CPSC 322
Introduction to Artificial Intelligence

October 27, 2004

Things...

Assignment 3 is due Friday morning

Send me pairings for term project no later than Sunday, Oct. 31

Midterm exam 2 is Monday

Bring me your first midterms before the second midterm if you want to be retested on problem 3

Jessica Hodgins talk, Thursday, October 28, 1-2pm, MacLeod 214

Back to heuristic search techniques

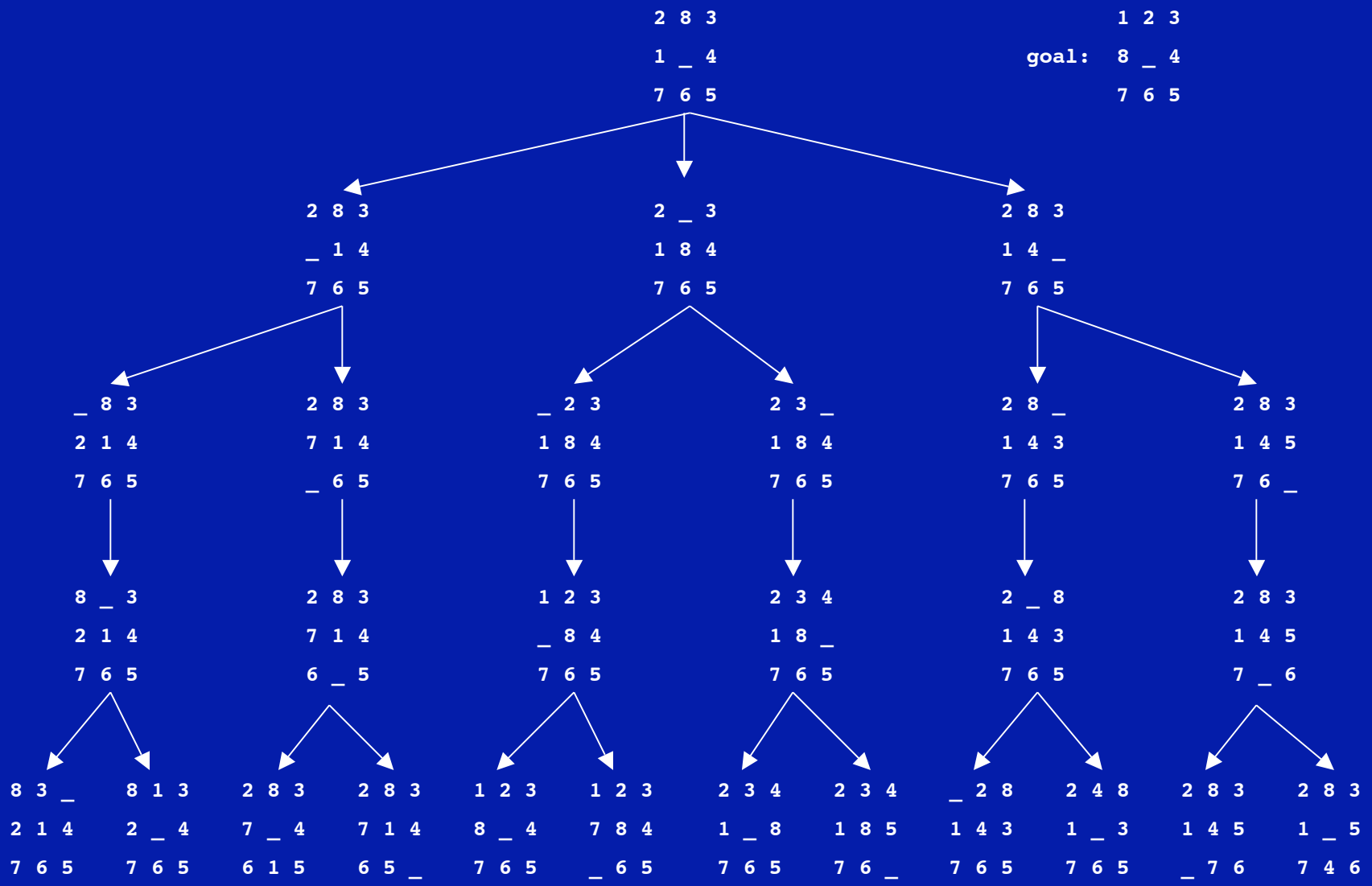
Heuristic depth-first search algorithm

Given a set of start nodes, a set of goal nodes,
and a graph (i.e., the nodes and arcs):

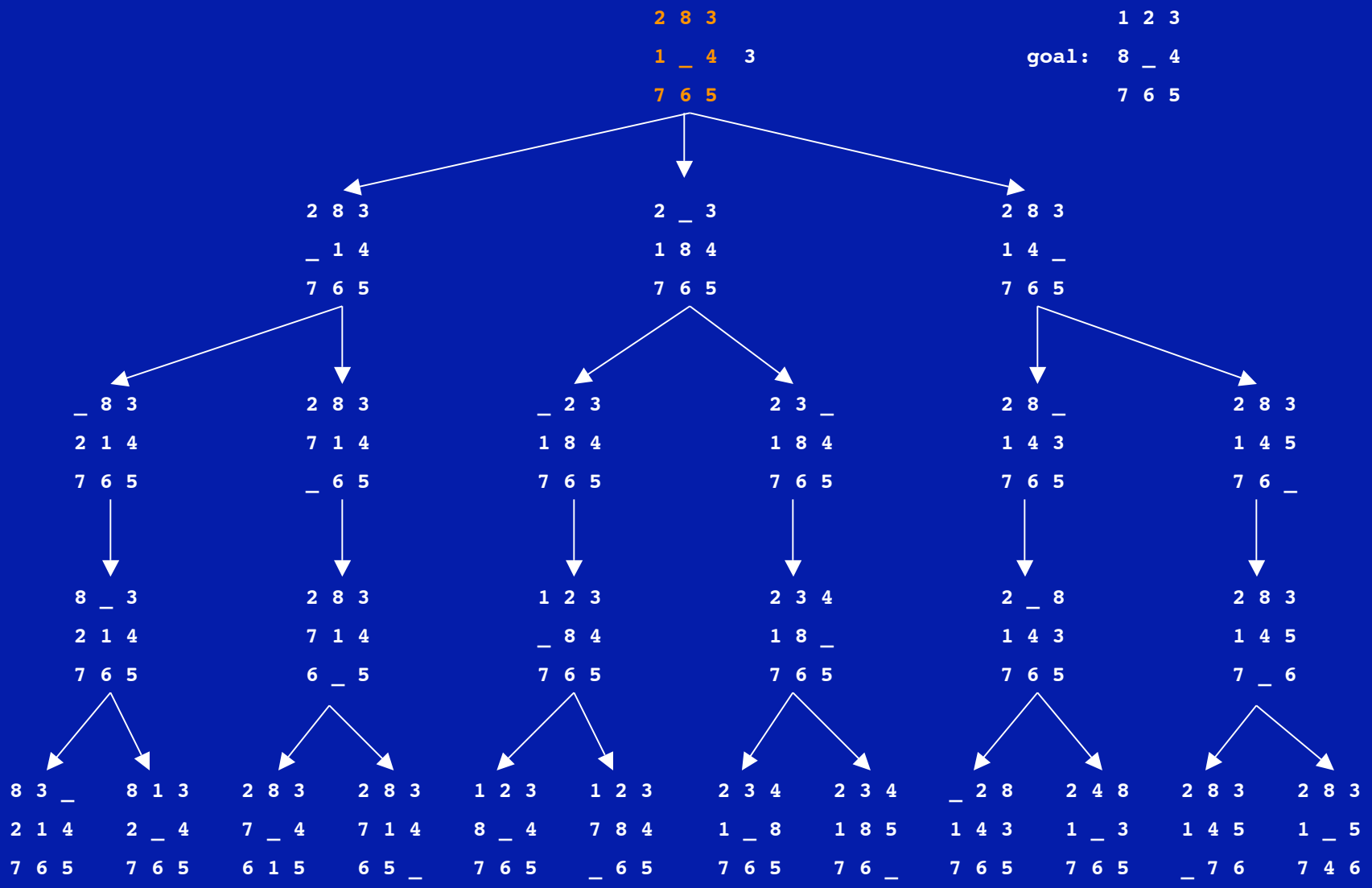
apply heuristic h(n) to start nodes
make a “list” of the start nodes - let’s call it the “frontier”
sort the frontier by h(n) values

repeat
  if no nodes on the frontier then terminate with failure
  choose one node from the front of the frontier and remove it
  if the chosen node matches the goal node
    then terminate with success
    else get next nodes (neighbors) and h(n) values
      and sort those nodes by h(n) values
      and put those sorted nodes on the front of the frontier
end repeat

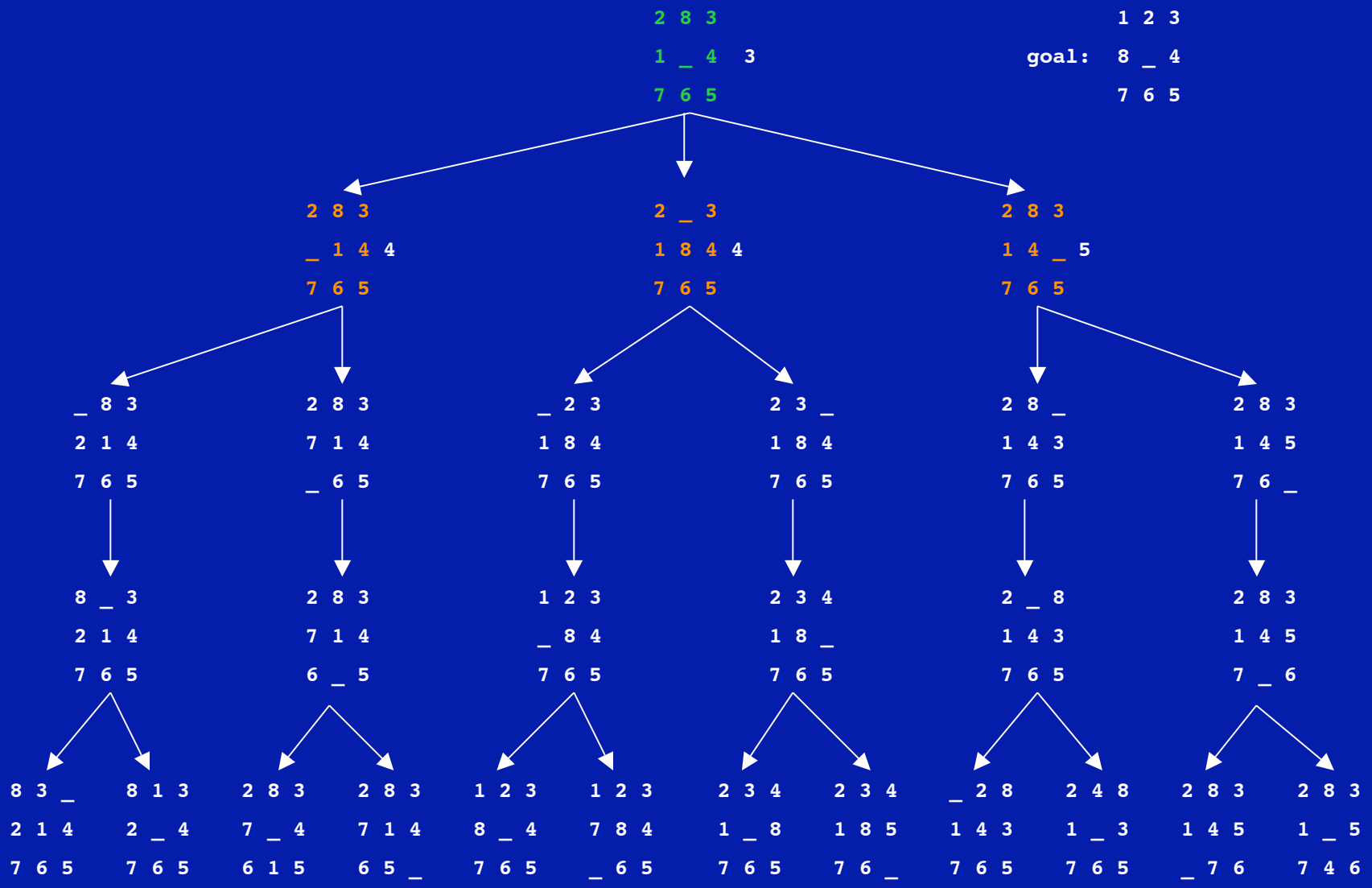
Heuristic depth-first search - tiles out of place



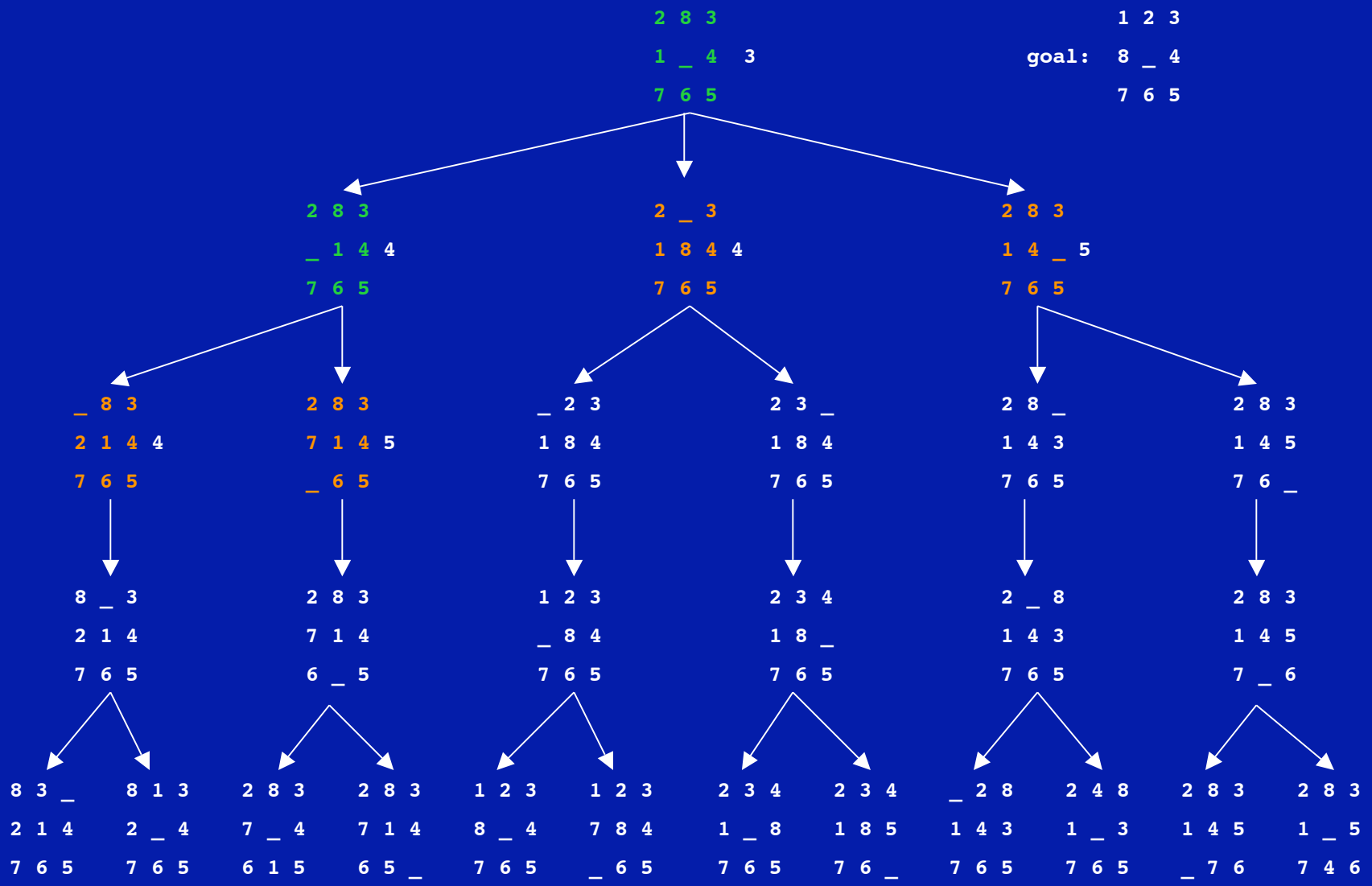
Heuristic depth-first search - tiles out of place



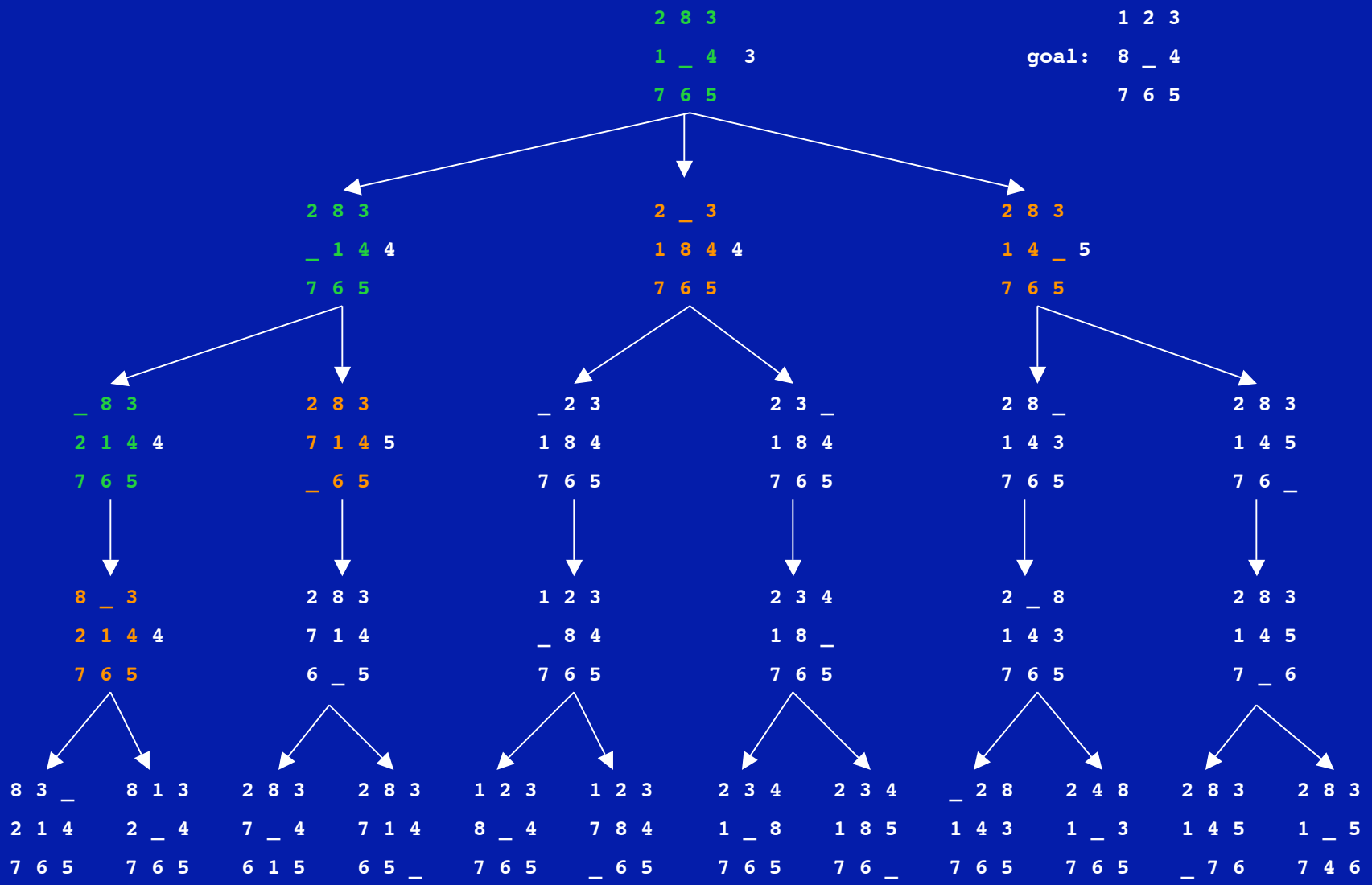
Heuristic depth-first search - tiles out of place



