CPSC 322
Introduction to Artificial Intelligence

October 18, 2004

Things...

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

Homework 3 coming this week

Term project coming soon

Comparing blind search strategies

All will find a solution in a finite space if a solution exists

Depth-first can get trapped in infinite recursion in an infinite space (except in CILOG)

Breadth-first and lowest-cost will find a solution even in an infinite space, even if one exists

Breadth-first will find path to goal with fewest arcs

Lowest-cost-first will find lowest cost path (of course) when arcs have different costs

Breadth-first is just depth-first but adding to frontier differently

Lowest-cost-first is just breadth-first with more info and sorting

Comparing blind search strategies

Breadth-first and lowest-cost-first seem wonderful, but they're gigantic space hogs in terms of how big frontier can be

There's a whole lot more of this good vs. bad stuff in your textbook (chapter 4.4)...read it

Ultimately, all three are expensive approaches to search

Reviewing some numbers

Is graph search feasible on really big problems?

8-tile puzzle: 9! nodes = 362,880
4.2 days if enter 1 every second
solved very quickly

15-tile puzzle: 15! nodes = 1,307,674,368,000
41,466 years...
solved in 20 min @ 10^9 nodes/sec

chess: 10^{120} nodes = a lot!
roughly 10^{79} atoms in universe
you do the math
still takes a long, long, long time

Two obstacles to overcome

For really interesting (i.e., big) problems, we need to find ways to...

1. reduce the amount of time spent searching

2. reduce the amount of time spent pre-building the graph

By the way, those reductions have to be significant (i.e., big)

Can we make search less expensive?

Usually we don't have time to spare. We want to find the path to goal with as little effort as possible.

We can use problem-specific knowledge to tell us how to choose the next node for exploration. This knowledge provides an estimate of the nearness of a given node to the goal node - sometimes called the "goodness" of a node.

Can we make search less expensive?

This knowledge about the problem domain used to make “educated guesses” about which node to choose next for further exploration is called heuristic knowledge.

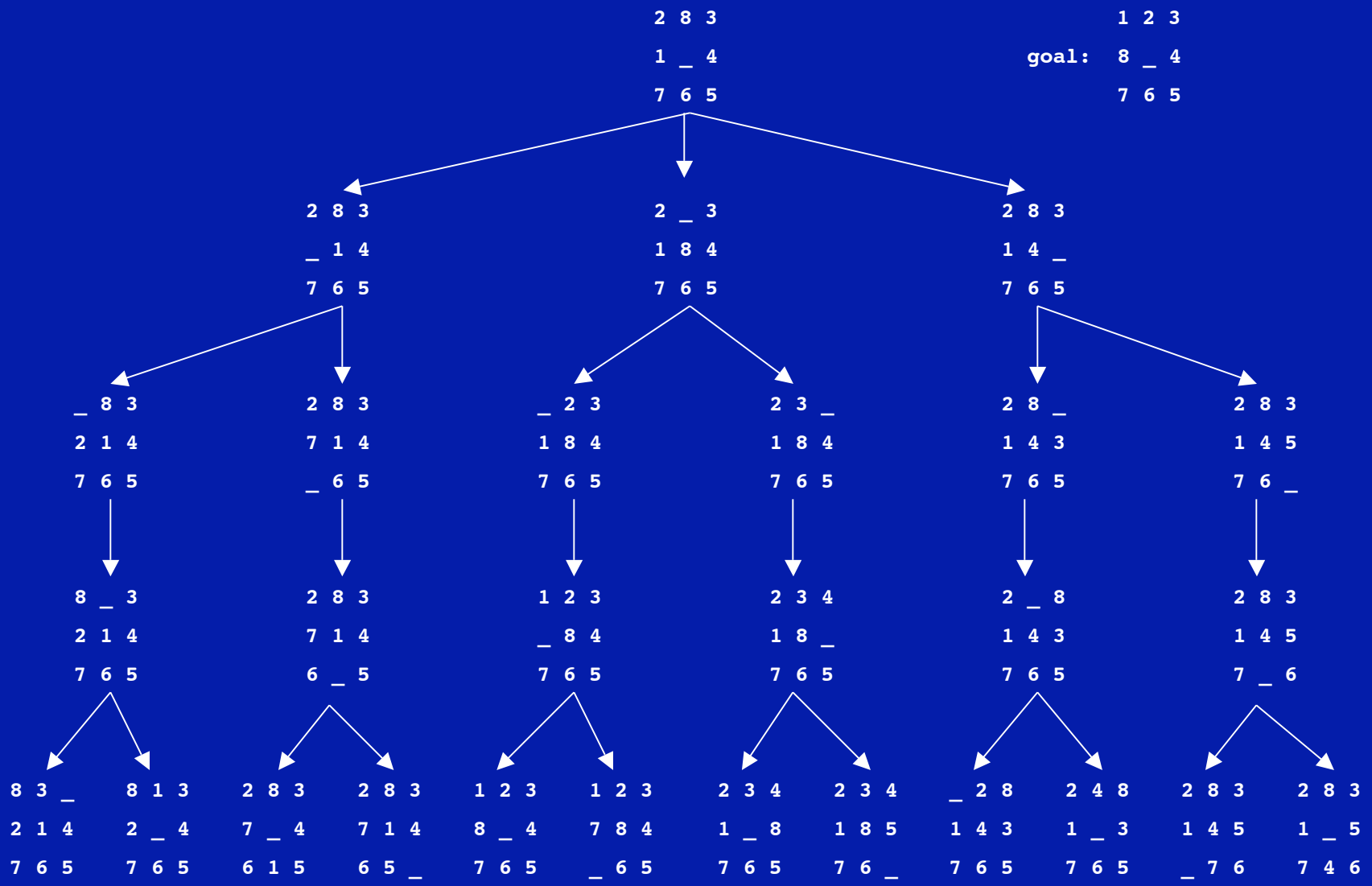
Good heuristic knowledge reduces the search significantly in many cases, but isn’t necessarily guaranteed to do the job in all cases.

