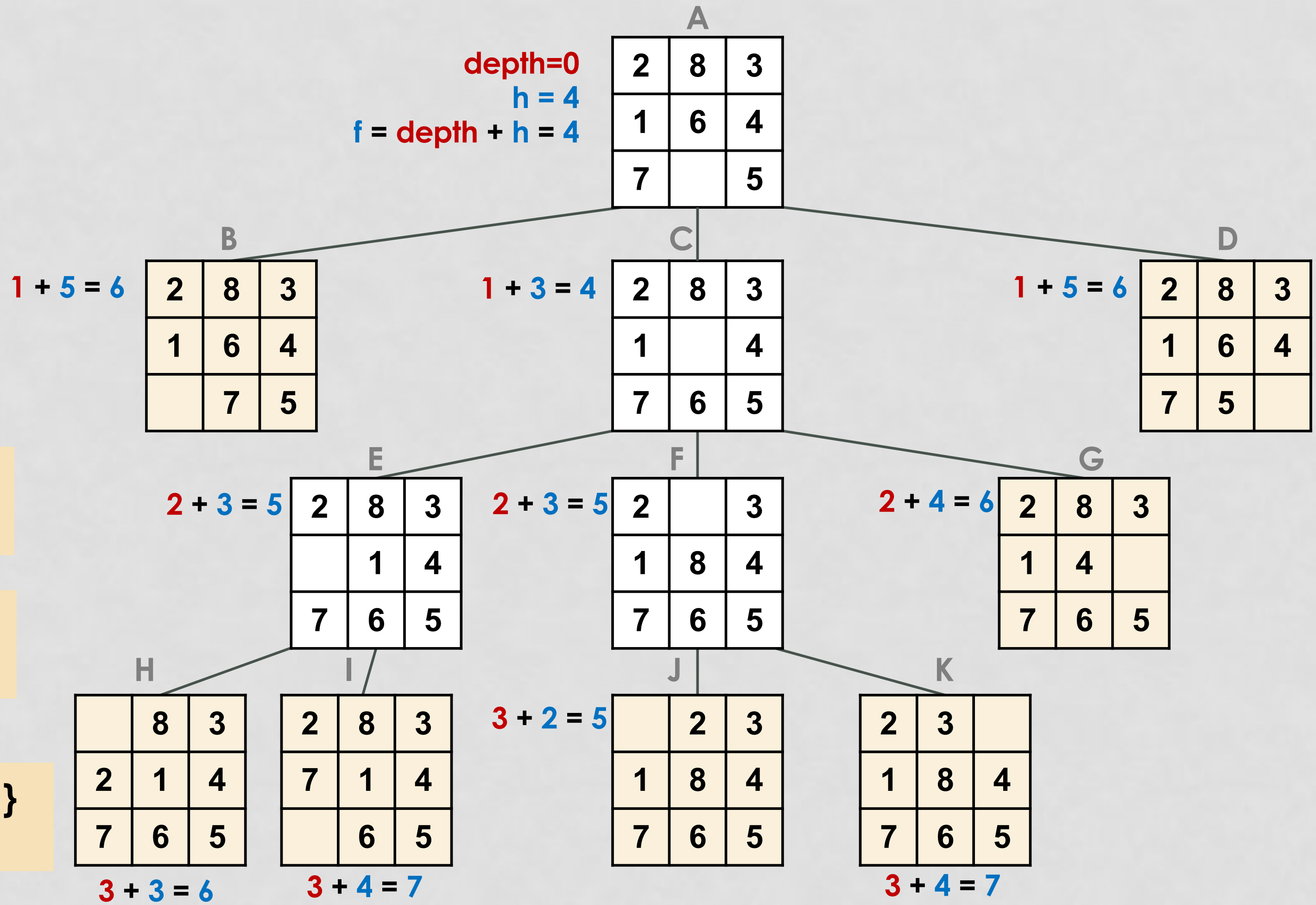
OPEN: { A }
CLOSED: { }

OPEN: { C B D }
CLOSED: { A }

OPEN: { E F G B D }
CLOSED: { A C }

OPEN: { F H G B D I }
CLOSED: { A C E }

OPEN: { J H G B D K I }
CLOSED: { A C E F }



OPEN: { A }
CLOSED: { }

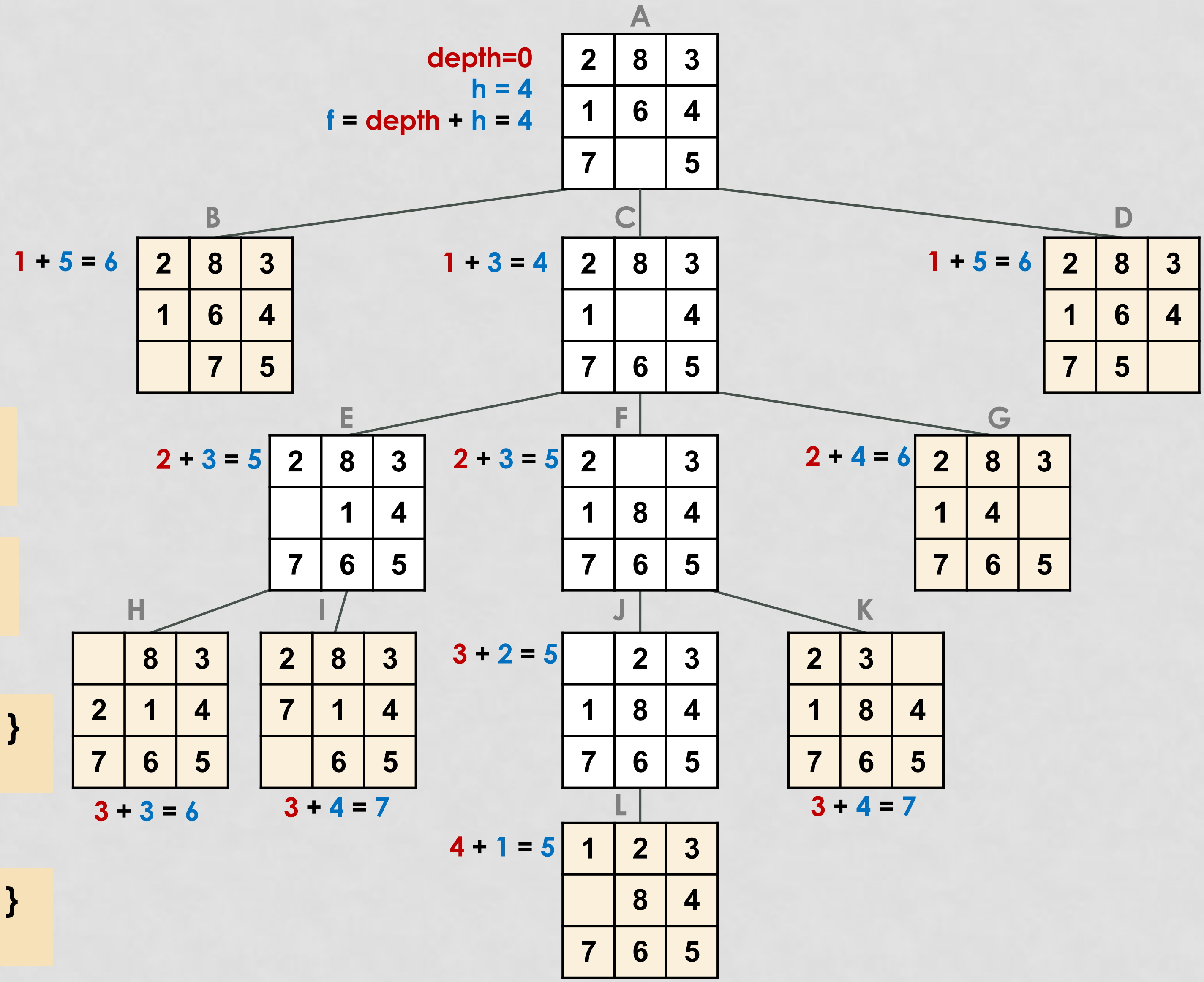
OPEN: { C B D }
CLOSED: { A }

OPEN: { E F G B D }
CLOSED: { A C }

OPEN: { F H G B D I }
CLOSED: { A C E }

OPEN: { J H G B D K I }
CLOSED: { A C E F }