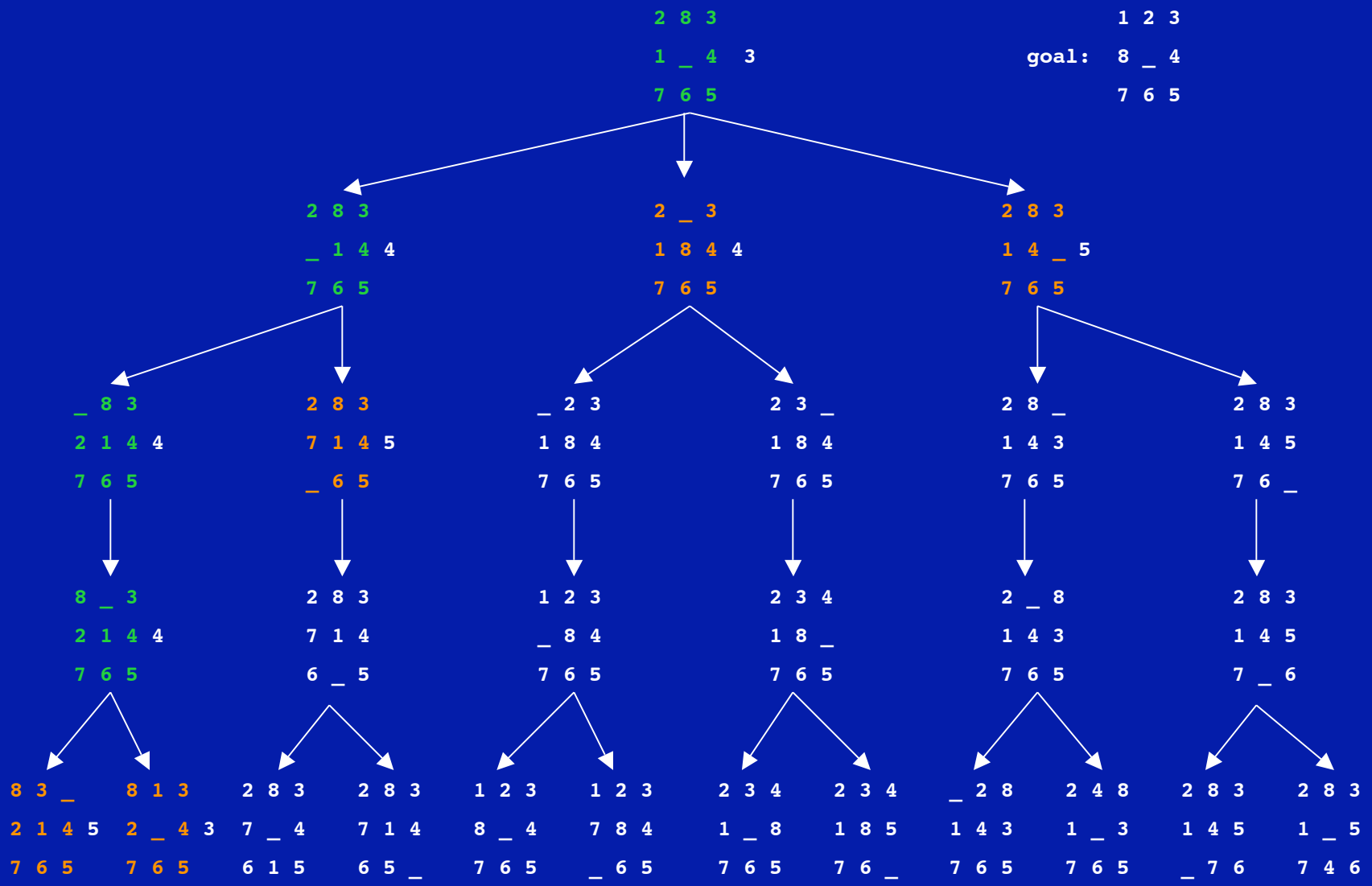
Heuristic depth-first search - tiles out of place



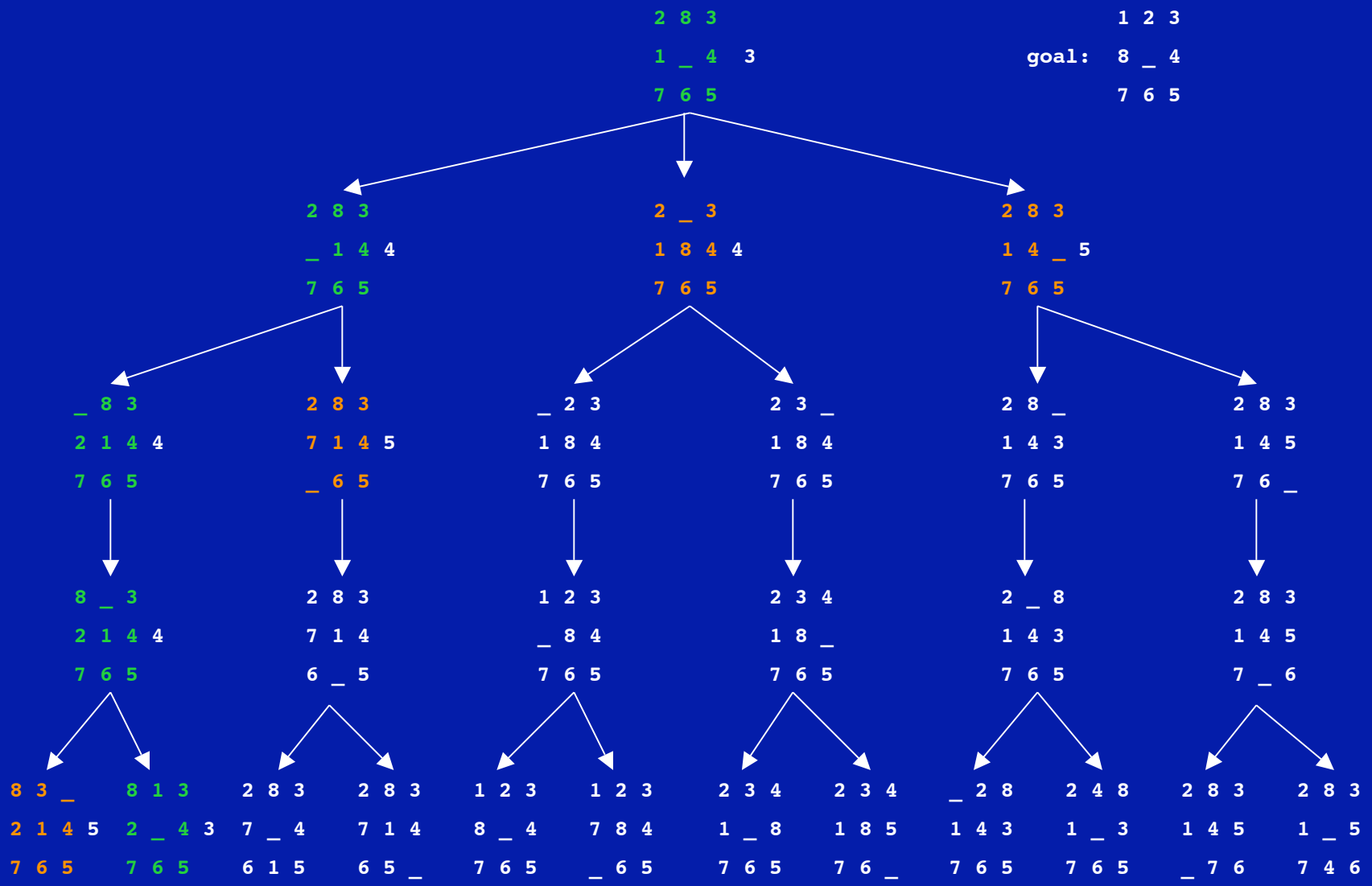
Heuristic depth-first search - tiles out of place



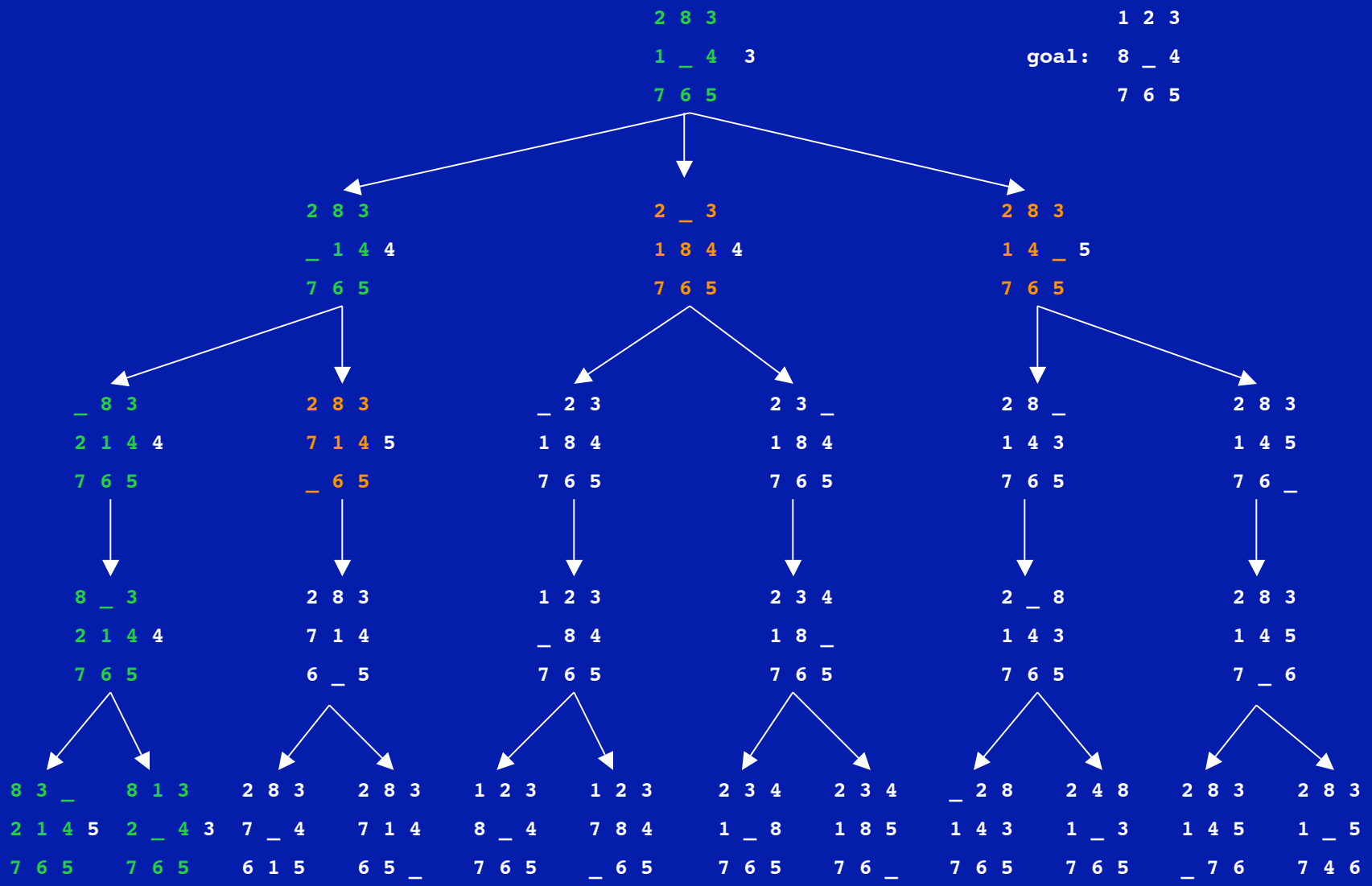
Heuristic depth-first search - tiles out of place



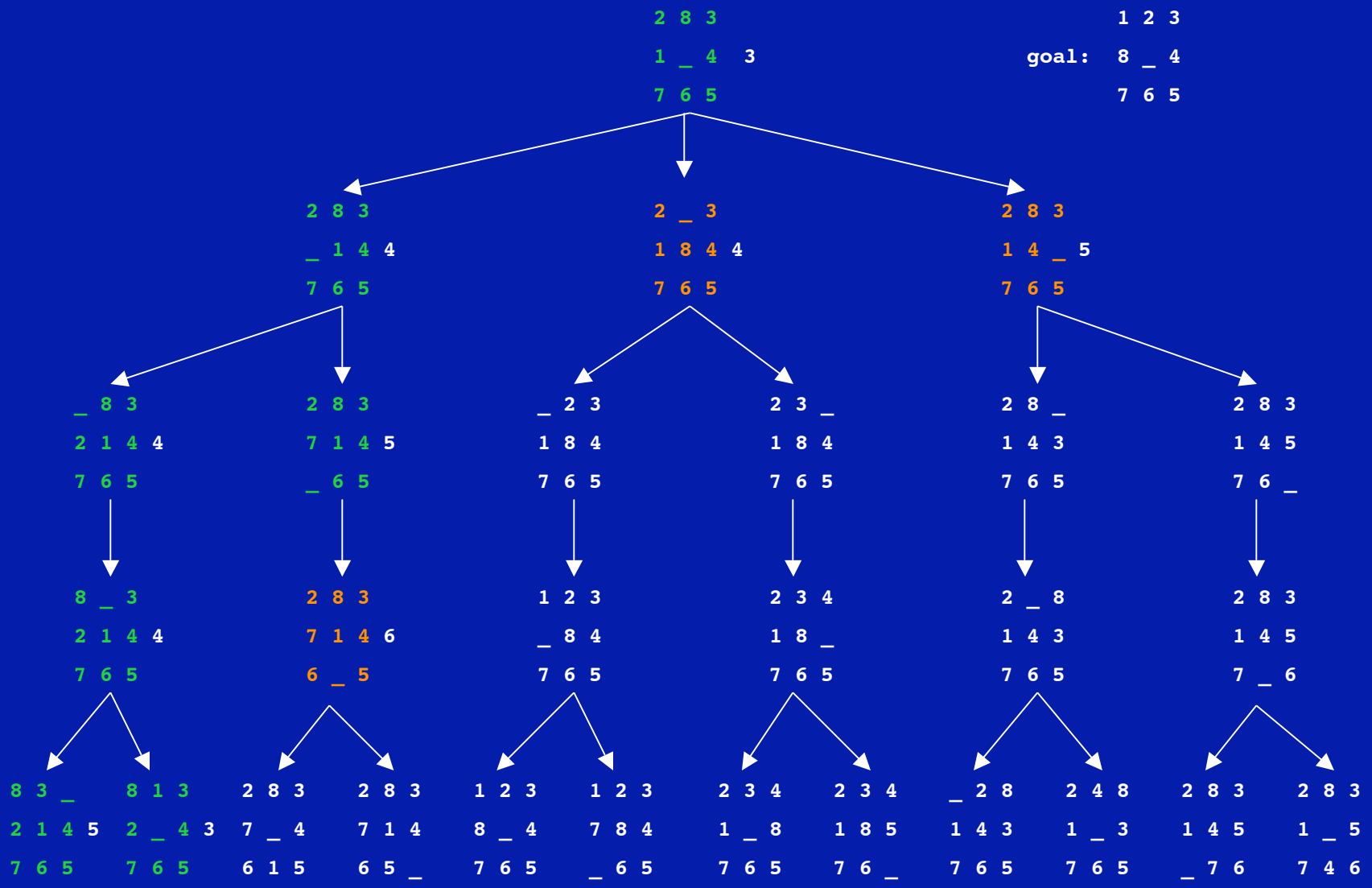
Heuristic depth-first search - tiles out of place