Terminology break

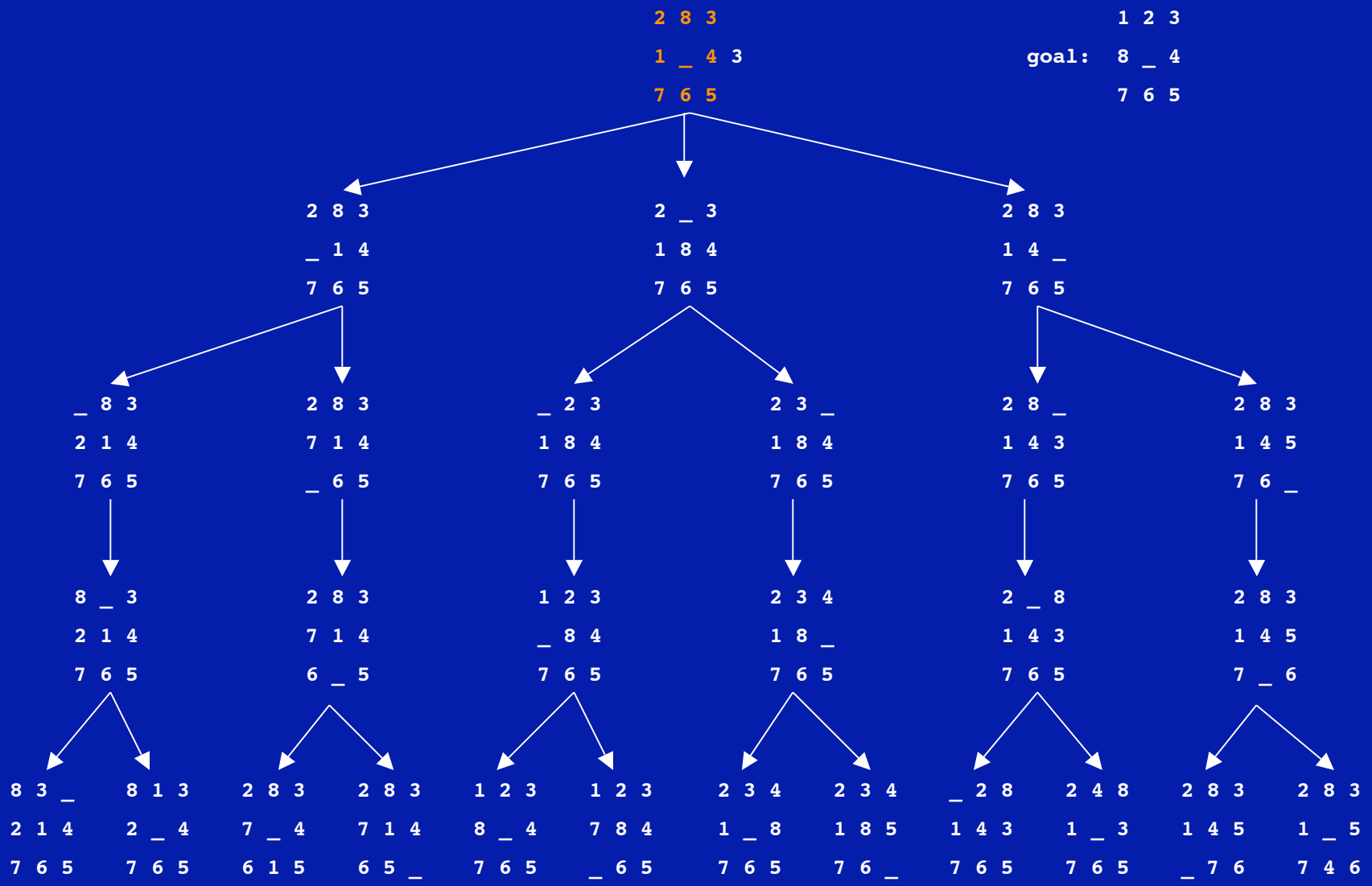
The generic name of the function that returns an estimate of the distance from some frontier node n to the goal node is h(n).

The generic name of the function that returns the actual distance from the start node to some frontier node n is g(n).

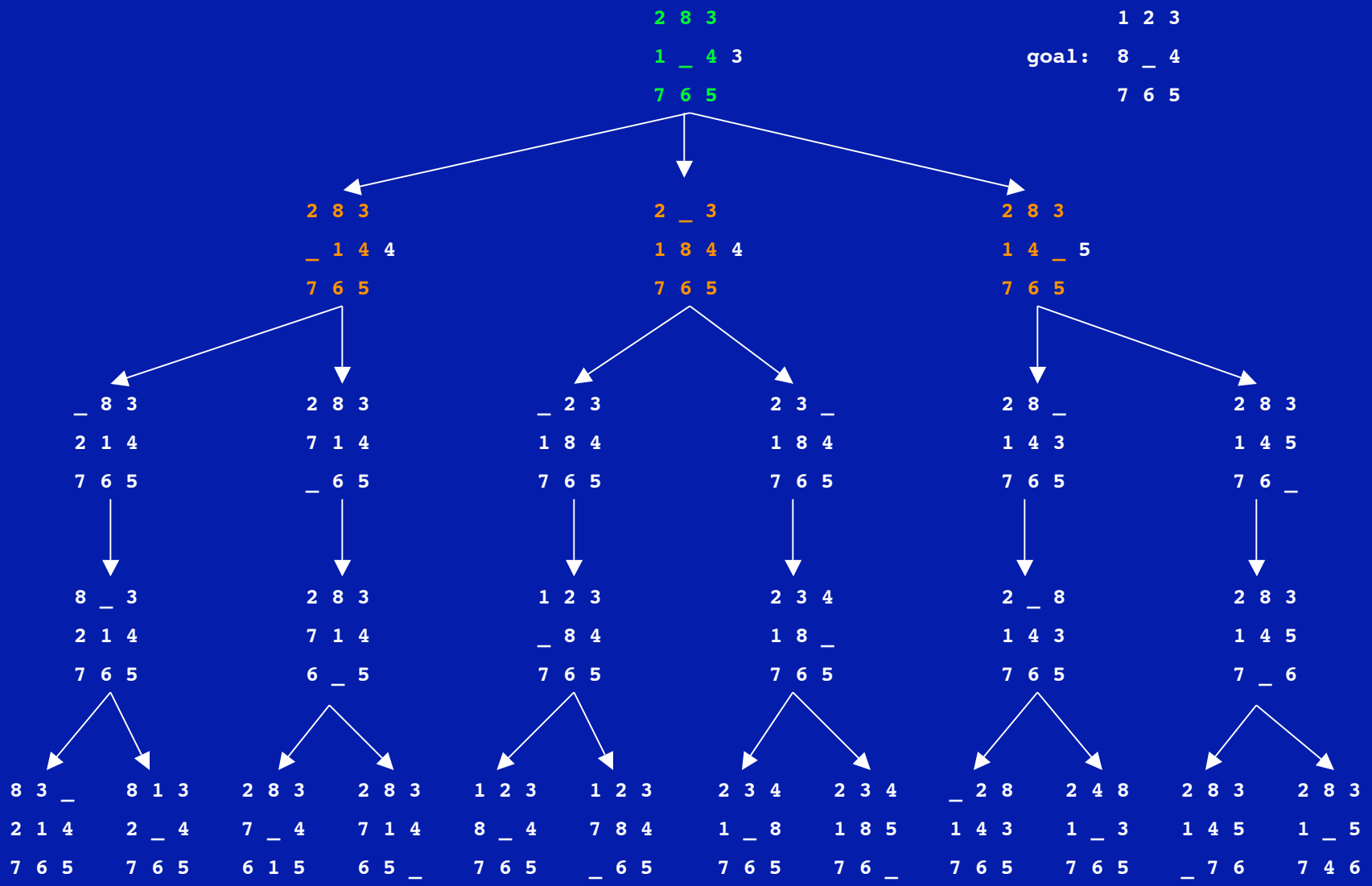
What's a good heuristic here?