OPEN: { L H G B D K I }
CLOSED: { A C E F J }



OPEN: { A }
CLOSED: { }

OPEN: { C B D }
CLOSED: { A }

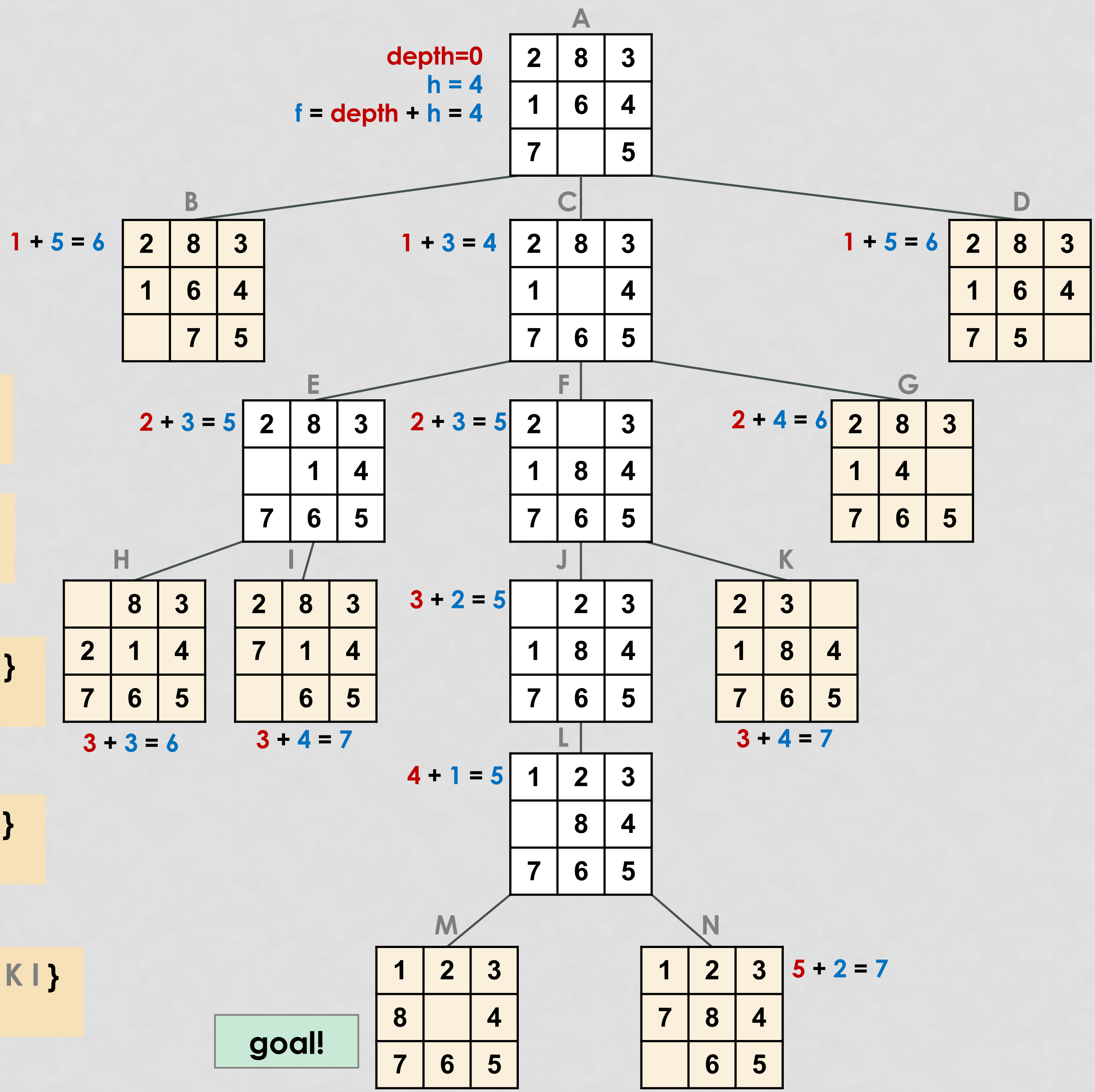
OPEN: { E F G B D }
CLOSED: { A C }

OPEN: { F H G B D I }
CLOSED: { A C E }

OPEN: { J H G B D K I }
CLOSED: { A C E F }

OPEN: { L H G B D K I }
CLOSED: { A C E F J }

OPEN: { M H G B D N K I }
CLOSED: { A C E F J L }



ALGORITHM A*

Best-First Search, *using Algorithm A**

- finds a shortest-path solution to a goal
- requires fewer node expansions than breadth-first search (an admissible heuristic guarantees you won't go deeper than a goal node, so you can't expand more nodes than breadth-first search.)
- requires fewer node expansions than depth-first search (unless you were lucky – a good heuristic will let you choose properly which nodes to examine next, rather than rely on luck.)
- should have improved performance with an improved heuristic.