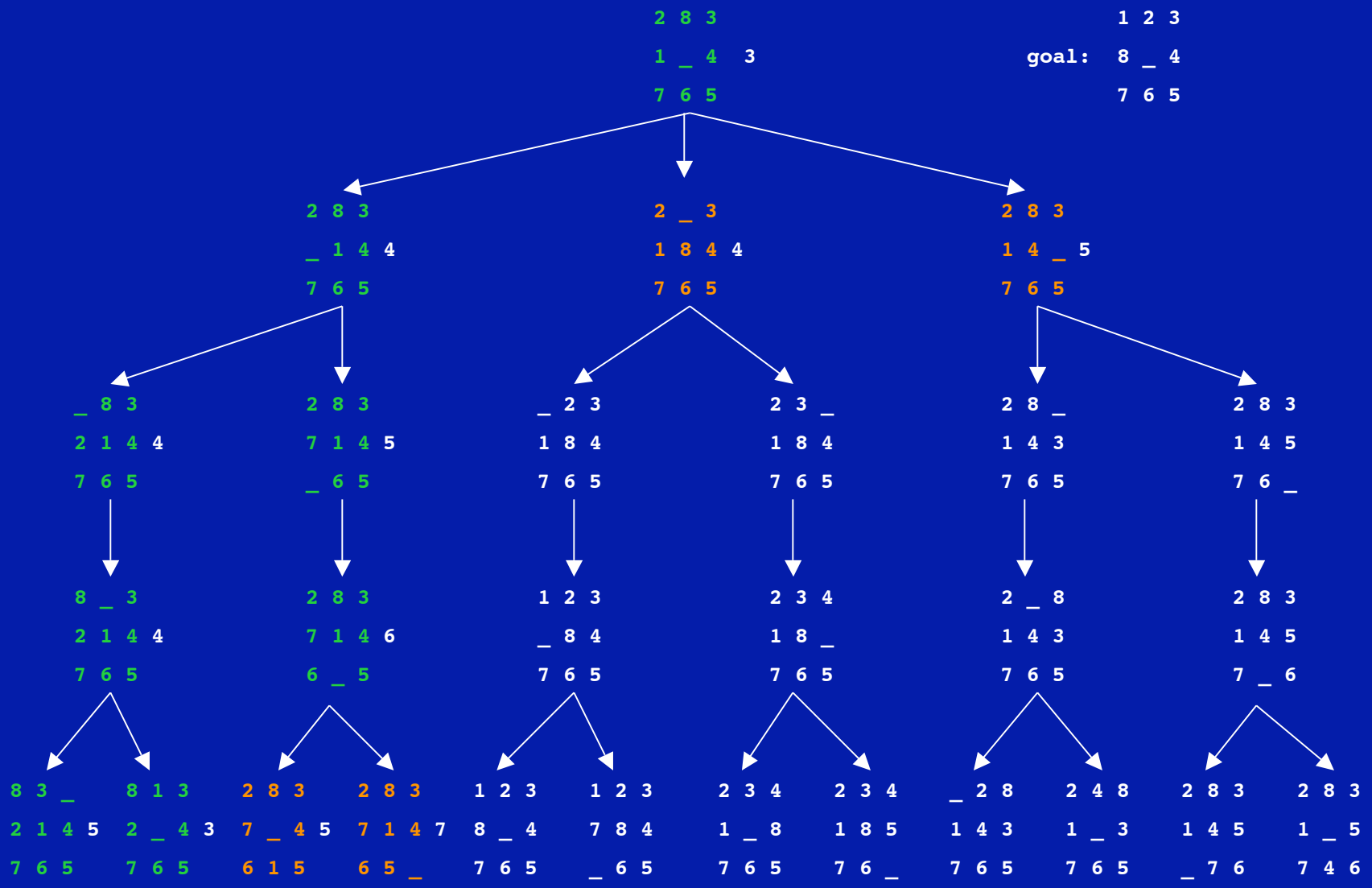
Heuristic depth-first search - tiles out of place



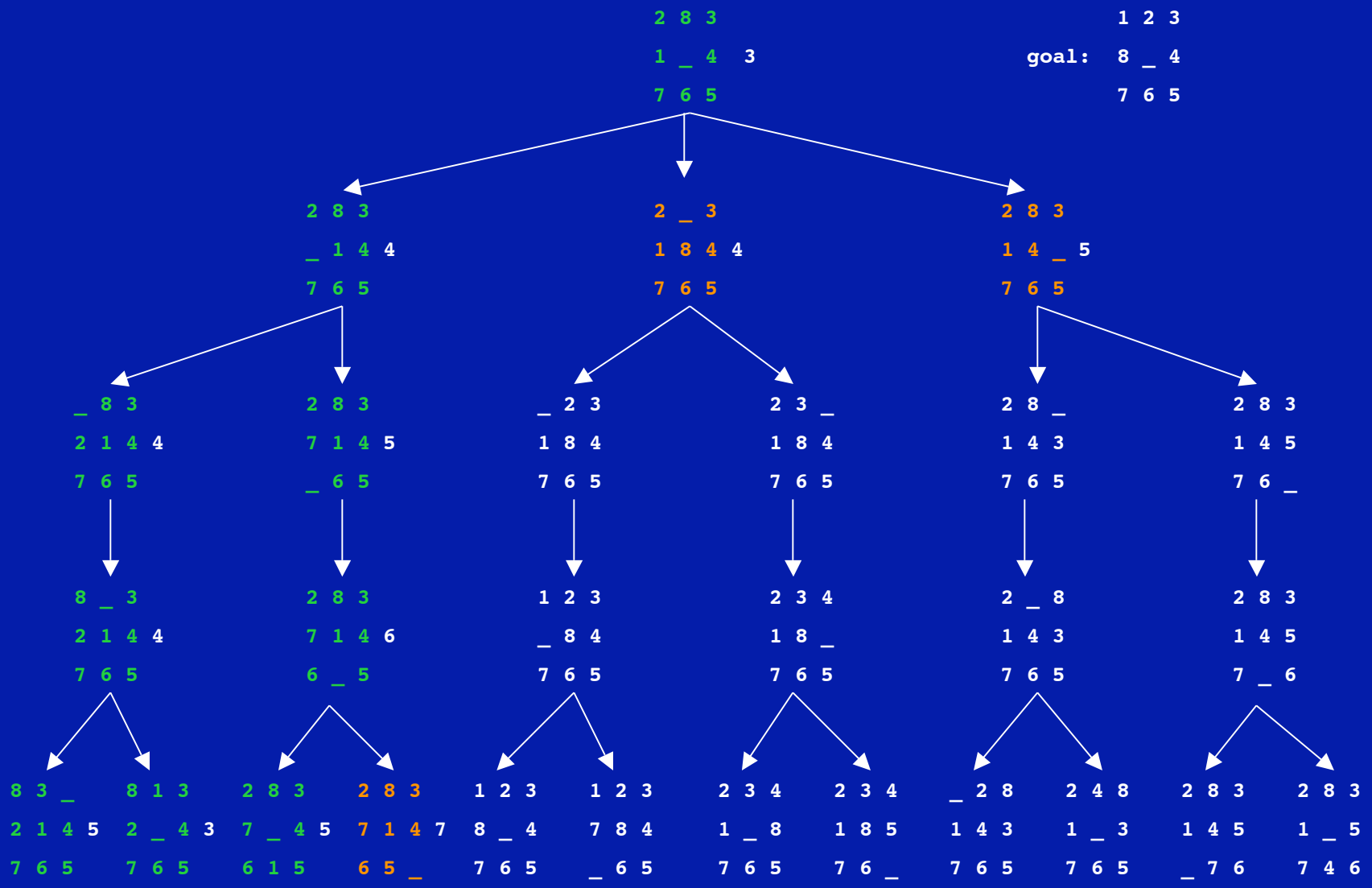
Heuristic depth-first search - tiles out of place



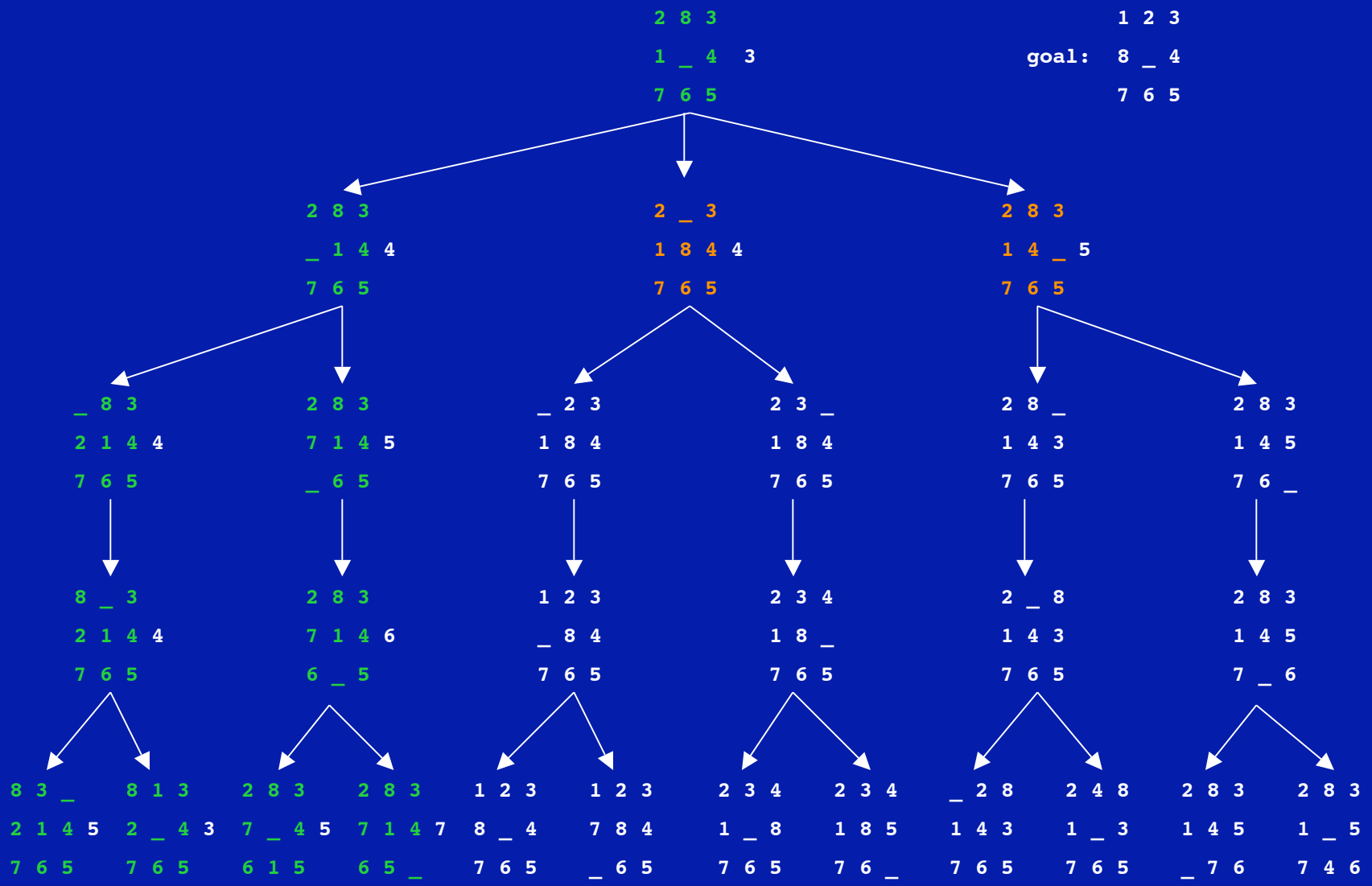
Heuristic depth-first search - tiles out of place



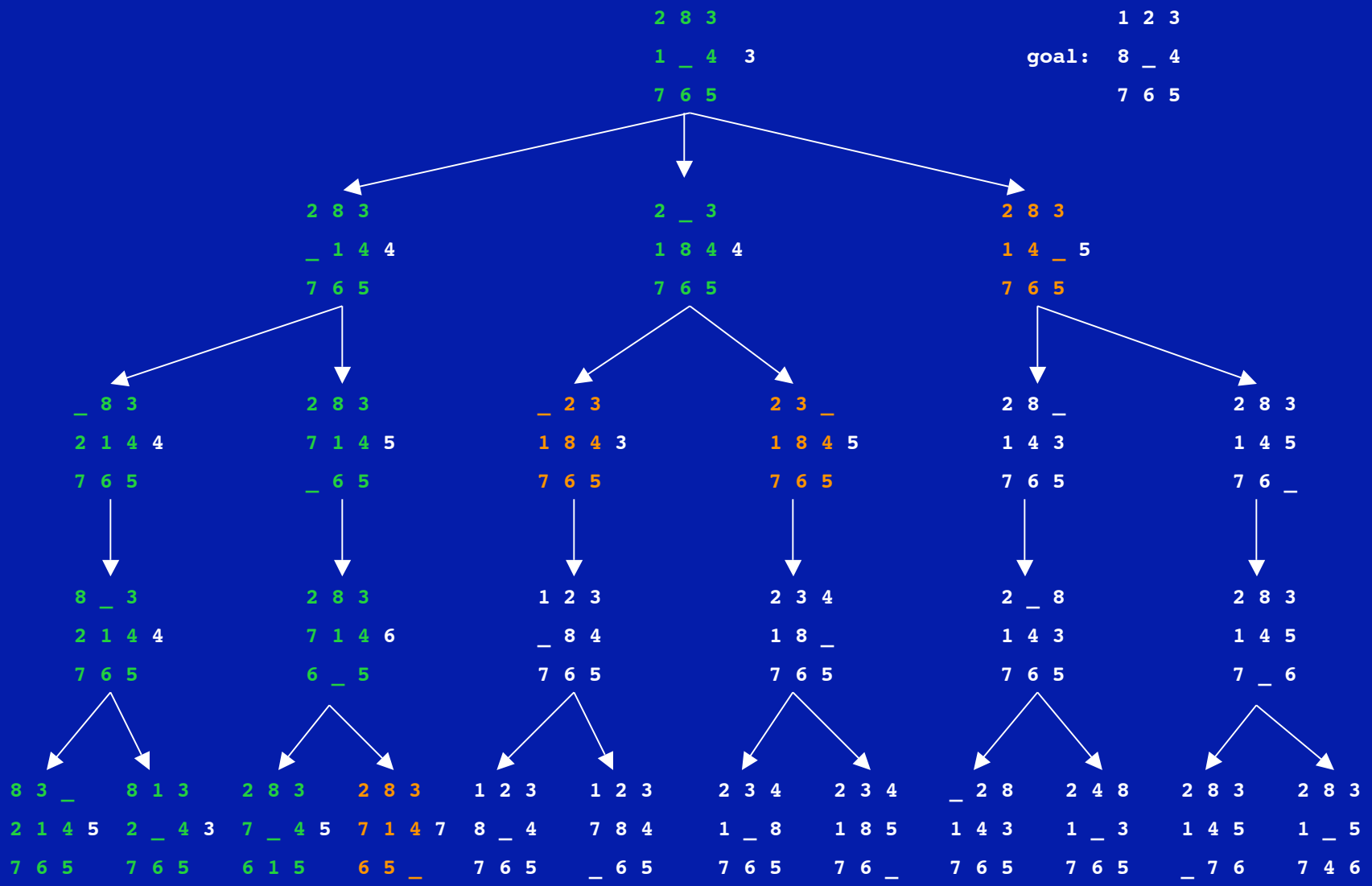
Heuristic depth-first search - tiles out of place



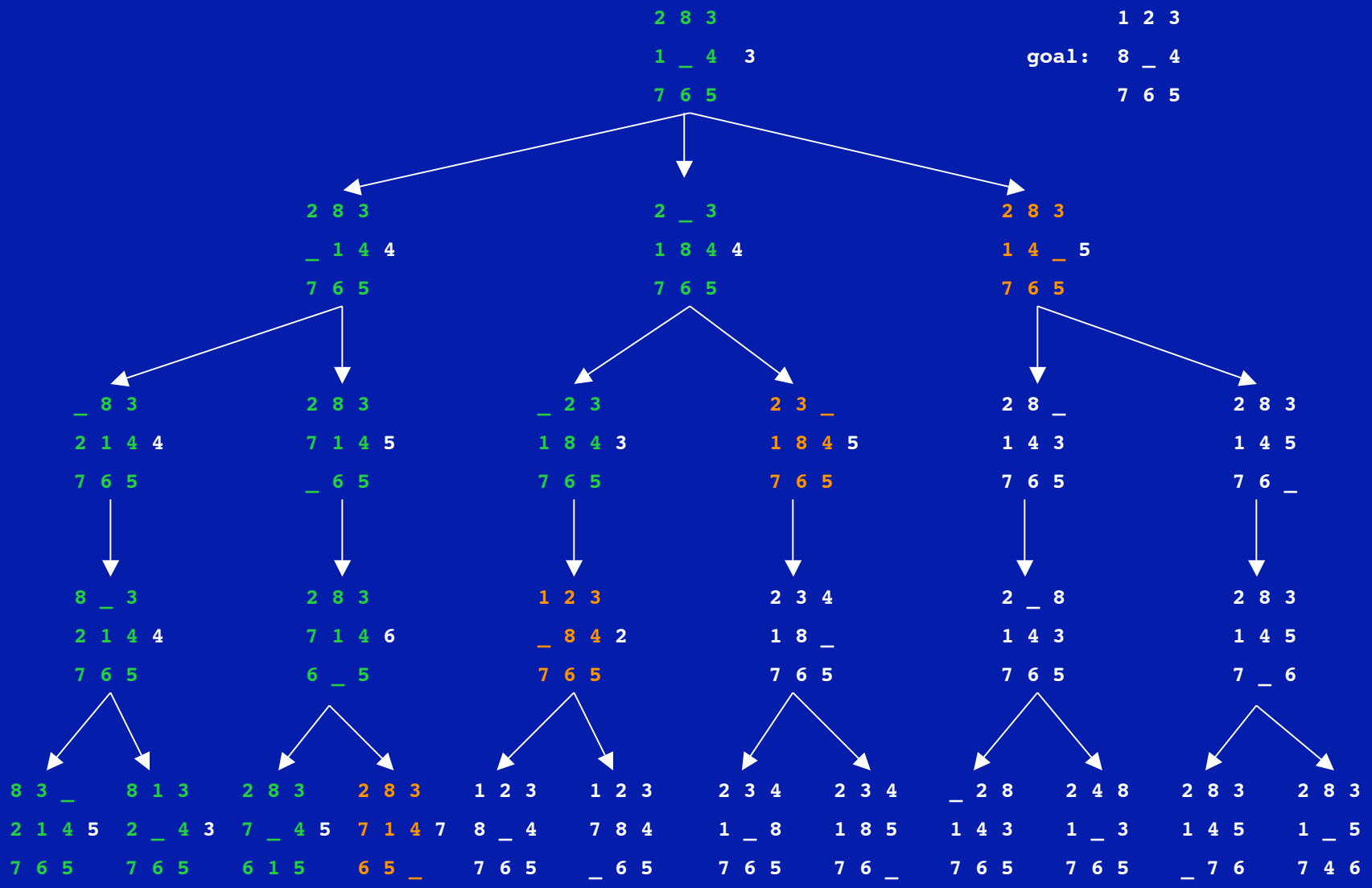
Heuristic depth-first search - tiles out of place