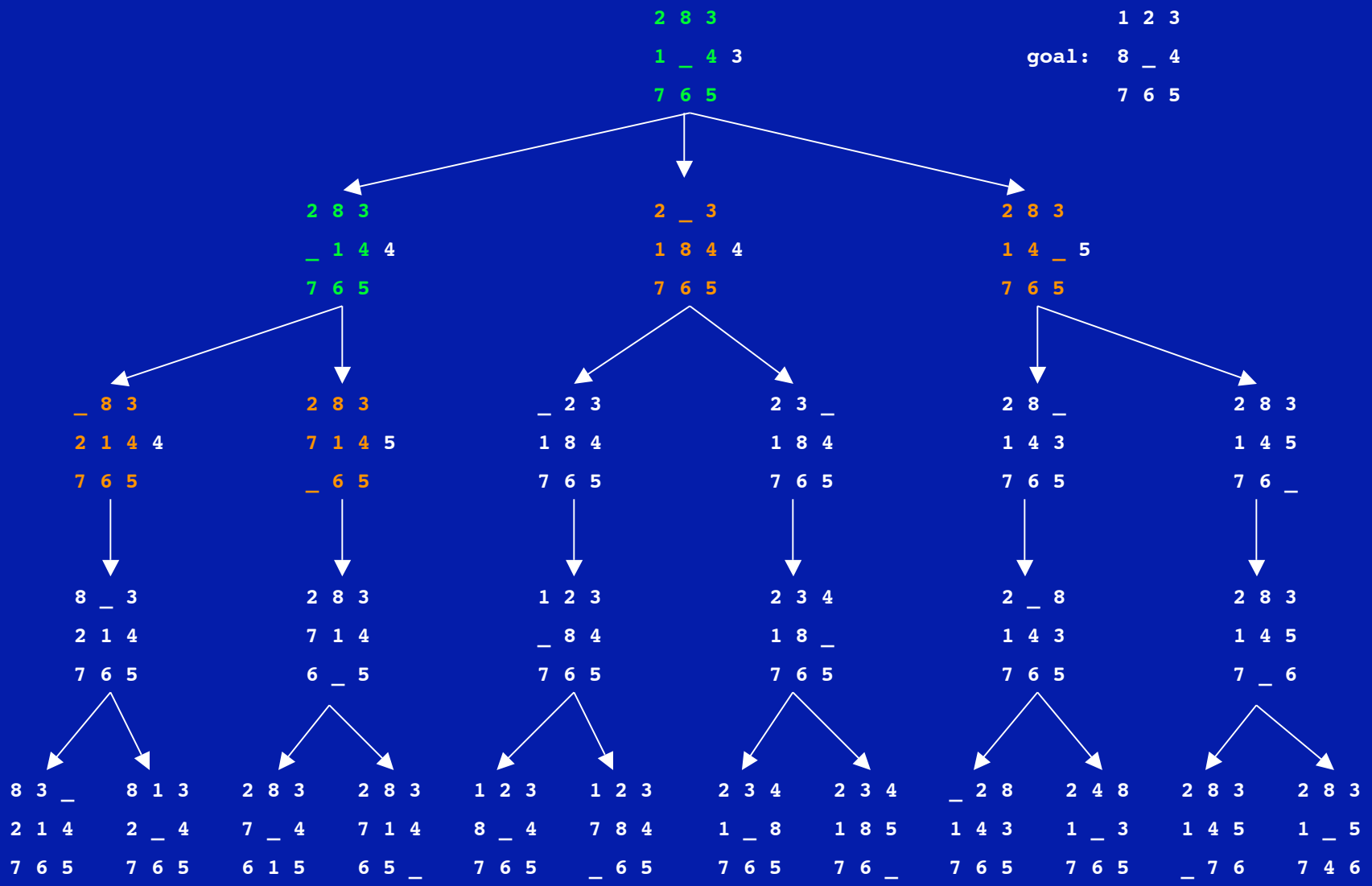
Heuristic best-first search - tiles out of place



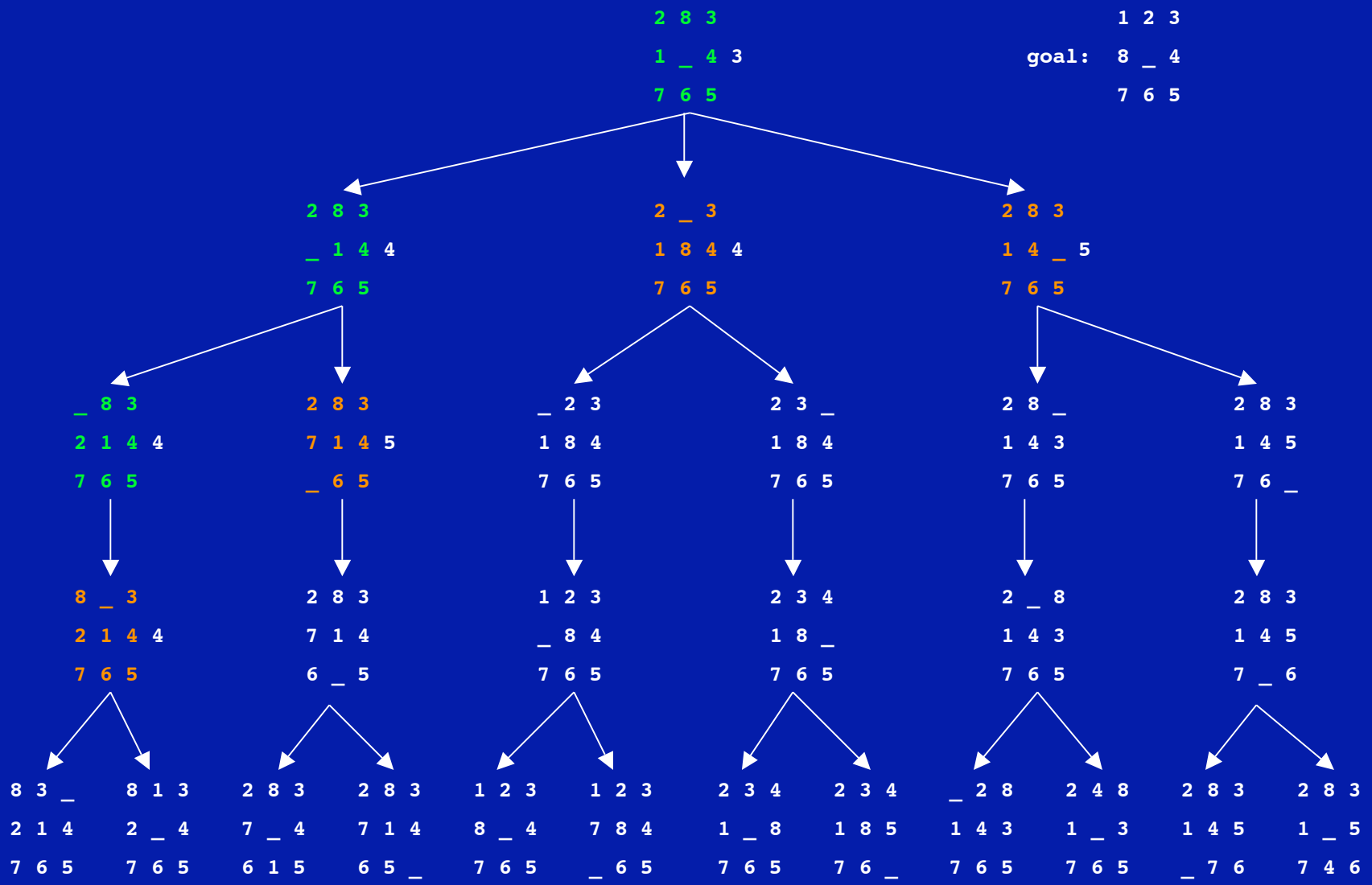
Heuristic best-first search - tiles out of place