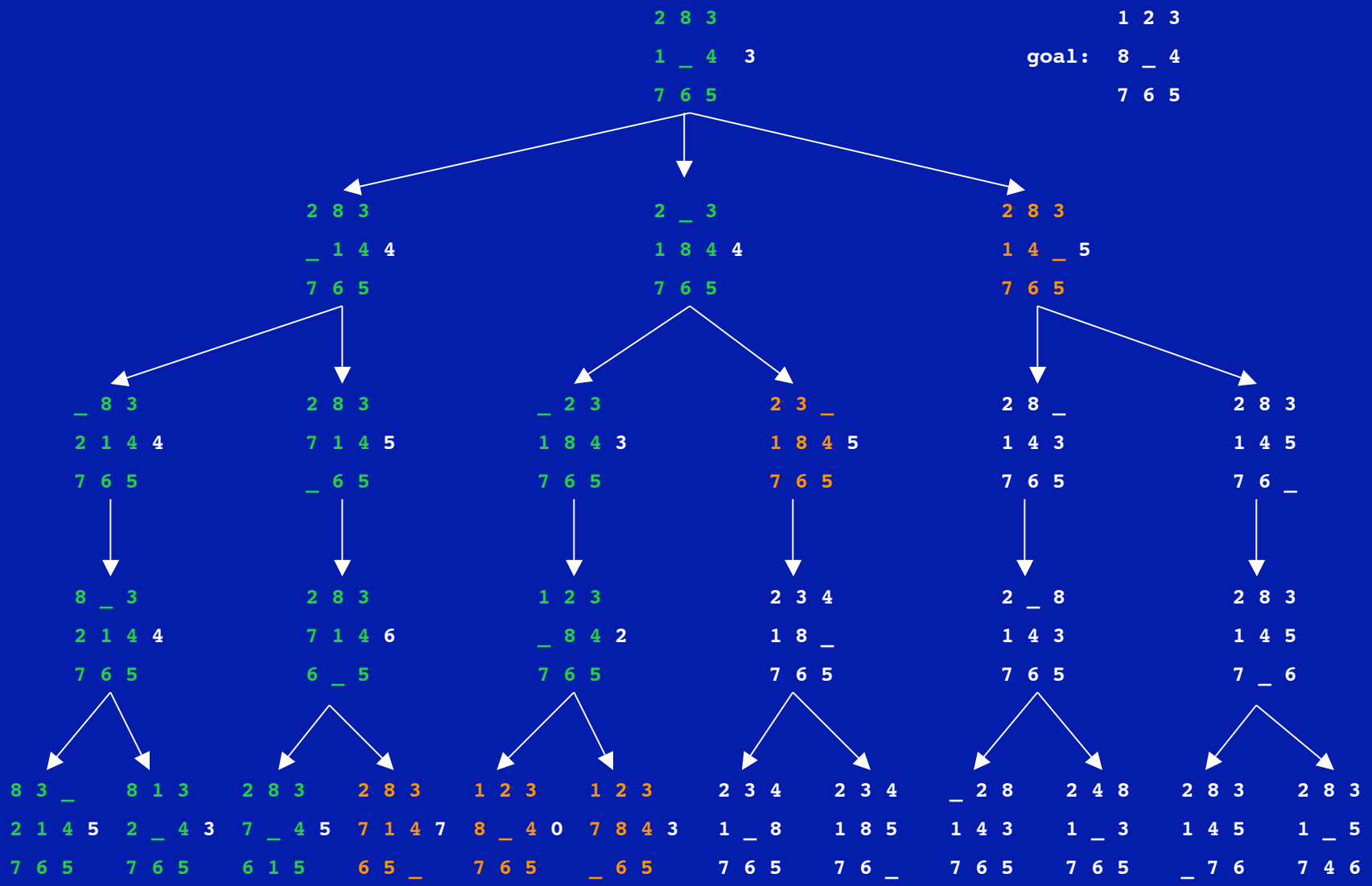
Heuristic depth-first search - tiles out of place



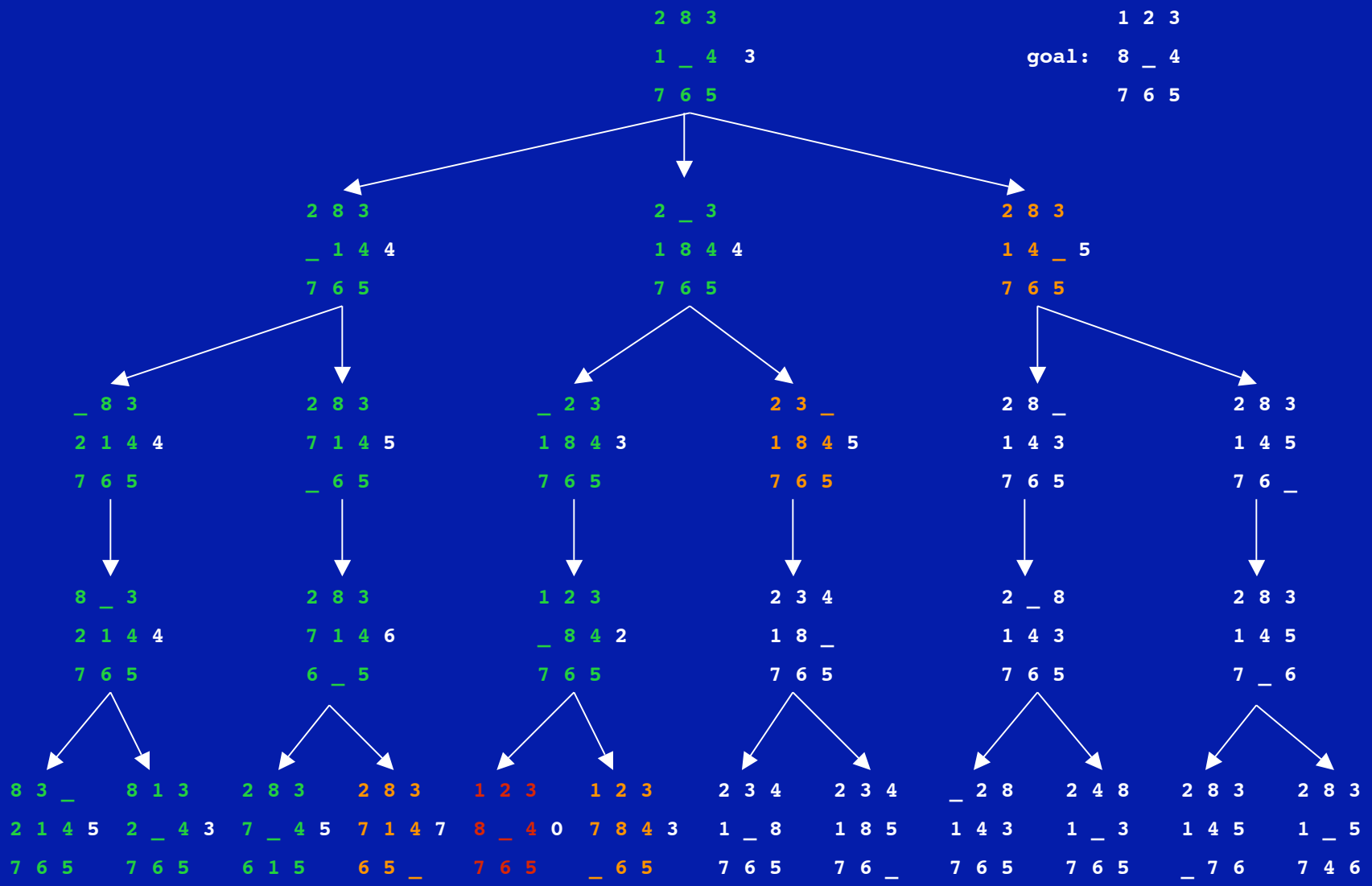
Heuristic depth-first search - tiles out of place



Heuristic depth-first search - tiles out of place



Heuristic depth-first search - tiles out of place



Why heuristic depth-first search?

Comparing search strategies

strategy	selection	halts?	space	optimal path
depth-first	last node added	no	linear	no
breadth-first	first node added	yes	exponential	yes[1]
lowest-cost first	minimal g(n)	yes	exponential	yes
best-first	globally minimal h(n)	no	exponential	no
heuristic depth-first	locally minimal h(n)	no	linear	no

1. assuming all arcs have the same cost

What's missing?

A heuristic search strategy that is guaranteed to find the optimal (lowest-cost) path from start node to goal node

What do the strategies that find optimal paths have in common?

Comparing search strategies

strategy	selection	halts?	space	optimal path
depth-first	last node added	no	linear	no
breadth-first	first node added	yes	exponential	yes[1]
lowest-cost first	minimal g(n)	yes	exponential	yes
best-first	globally minimal h(n)	no	exponential	no
heuristic depth-first	locally minimal h(n)	no	linear	no

1. assuming all arcs have the same cost

Would this work?

The search strategies that find optimal paths all take g(n) -- the cost of the path from start node to frontier node -- into consideration...

...and best-first search seems like a reasonably sane heuristic search (although it's exponential)...

...what would happen if we rewrite best-first search to include g(n) as well as h(n)?

Oh, by the way, we give the name f(n) to the sum of g(n) and h(n), as in f(n) = g(n) + h(n)

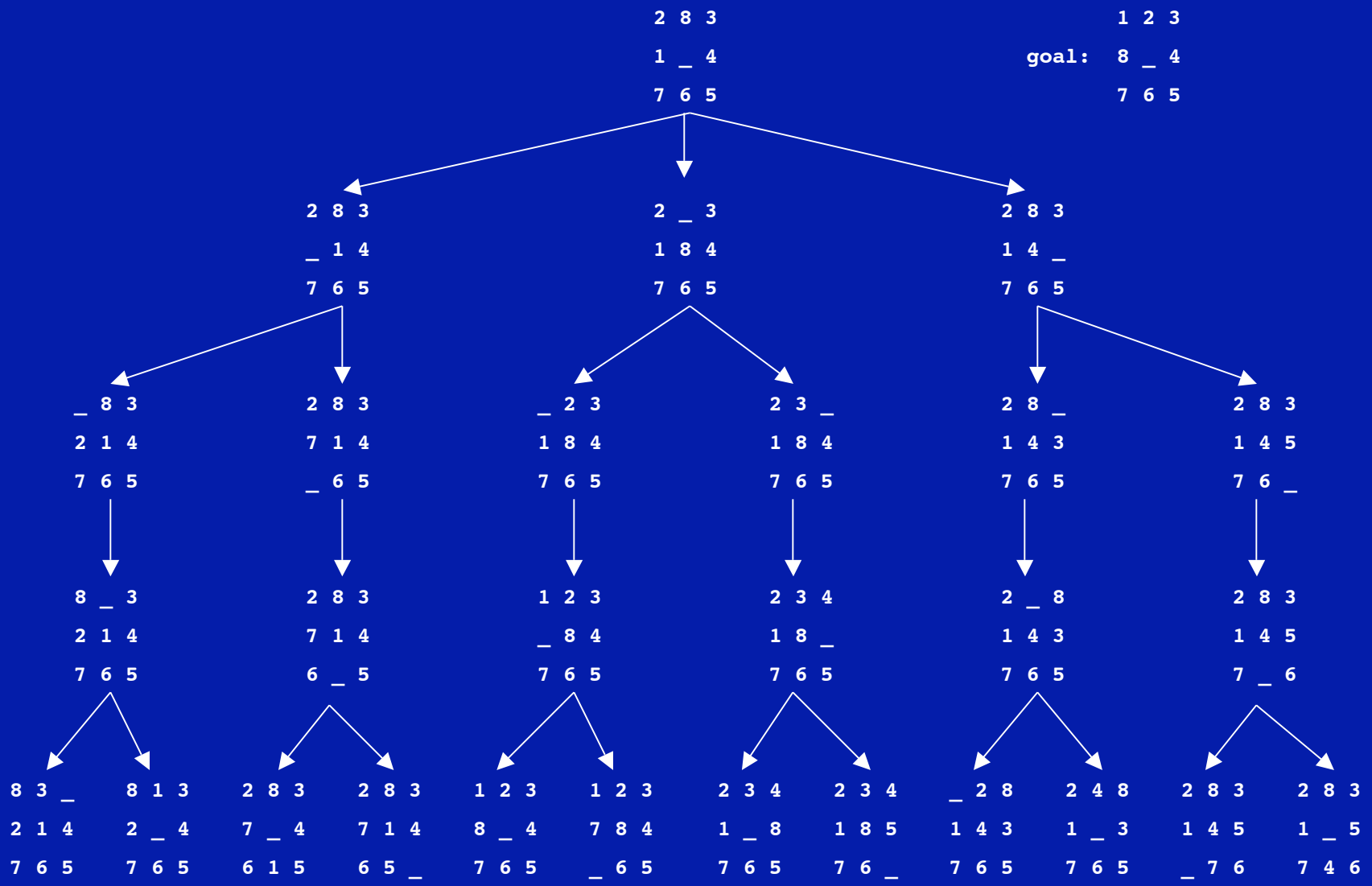
Refresher:
Heuristic best-first search algorithm

Given a set of start nodes, a set of goal nodes,
and a graph (i.e., the nodes and arcs):

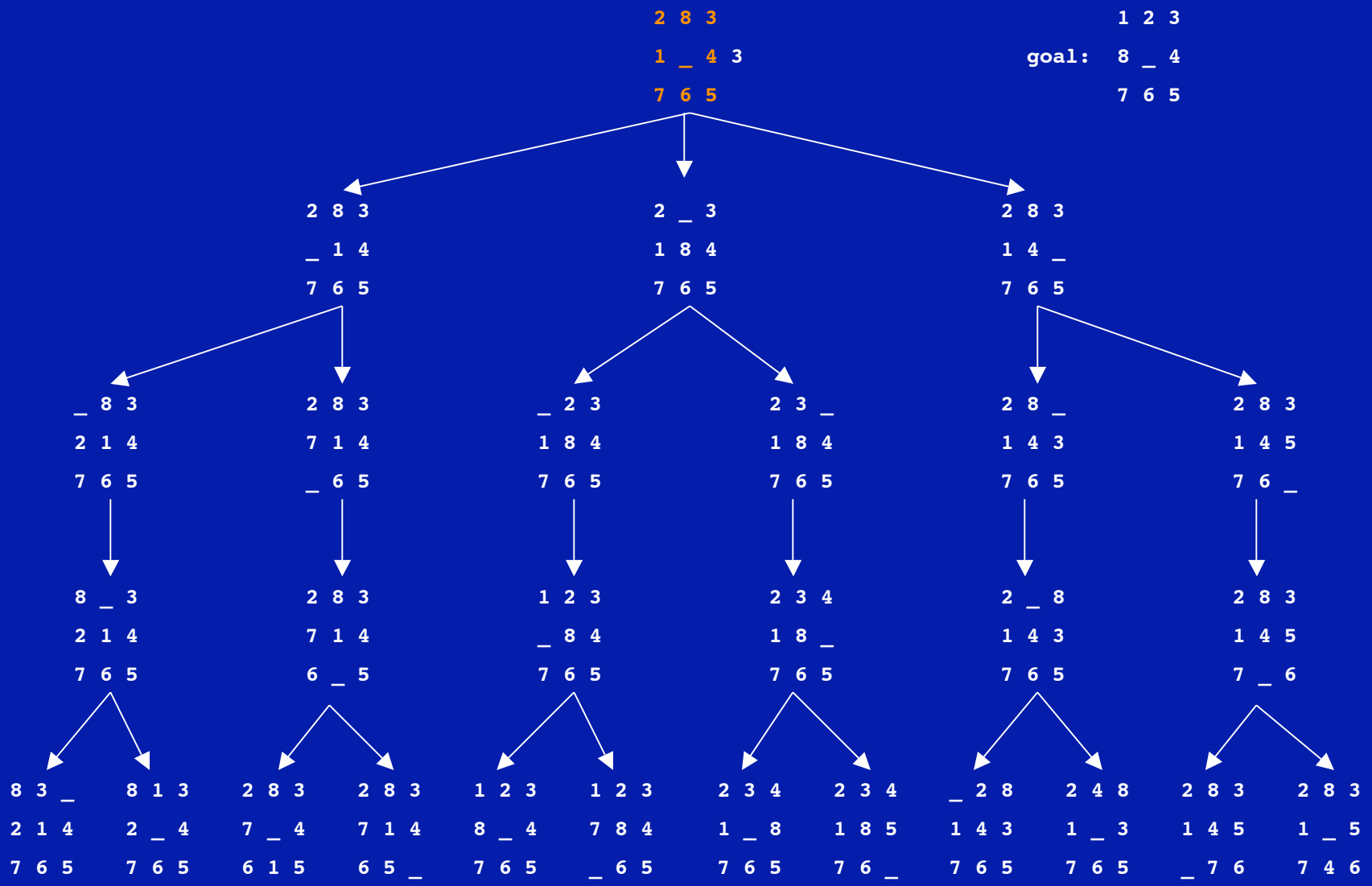
apply heuristic h(n) to start nodes
make a “list” of the start nodes - let’s call it the “frontier”
sort the frontier by h(n) values

repeat
  if no nodes on the frontier then terminate with failure
  choose one node from the front of the frontier and remove it
  if the chosen node matches the goal node
    then terminate with success
    else put next nodes (neighbors) and h(n) values on frontier
        and sort frontier by h(n) values
end repeat