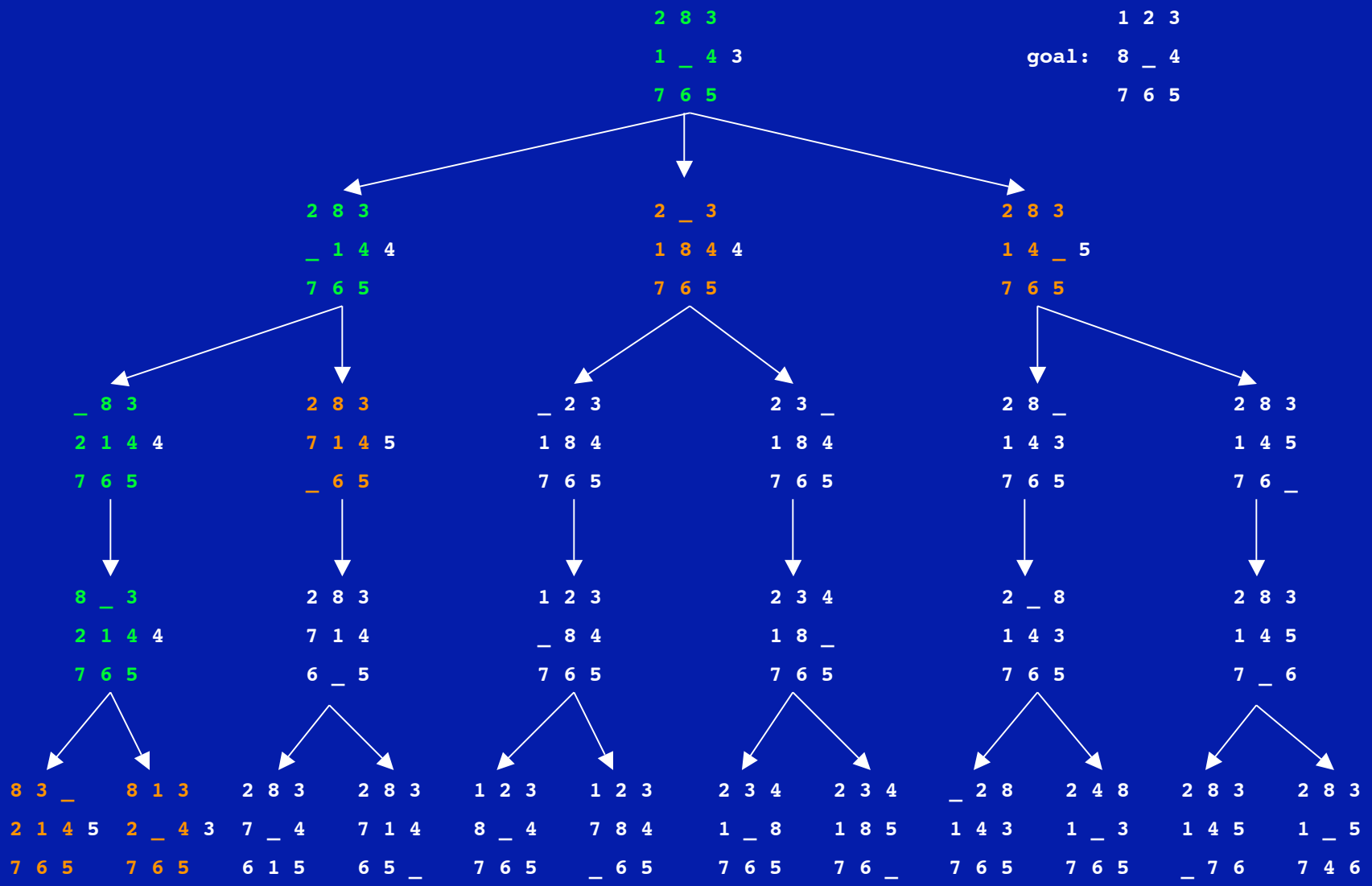
Heuristic best-first search - tiles out of place



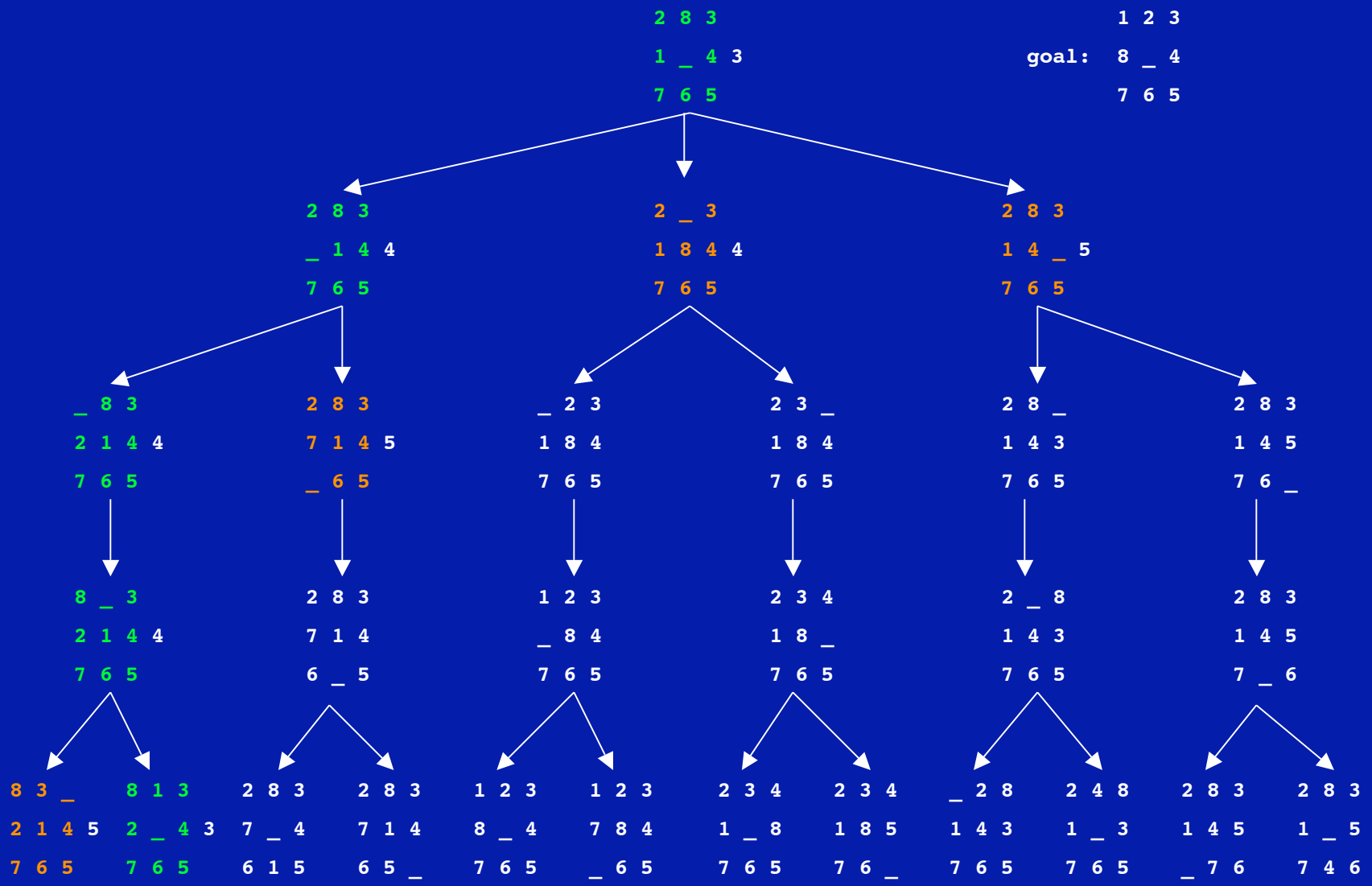
Heuristic best-first search - tiles out of place



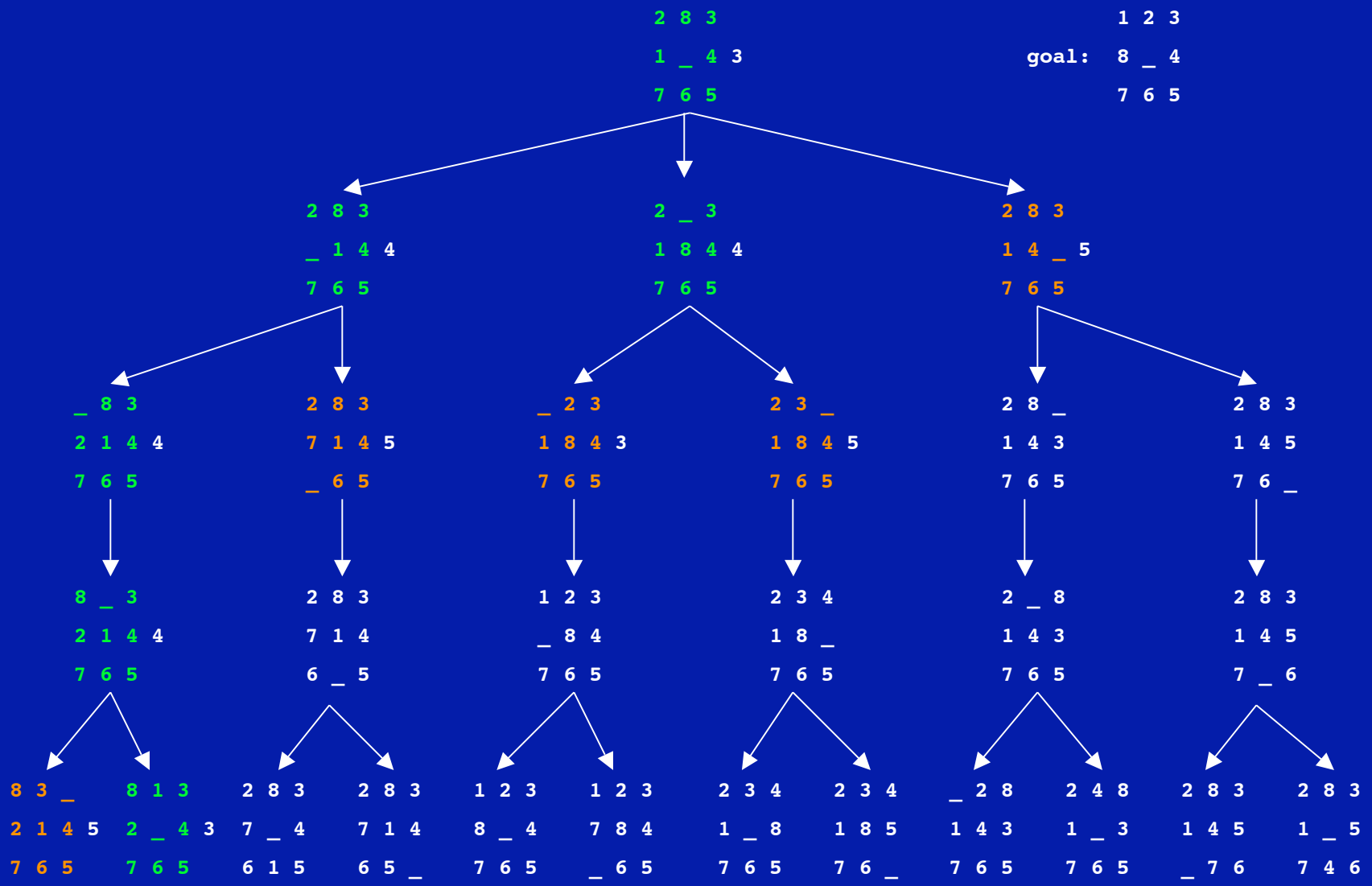
Heuristic best-first search - tiles out of place



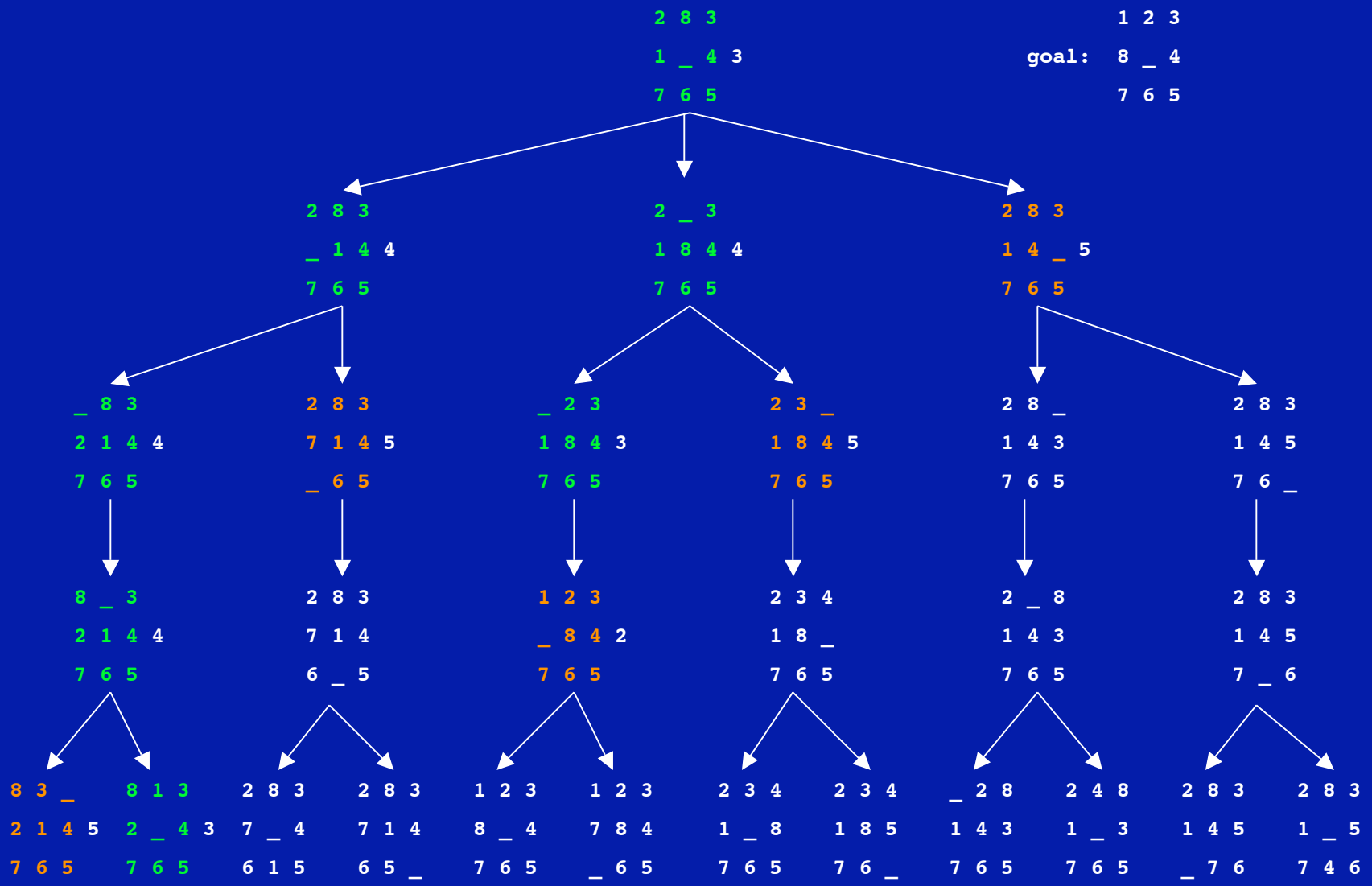
Heuristic best-first search - tiles out of place