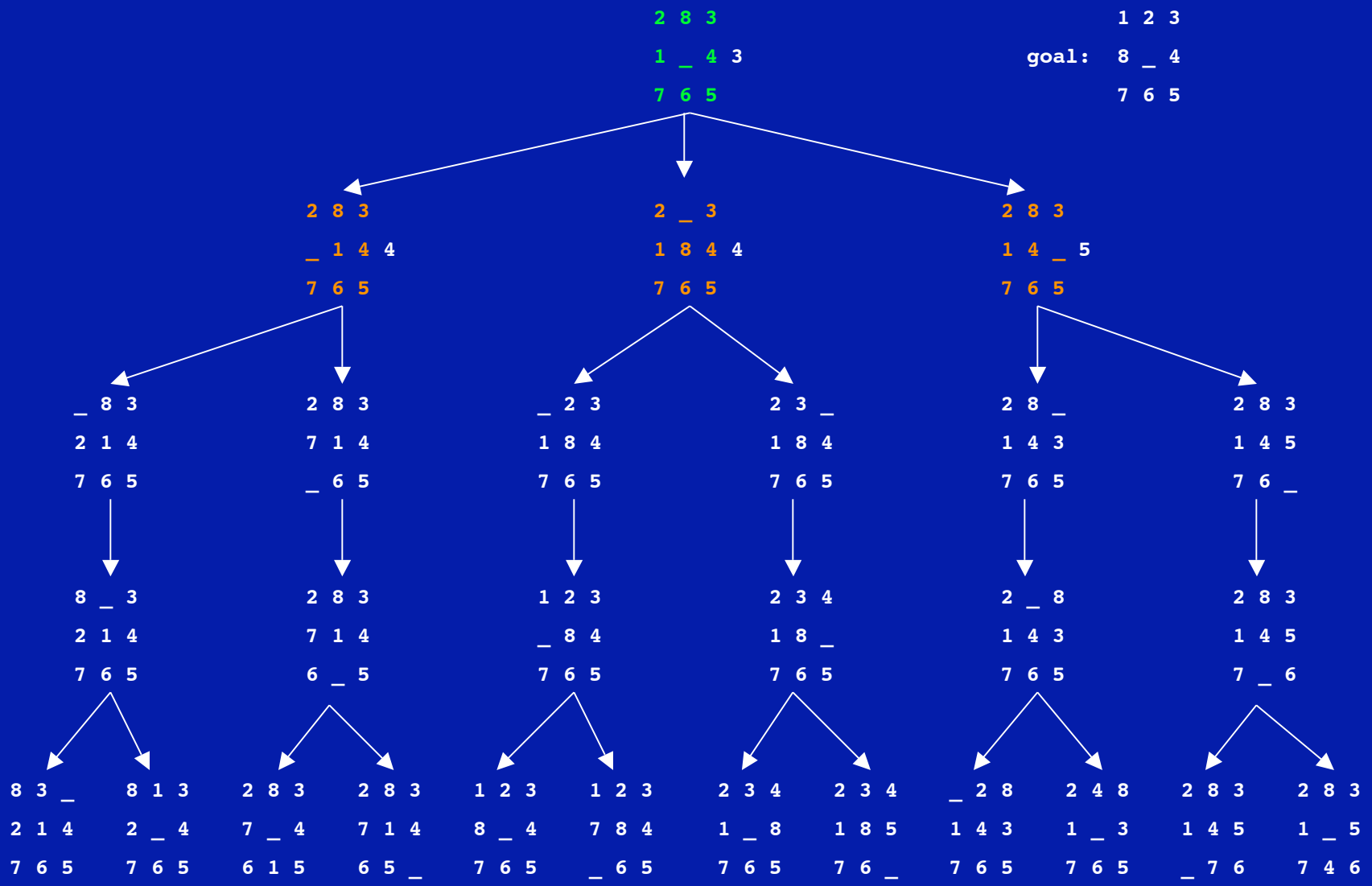
Heuristic best-first search - tiles out of place



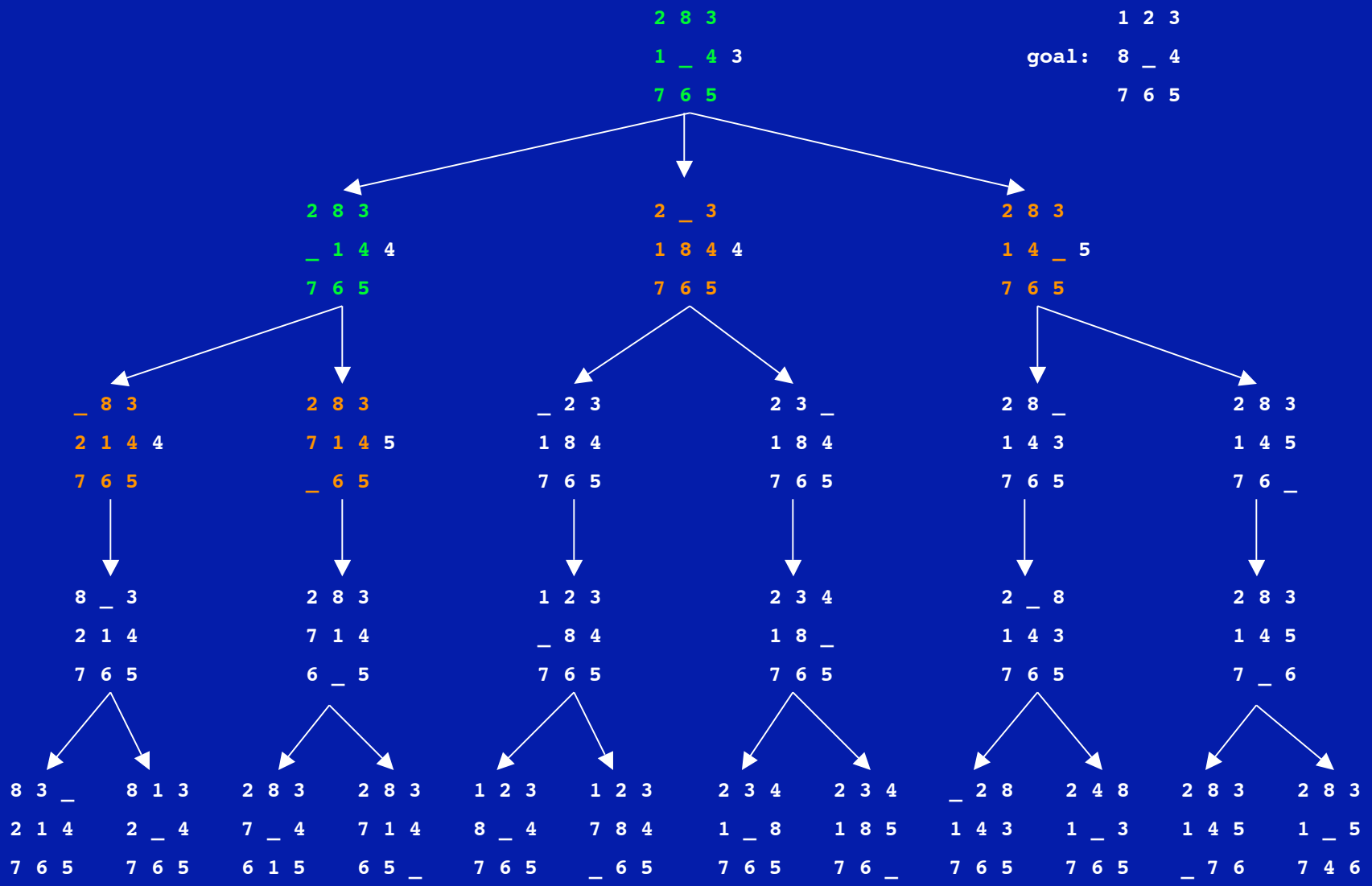
Heuristic best-first search - tiles out of place



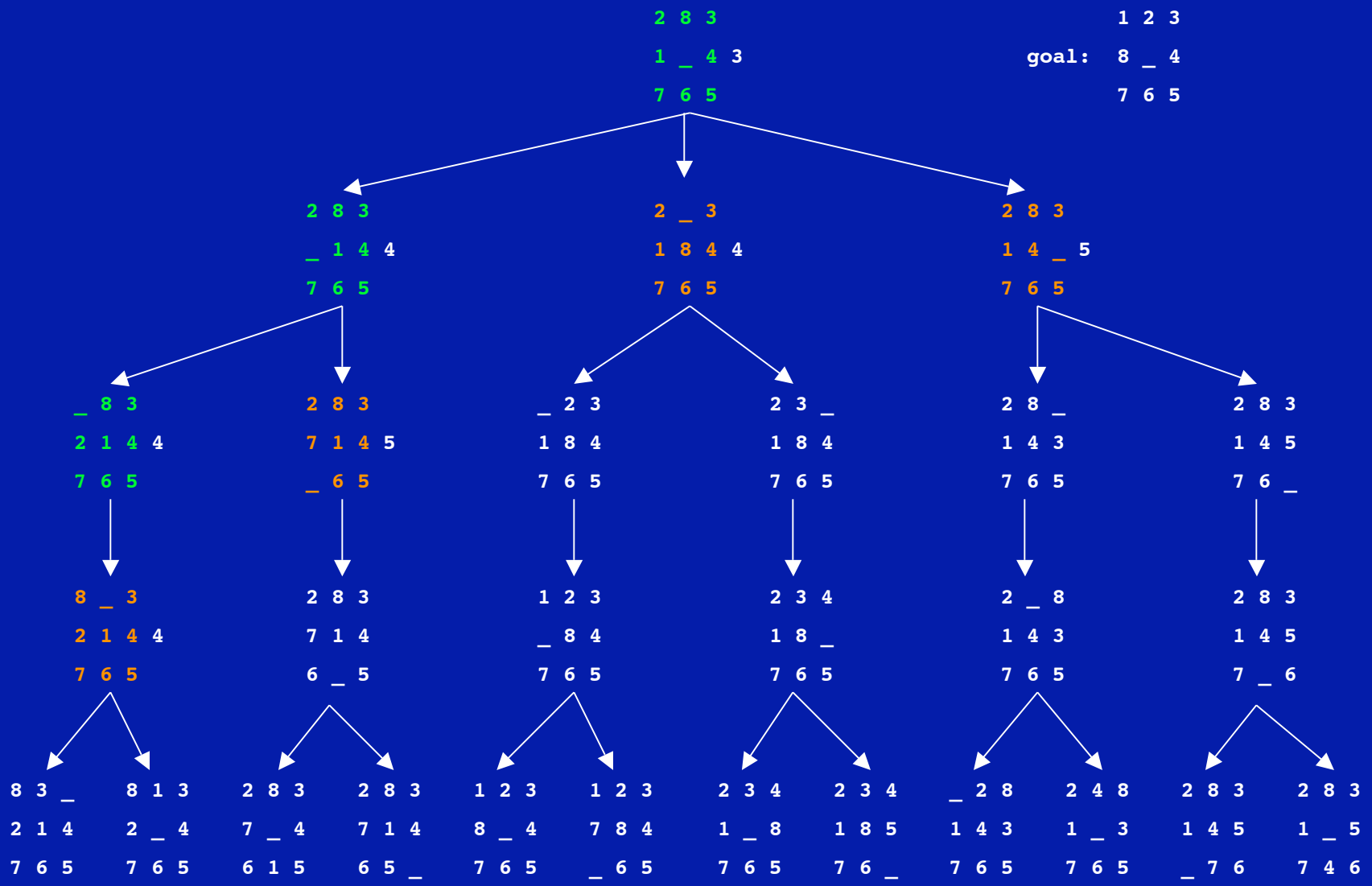
Heuristic best-first search - tiles out of place



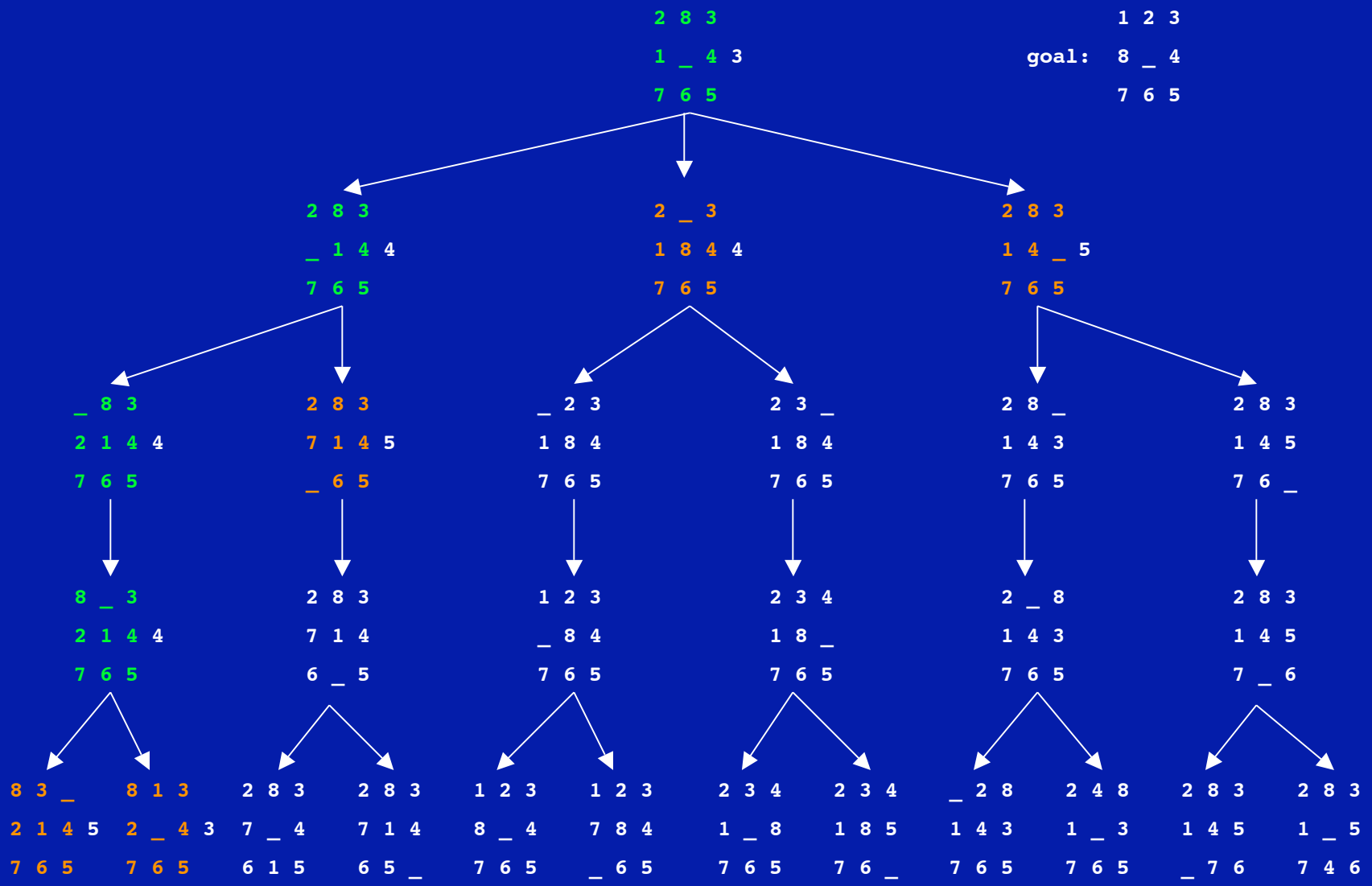
Heuristic best-first search - tiles out of place



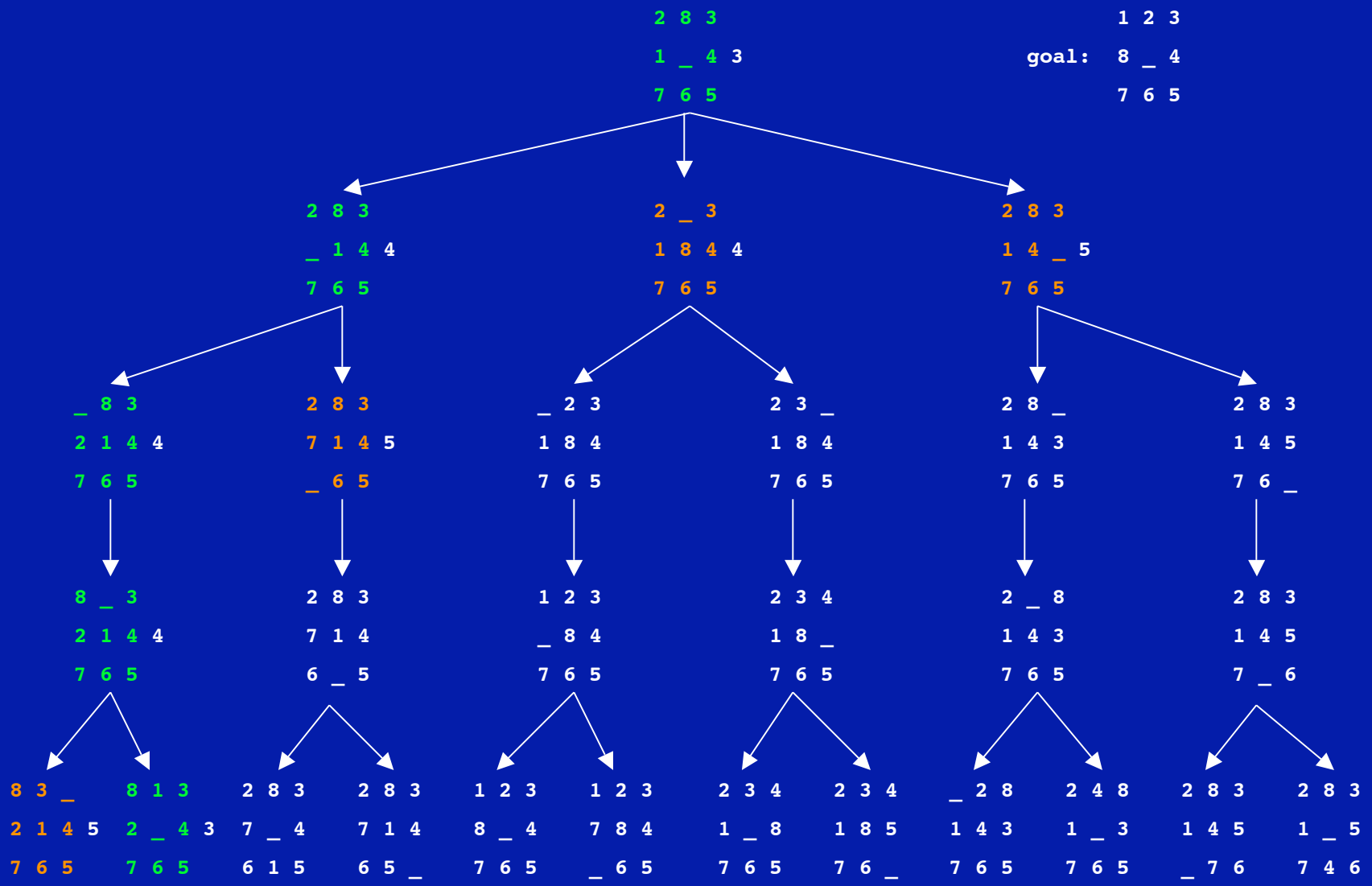
Heuristic best-first search - tiles out of place