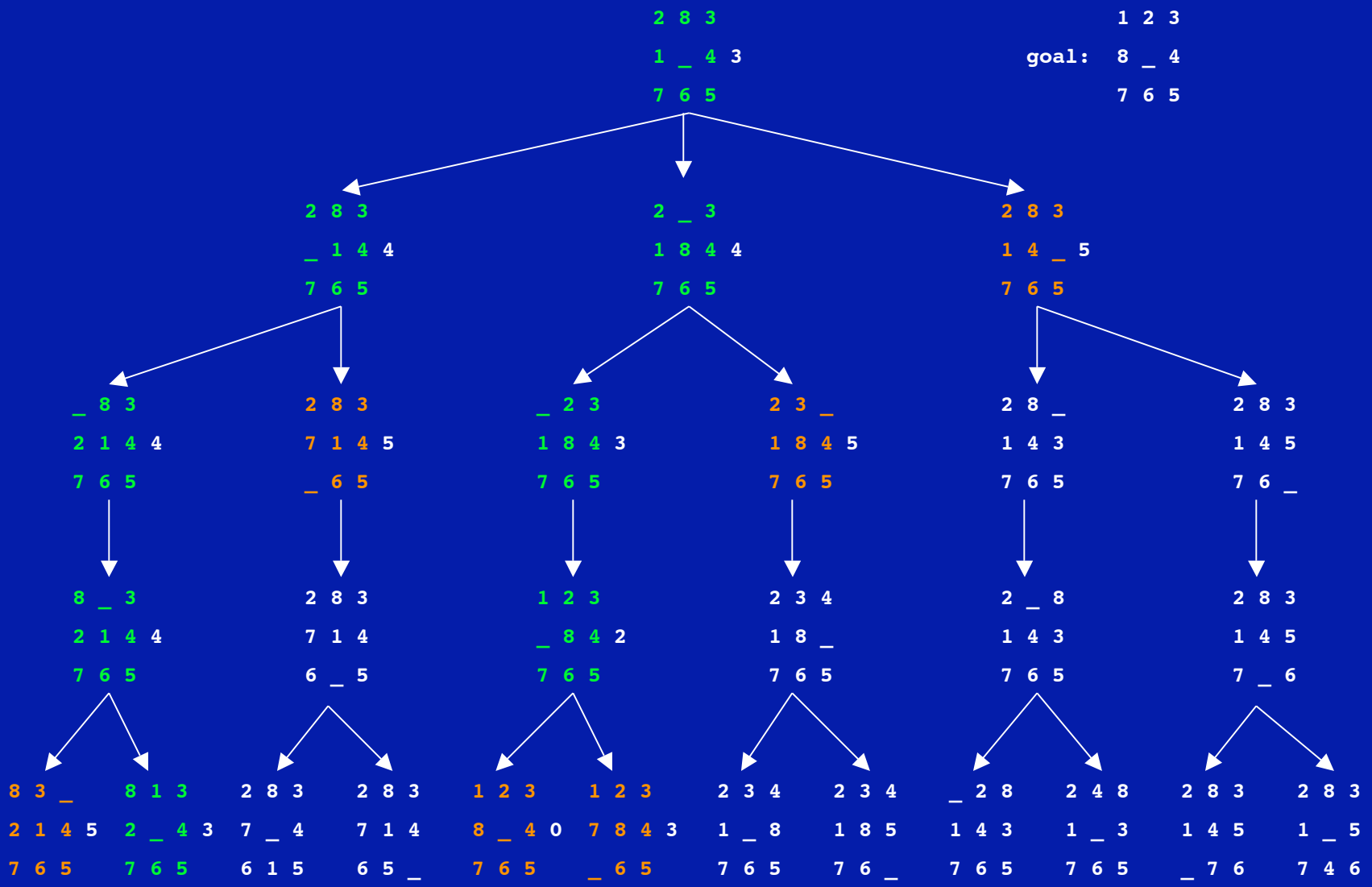
Heuristic best-first search - tiles out of place



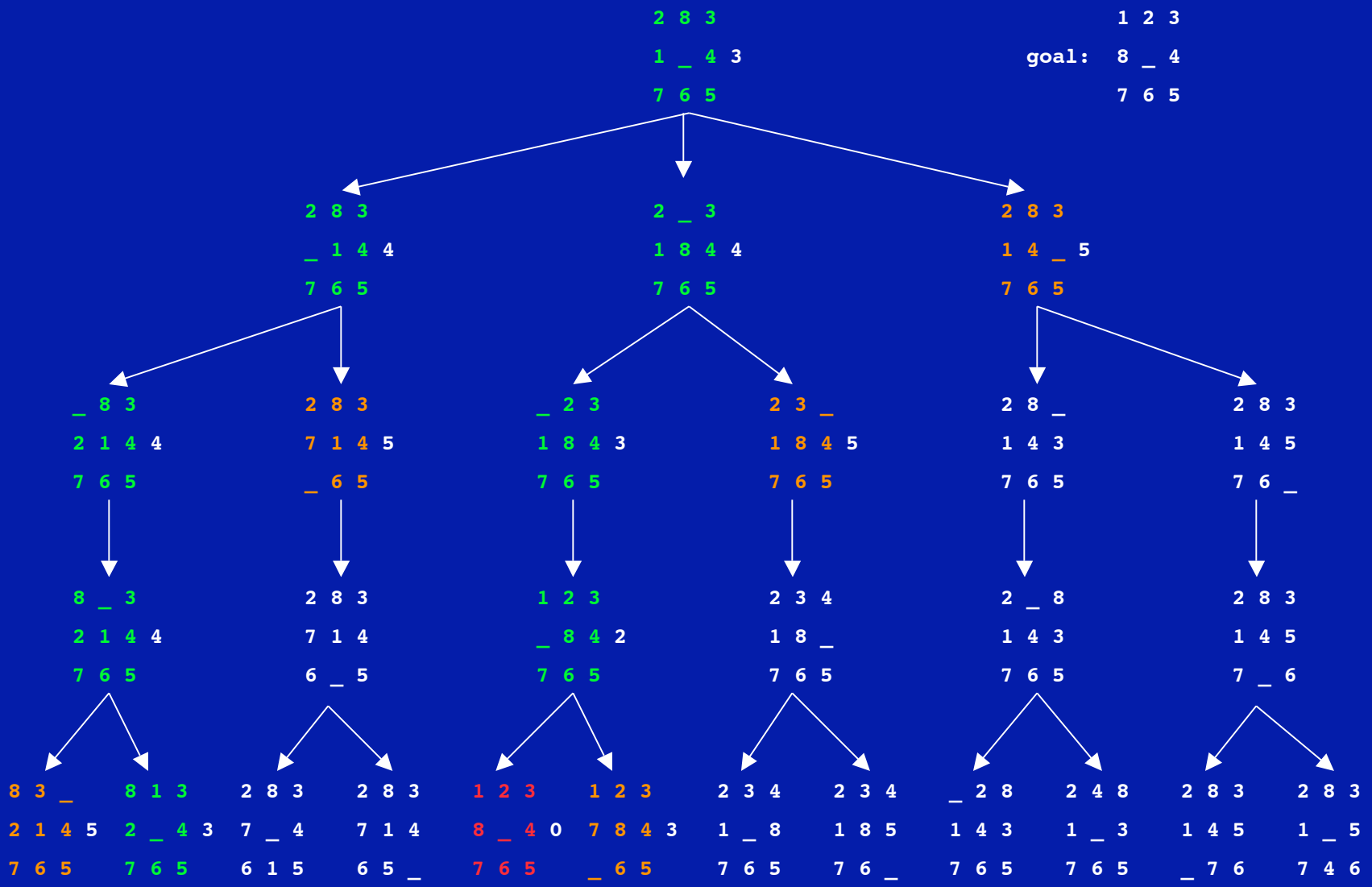
Heuristic best-first search - tiles out of place



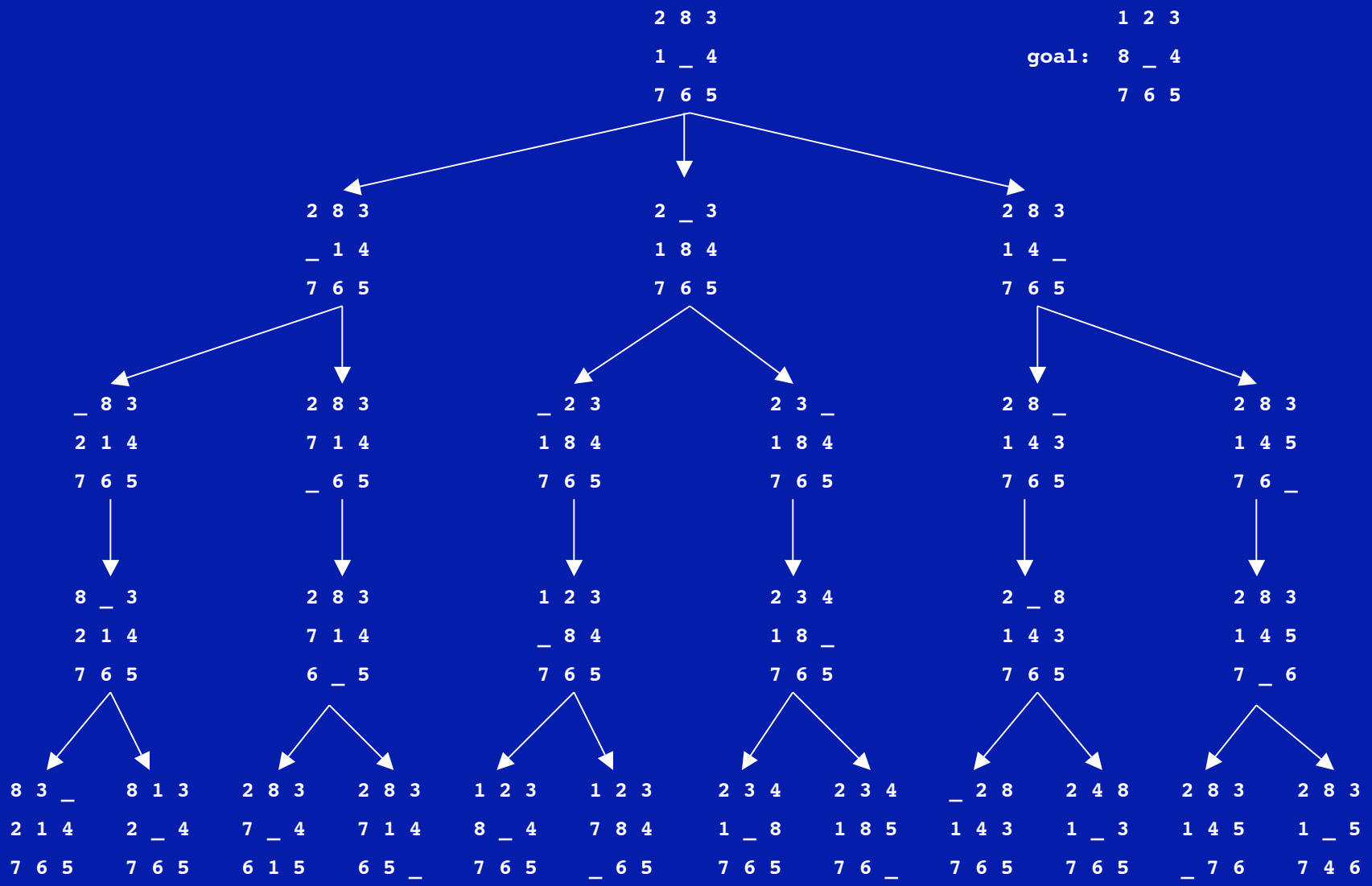
Heuristic best-first search - tiles out of place



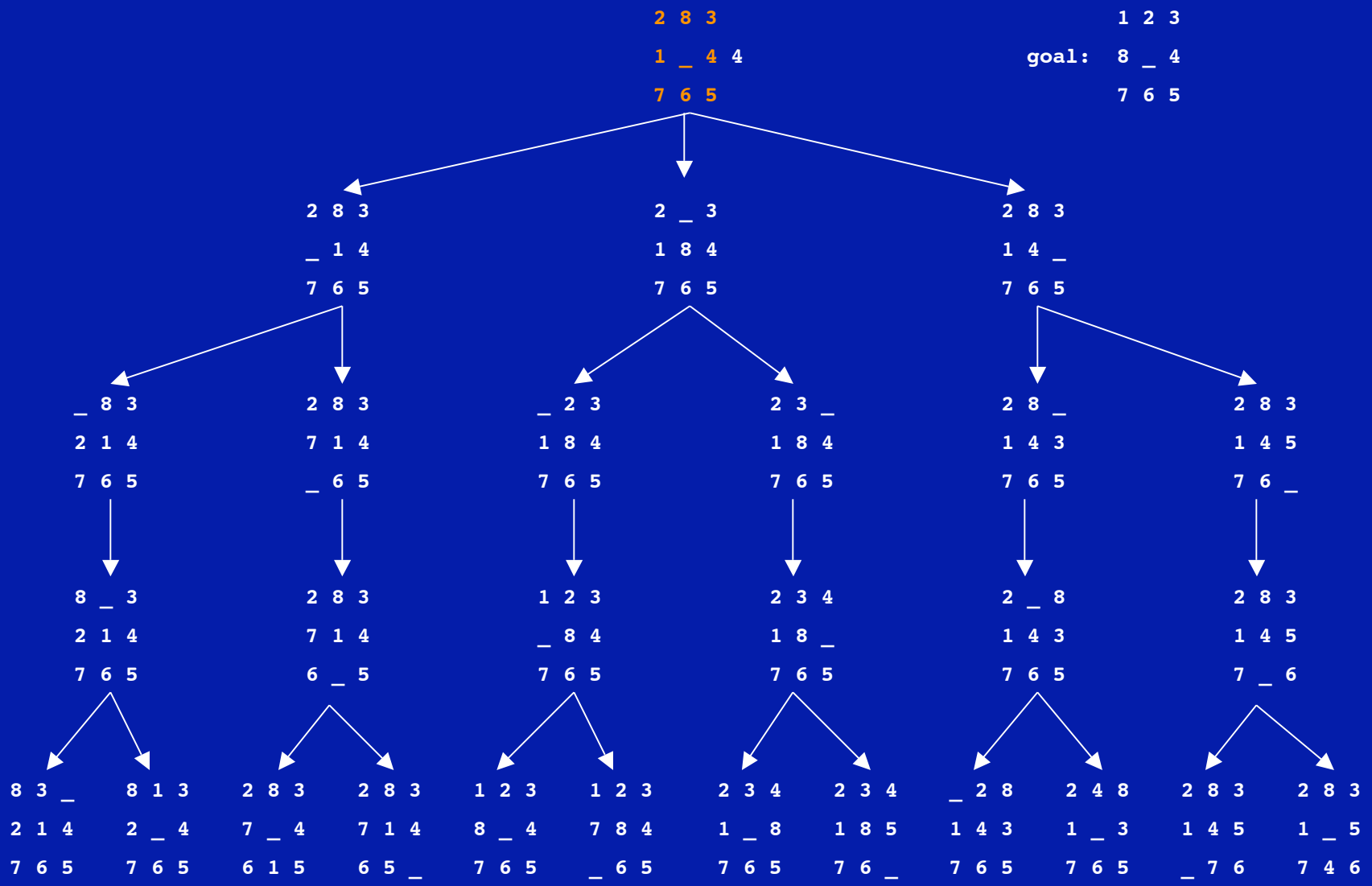
Heuristic best-first search - tiles out of place