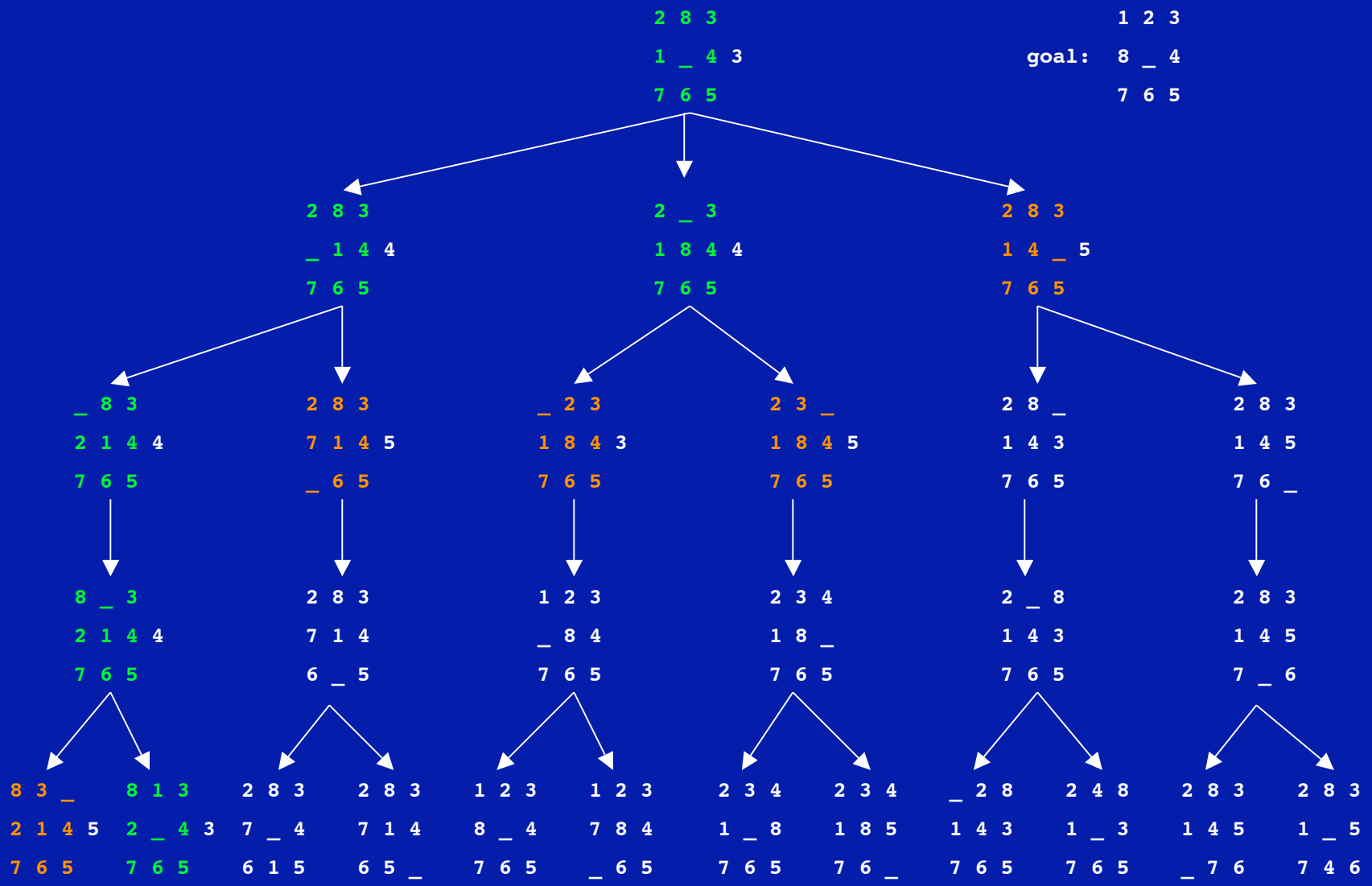
Heuristic best-first search - tiles out of place



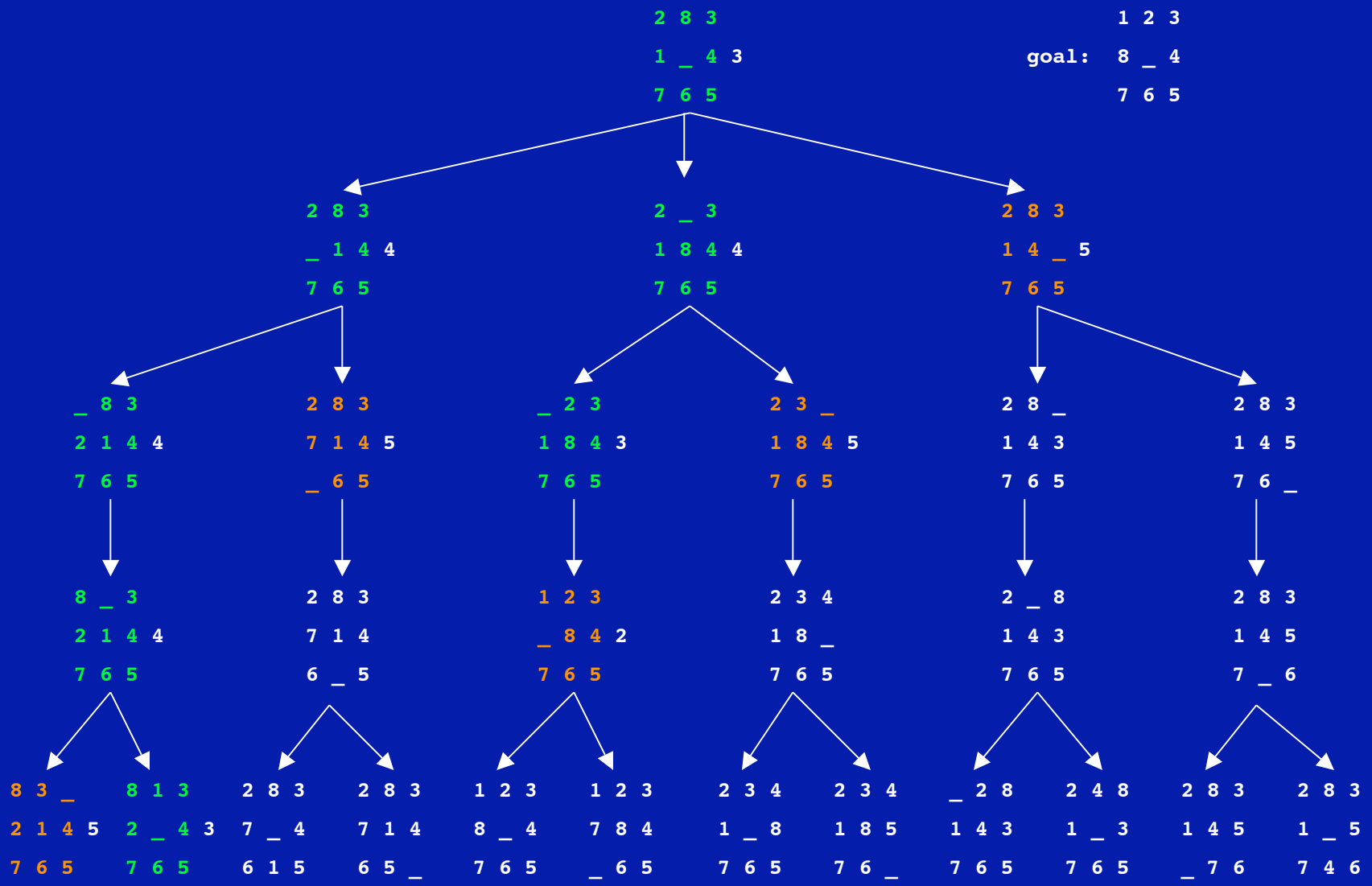
Heuristic best-first search - tiles out of place



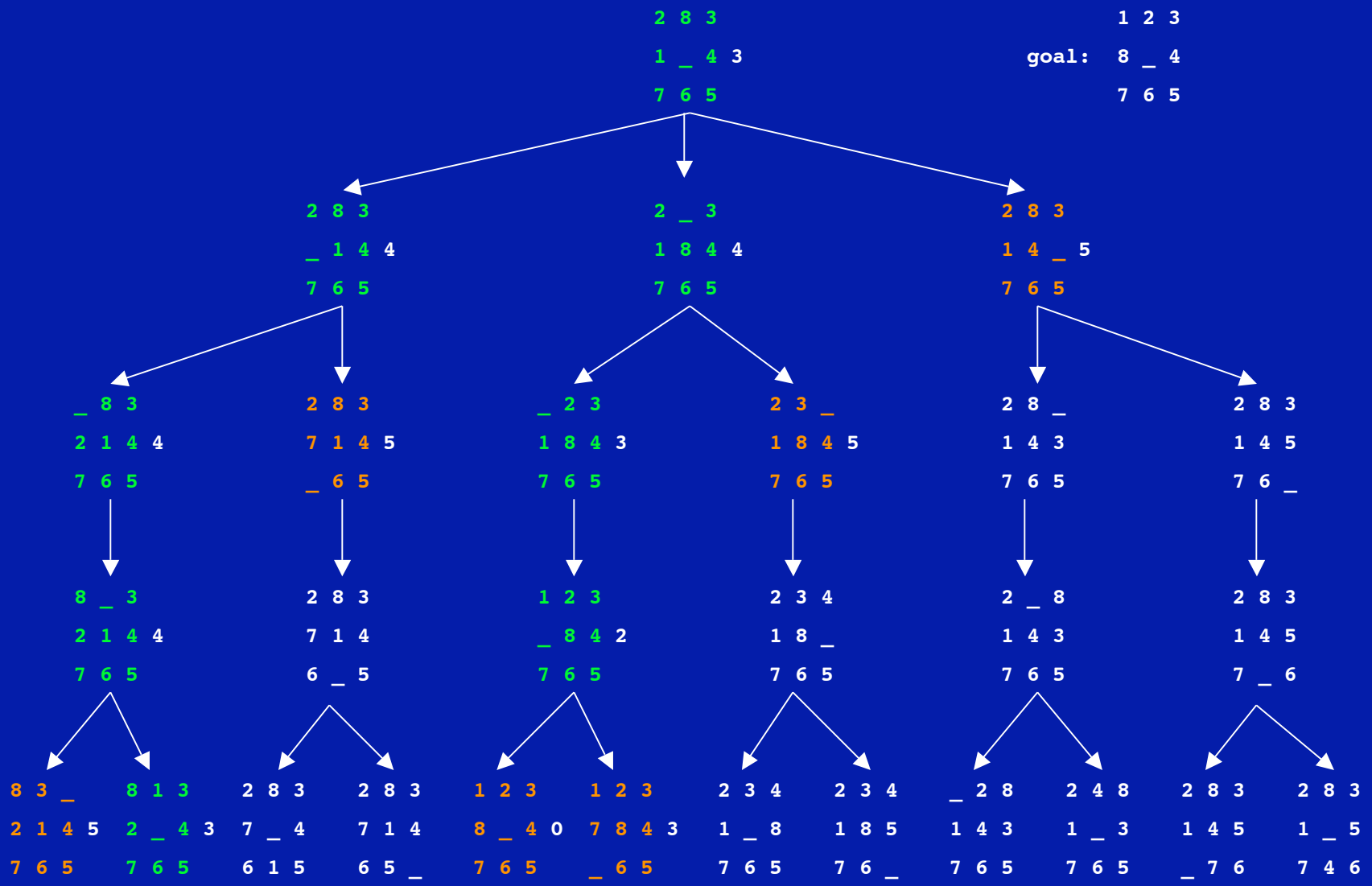
Heuristic best-first search - tiles out of place



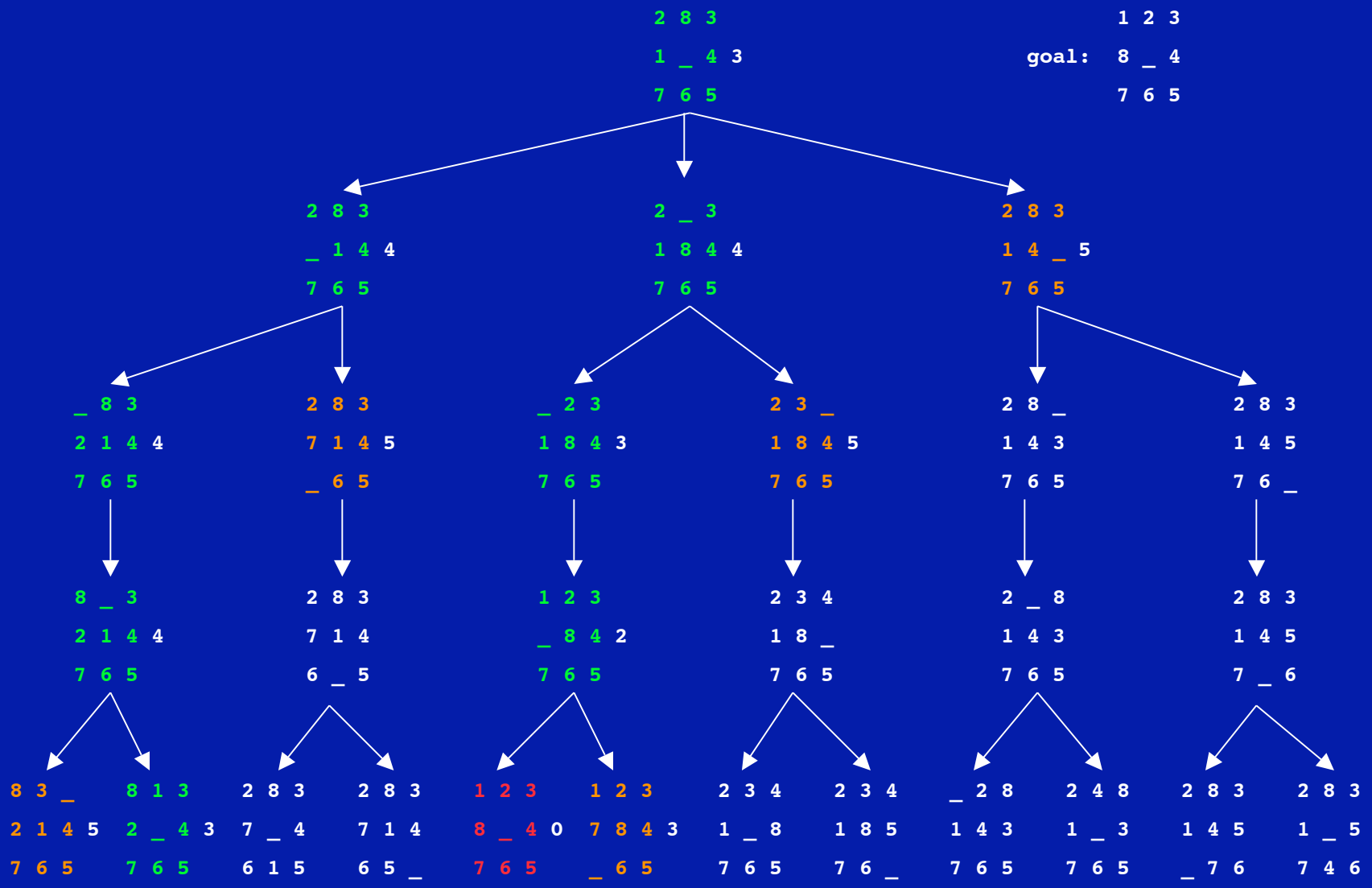
Heuristic best-first search - tiles out of place



Heuristic best-first search - tiles out of place



Heuristic best-first search - tiles out of place



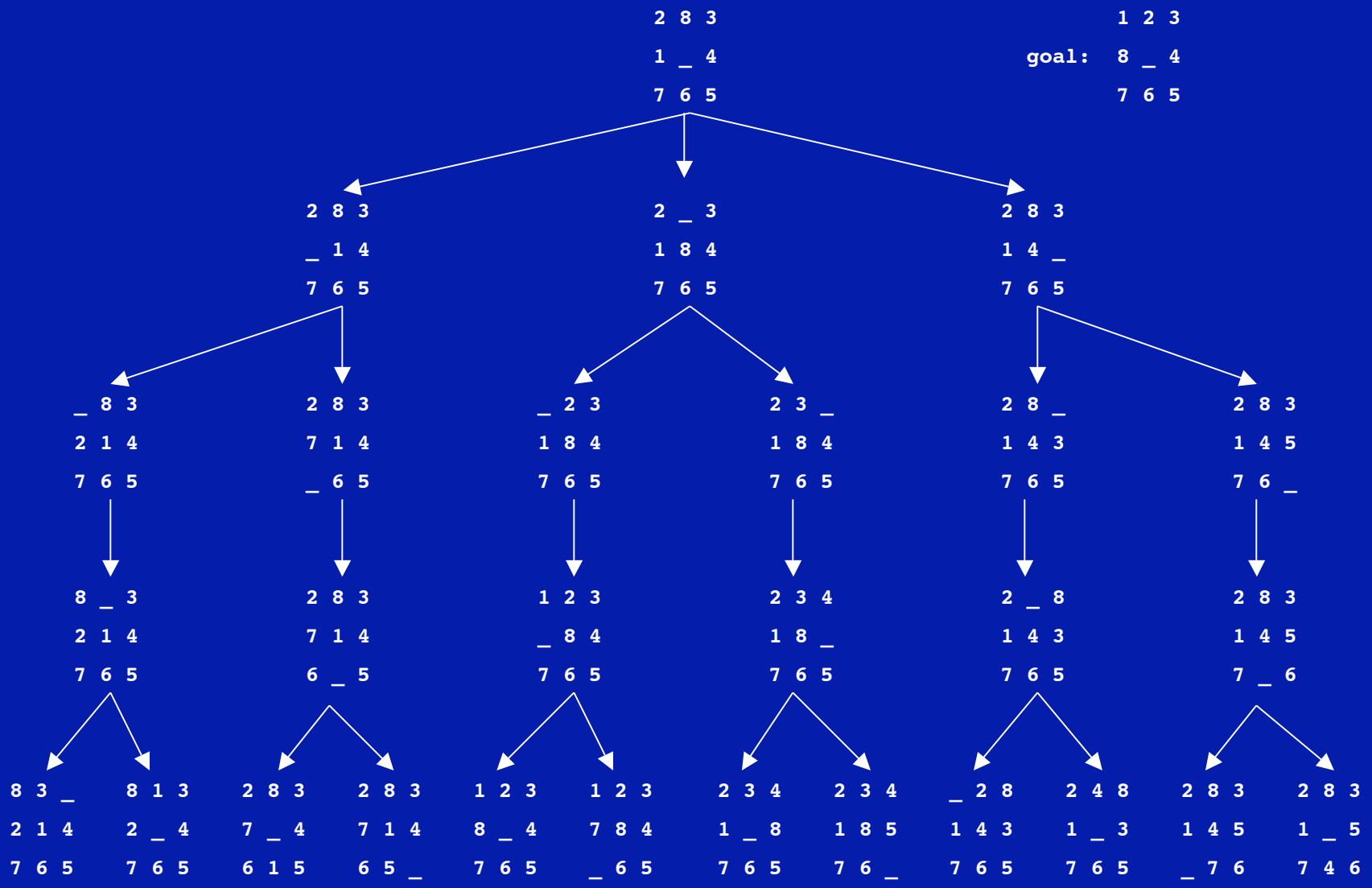
Best-first with g(n) included

Given a set of start nodes, a set of goal nodes,
and a graph (i.e., the nodes and arcs):

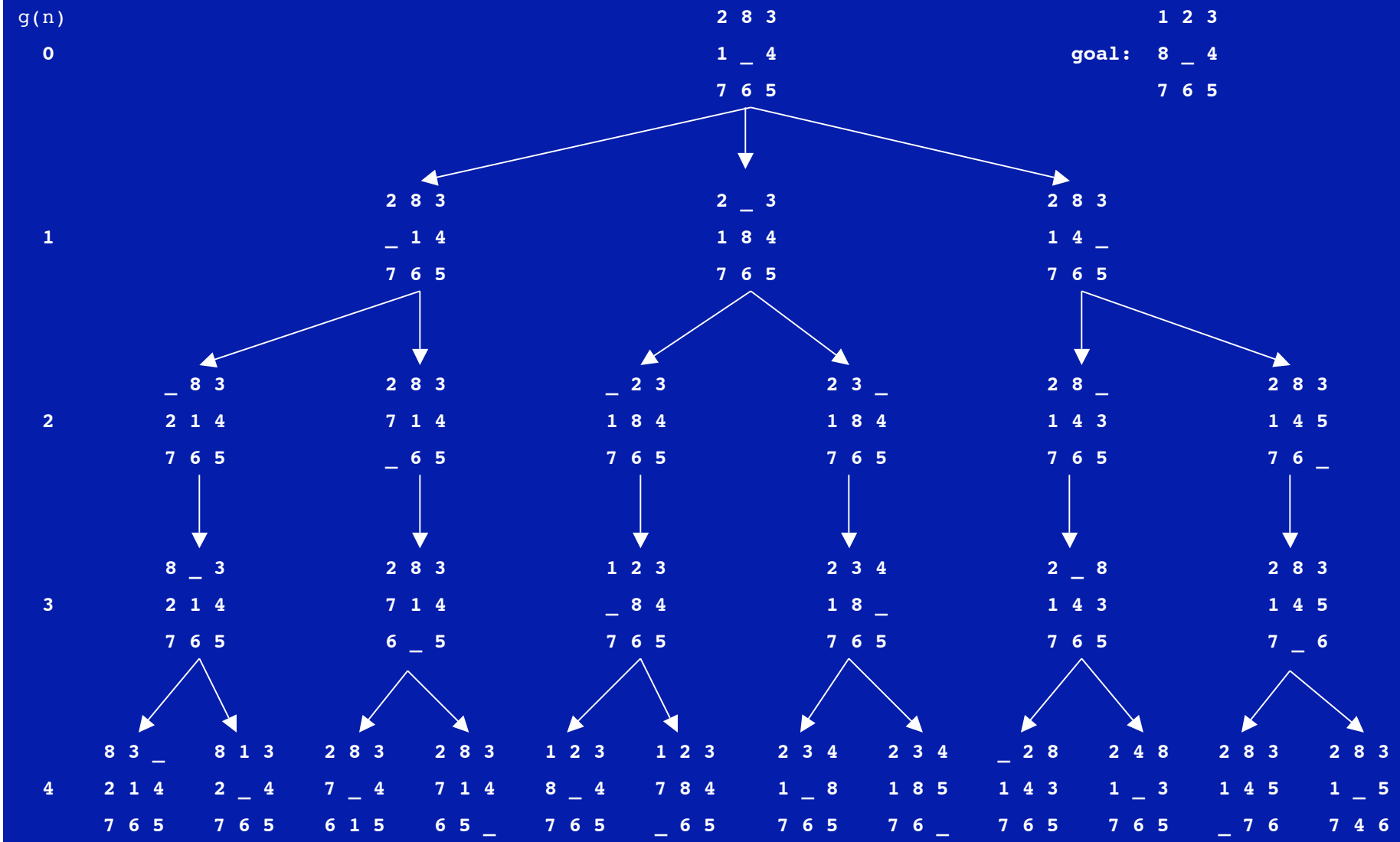
apply heuristic g(n)+h(n) to start nodes
make a “list” of the start nodes - let’s call it the “frontier”
sort the frontier by g(n)+h(n) values

repeat
  if no nodes on the frontier then terminate with failure
  choose one node from the front of the frontier and remove it
  if the chosen node matches the goal node
    then terminate with success
    else put next nodes (neighbors) and g(n)+h(n) values on frontier
        and sort frontier by g(n)+h(n) values
end repeat

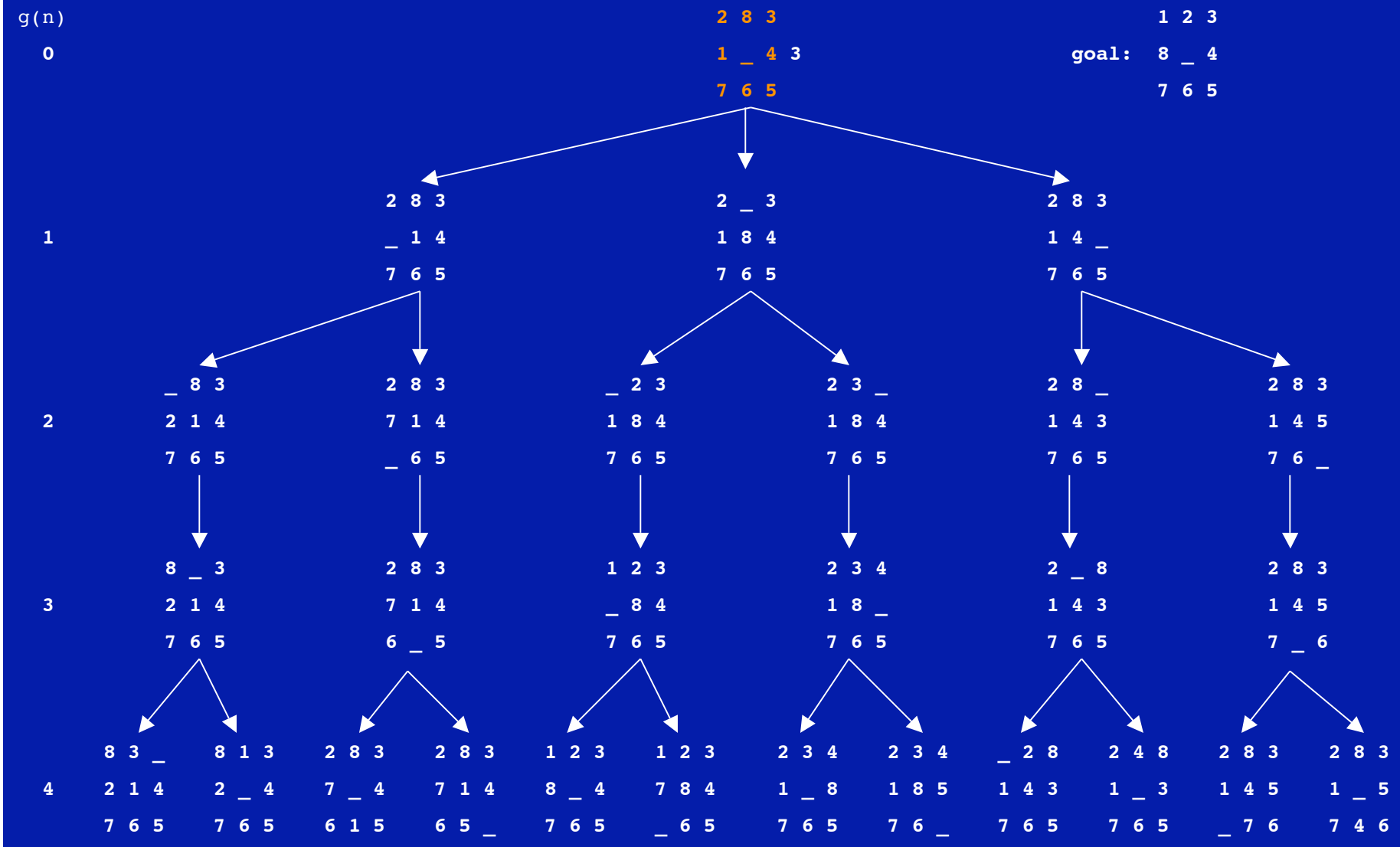
Combining g(n) + h(n)



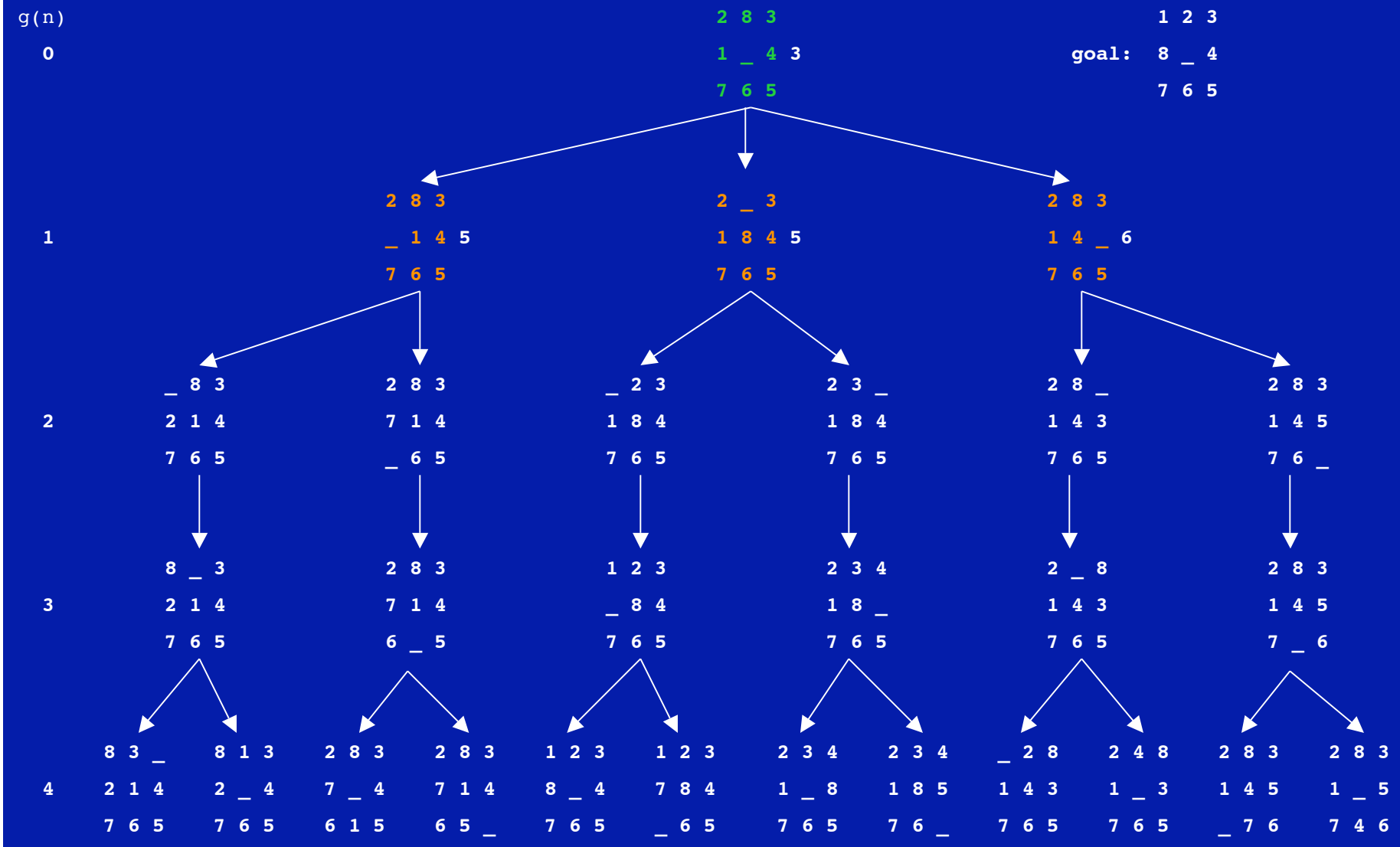
g(n) is actual path length from start node
h(n) is tiles out of place



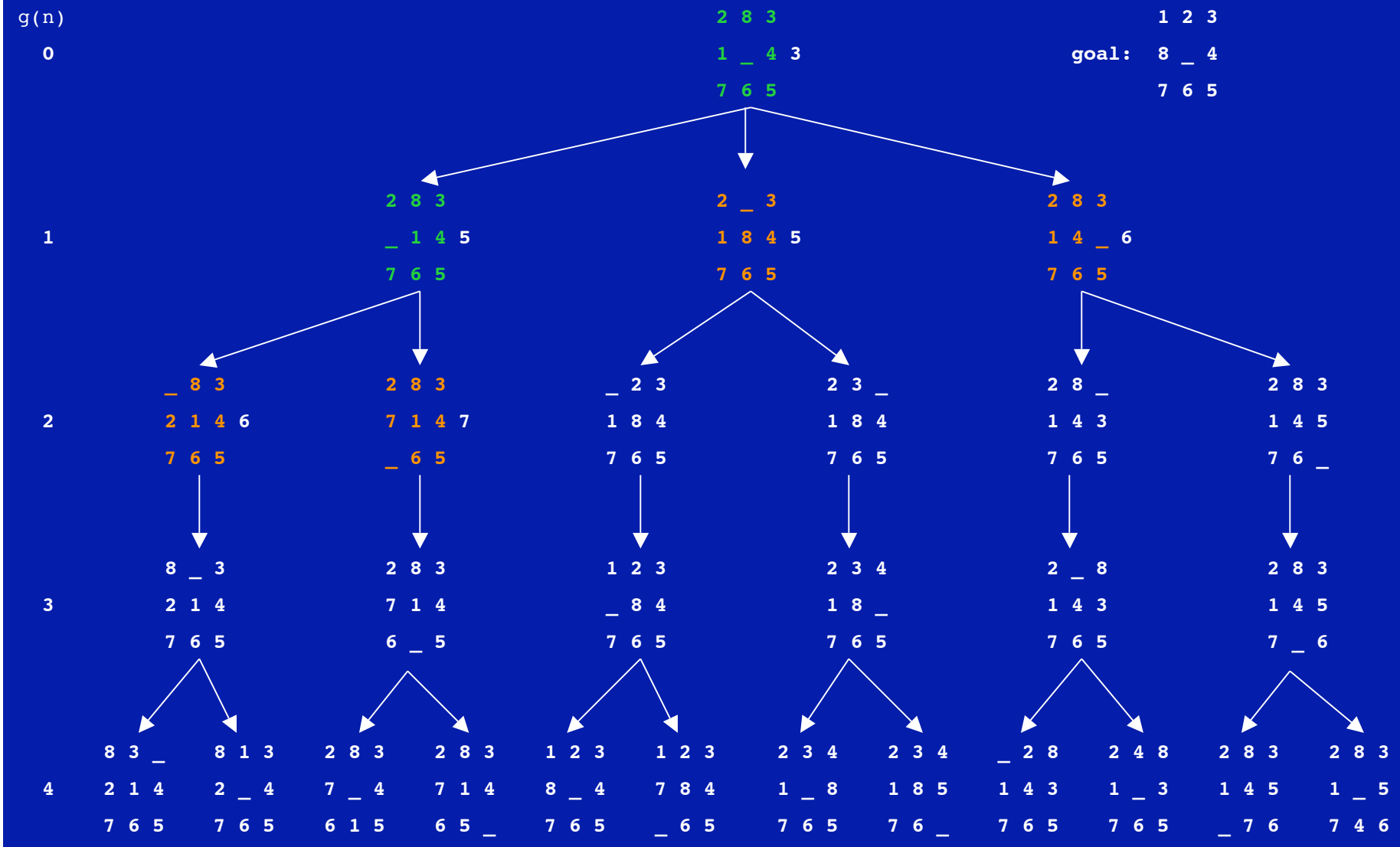
g(n) is actual path length from start node
h(n) is tiles out of place



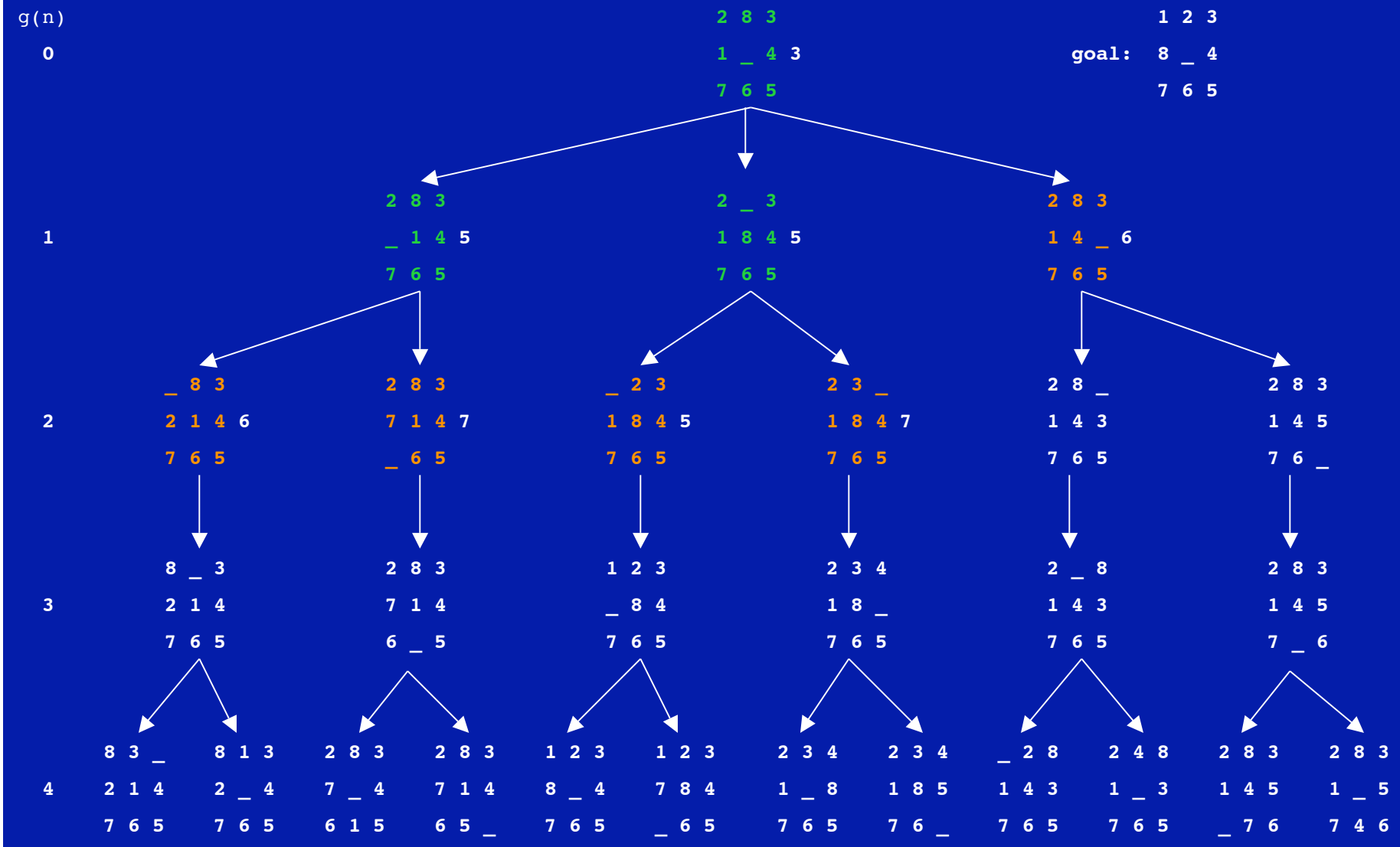
g(n) is actual path length from start node
h(n) is tiles out of place



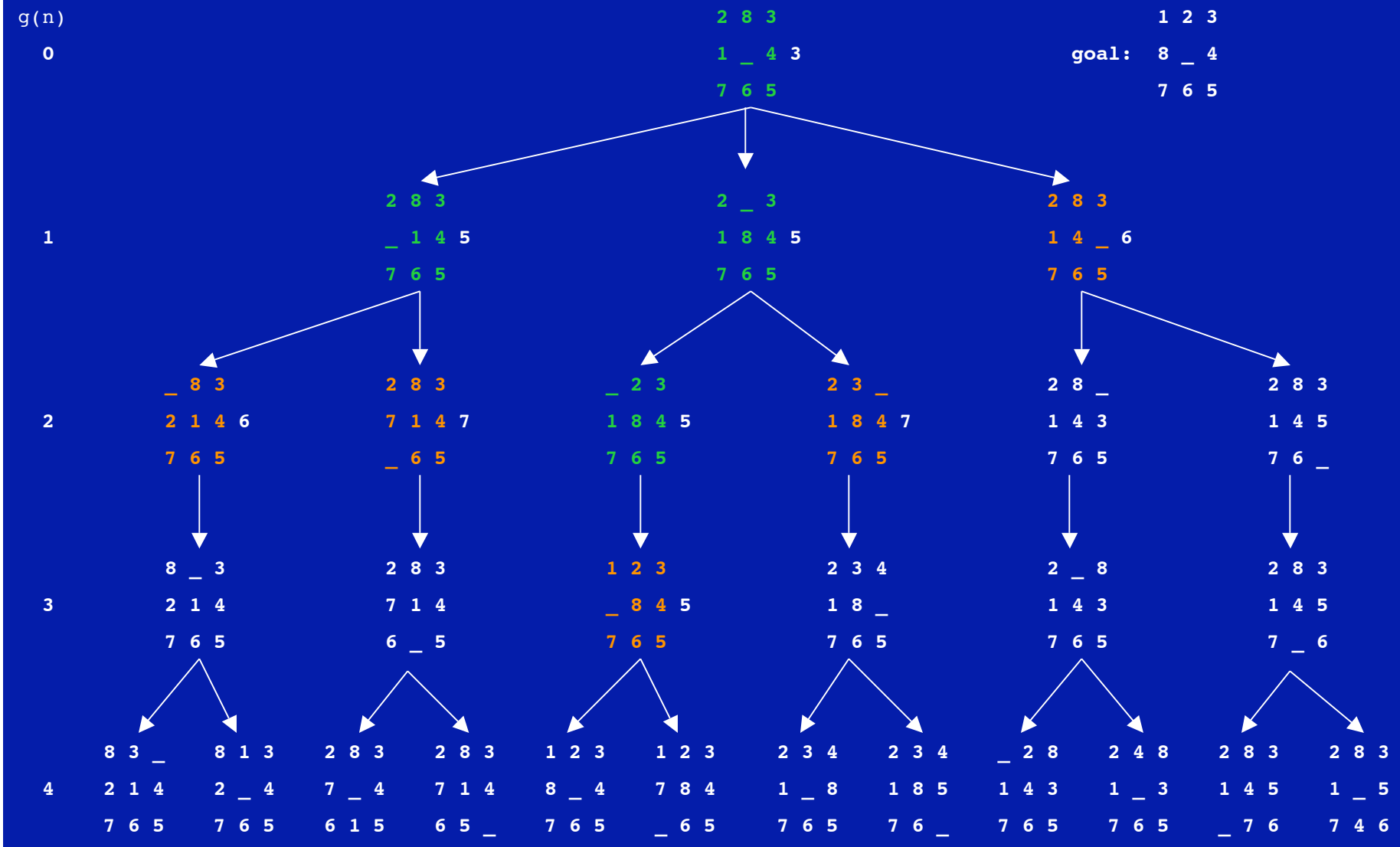
g(n) is actual path length from start node
h(n) is tiles out of place



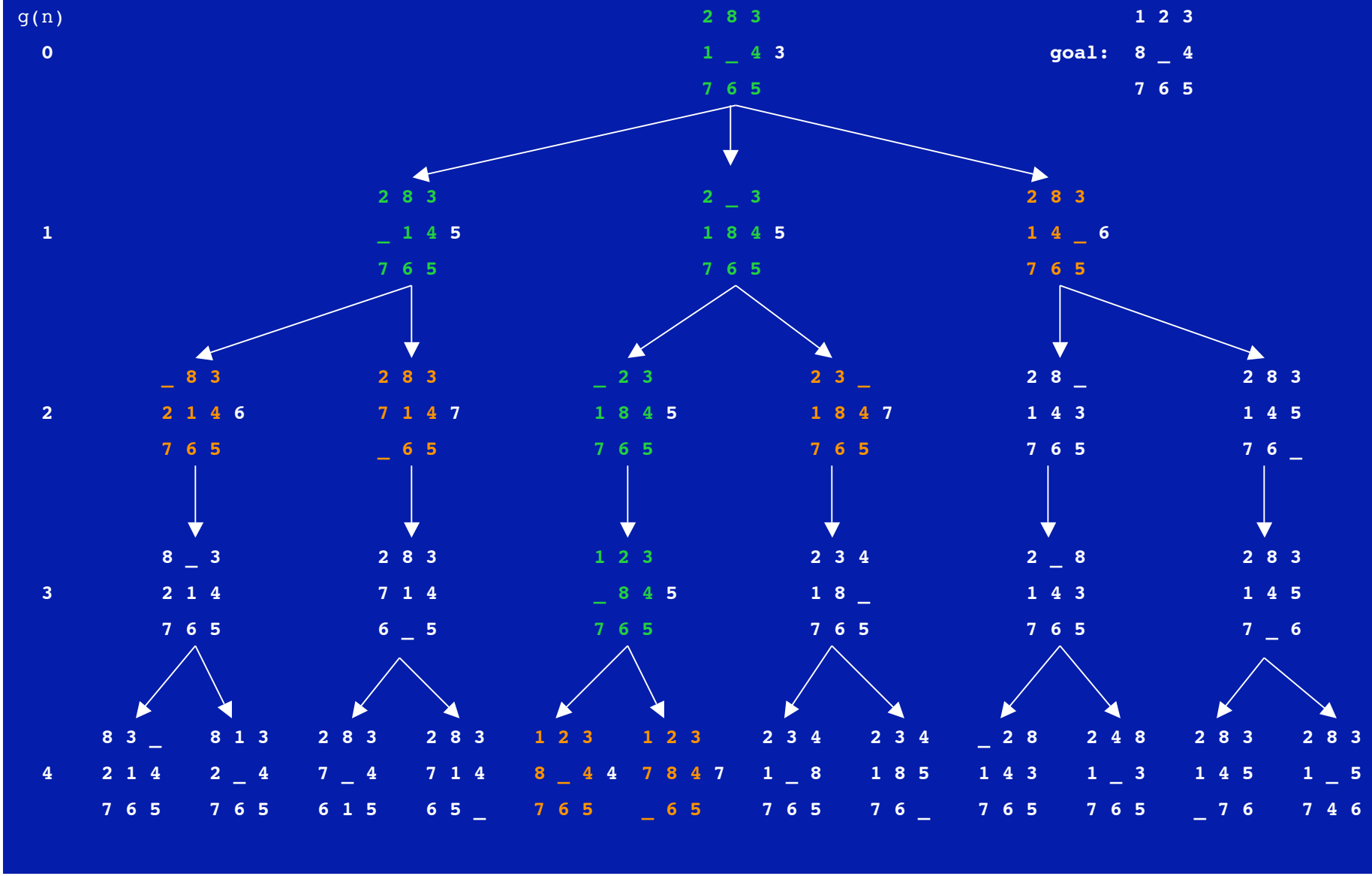
g(n) is actual path length from start node
h(n) is tiles out of place



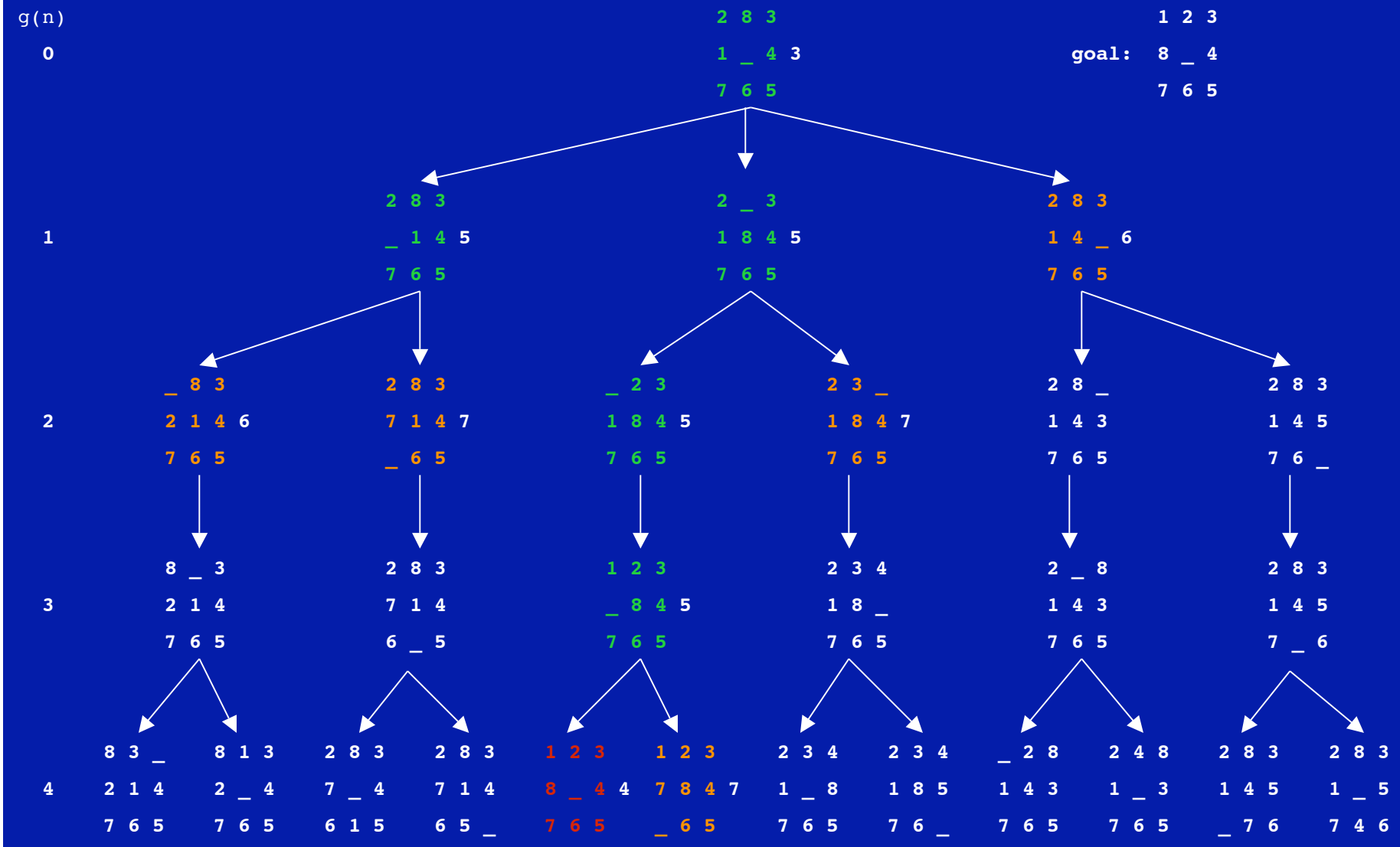
g(n) is actual path length from start node
h(n) is tiles out of place



g(n) is actual path length from start node
h(n) is tiles out of place



g(n) is actual path length from start node
h(n) is tiles out of place



Best-first with g(n) included is called...

Given a set of start nodes, a set of goal nodes,
and a graph (i.e., the nodes and arcs):

apply heuristic g(n)+h(n) to start nodes
make a “list” of the start nodes - let’s call it the “frontier”
sort the frontier by g(n)+h(n) values

repeat
  if no nodes on the frontier then terminate with failure
  choose one node from the front of the frontier and remove it
  if the chosen node matches the goal node
    then terminate with success
    else put next nodes (neighbors) and g(n)+h(n) values on frontier
        and sort frontier by g(n)+h(n) values
end repeat

Search algorithm A

Given a set of start nodes, a set of goal nodes,
and a graph (i.e., the nodes and arcs):

apply heuristic g(n)+h(n) to start nodes
make a “list” of the start nodes - let’s call it the “frontier”
sort the frontier by g(n)+h(n) values

repeat
  if no nodes on the frontier then terminate with failure
  choose one node from the front of the frontier and remove it
  if the chosen node matches the goal node
    then terminate with success
    else put next nodes (neighbors) and g(n)+h(n) values on frontier
        and sort frontier by g(n)+h(n) values
end repeat

Search algorithm A

If the branching factor in the state space is finite

and

If the arc costs are bounded above 0 (i.e., there is some e > 0 such that all arc costs are greater than e)

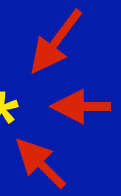
and

h(n) is a lower bound on the actual minimum cost of the shortest path from node n to a goal node

then

we call this search algorithm...

Search algorithm A*



If the branching factor in the state space is finite

and

If the arc costs are bounded above 0 (i.e., there is some e > 0 such that all arc costs are greater than e)

and

h(n) is a lower bound on the actual minimum cost of the shortest path from node n to a goal node

then

we call this search algorithm...

Search algorithm A*

More important than the odd name is the fact that algorithm A* is admissible

That's AI shorthand for "the algorithm will find an optimal path, if one exists, from the start to the goal, and that the first path found from the start to the goal will be optimal, even if the search space is infinite"

Admissibility does not, however, guarantee that an optimal path will be found quickly

Search algorithm A*

Using the “tiles out of place” heuristic for h(n) in the 8-tile puzzle search is an example of an A* algorithm -- “tiles out of place” is an admissible heuristic

The “Manhattan distance” heuristic is also admissible

Back to the question: Would this work?

Yes!

The search strategies that find optimal paths all take g(n) -- the cost of the path from start node to frontier node -- into consideration...

...and best-first search seems like a reasonably sane heuristic search (although it's exponential)...

...what would happen if we rewrite best-first search to include g(n) as well as h(n)?

We get a heuristic search strategy that's guaranteed to find an optimal path to the goal if one exists!

Comparing search strategies

strategy	selection	halts?	space	optimal path
depth-first	last node added	no	linear	no
breadth-first	first node added	yes	exponential	yes[1]
lowest-cost first	minimal g(n)	yes	exponential	yes
best-first	globally minimal h(n)	no	exponential	no
heuristic depth-first	locally minimal h(n)	no	linear	no
A*	minimal g(n)+h(n)	yes	exponential	yes

1. assuming all arcs have the same cost

Refinements to search strategies

Cycle checking

- you don’t want your search algorithm looping through cycles in the graph
- if your search algorithm puts paths from start node to frontier on the frontier list, it can compare new neighbor nodes to the nodes on the path before adding them to eliminate the possibility of cycles
- A* doesn’t have to “worry” about cycles...why?

Refinements to search strategies

Iterative deepening

- Breadth-first search is optimal but hogs space;
  depth-first search conserves space but may not halt
- Iterative deepening combines the best of both:
  employ depth-first search to some depth bound;
  if the goal is not found, discard what's been done,
  increase the depth bound and do depth-first search again
- Computational overhead will be increased, but it may
  not be as bad as it might seem at first (read your book)

Refinements to search strategies

Direction of search

- It's not necessary to begin searching at the start node; search can begin with a goal node
- In general, if goal nodes are explicitly defined, and the branching factor (number of arcs) leading into nodes tends to be smaller than the branching factor leading away from nodes, the problem might be a good candidate for backward search (8-tile puzzle)
- However, if goal nodes are determined implicitly by a predicate (leading to many goal nodes), forward search is probably better (chess...where would you start?)

Refinements to search strategies

There are other refinements in chapter 4.6

- you should read them
- but you can skip chapter 4.7 for now...we may get back to it later in the term