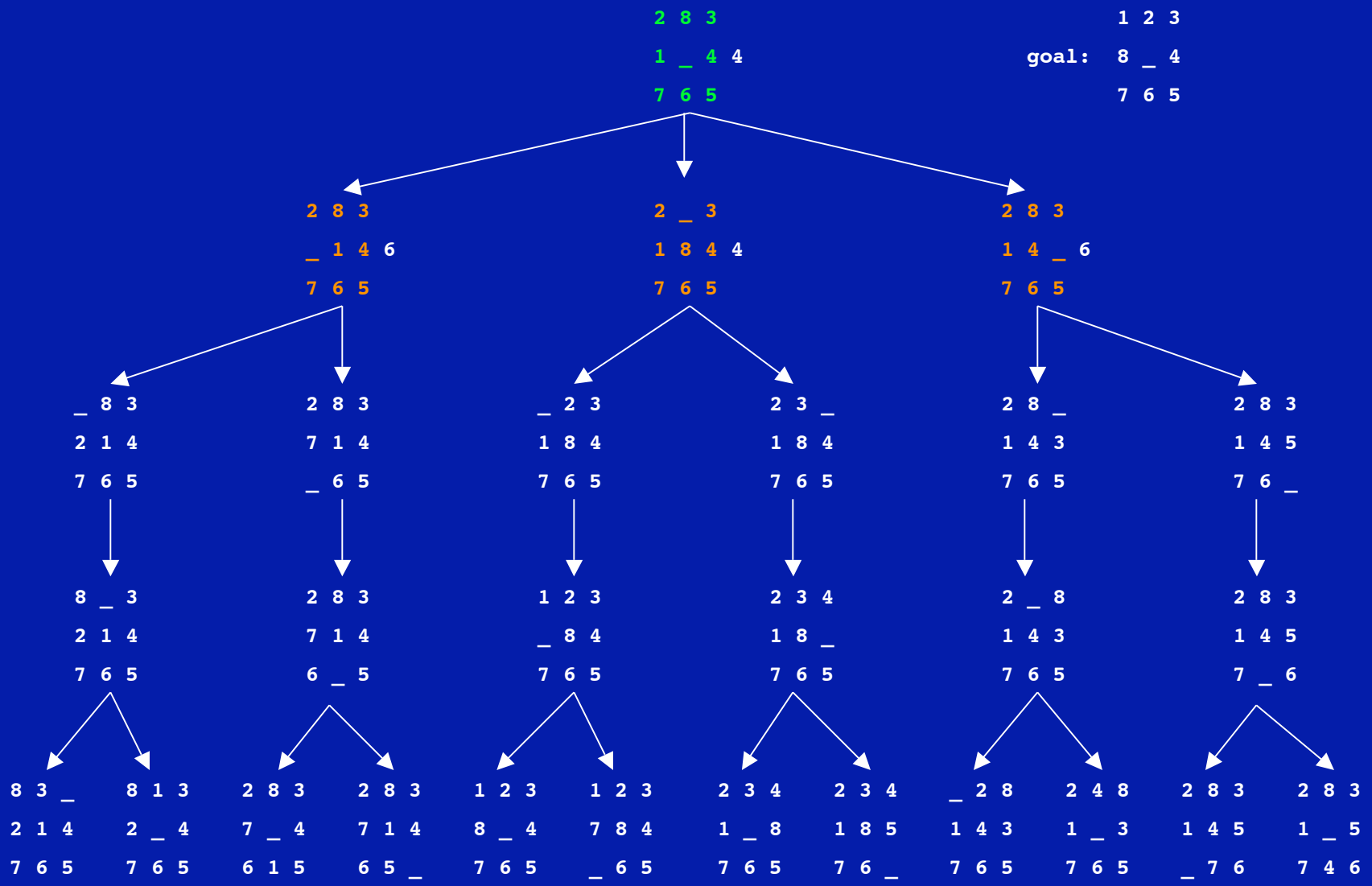
Is there a better heuristic?



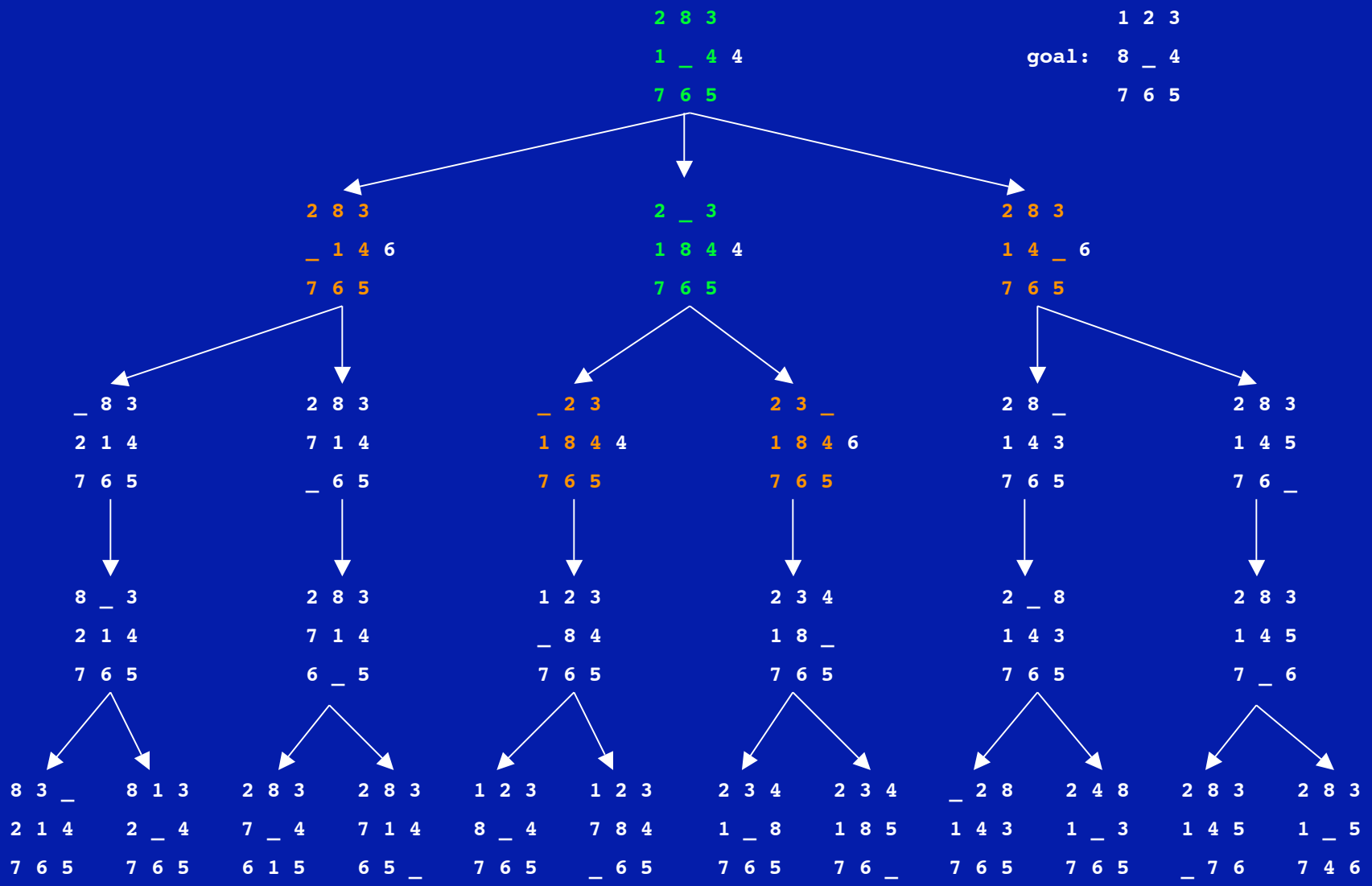
Heuristic best-first search - Manhattan distance



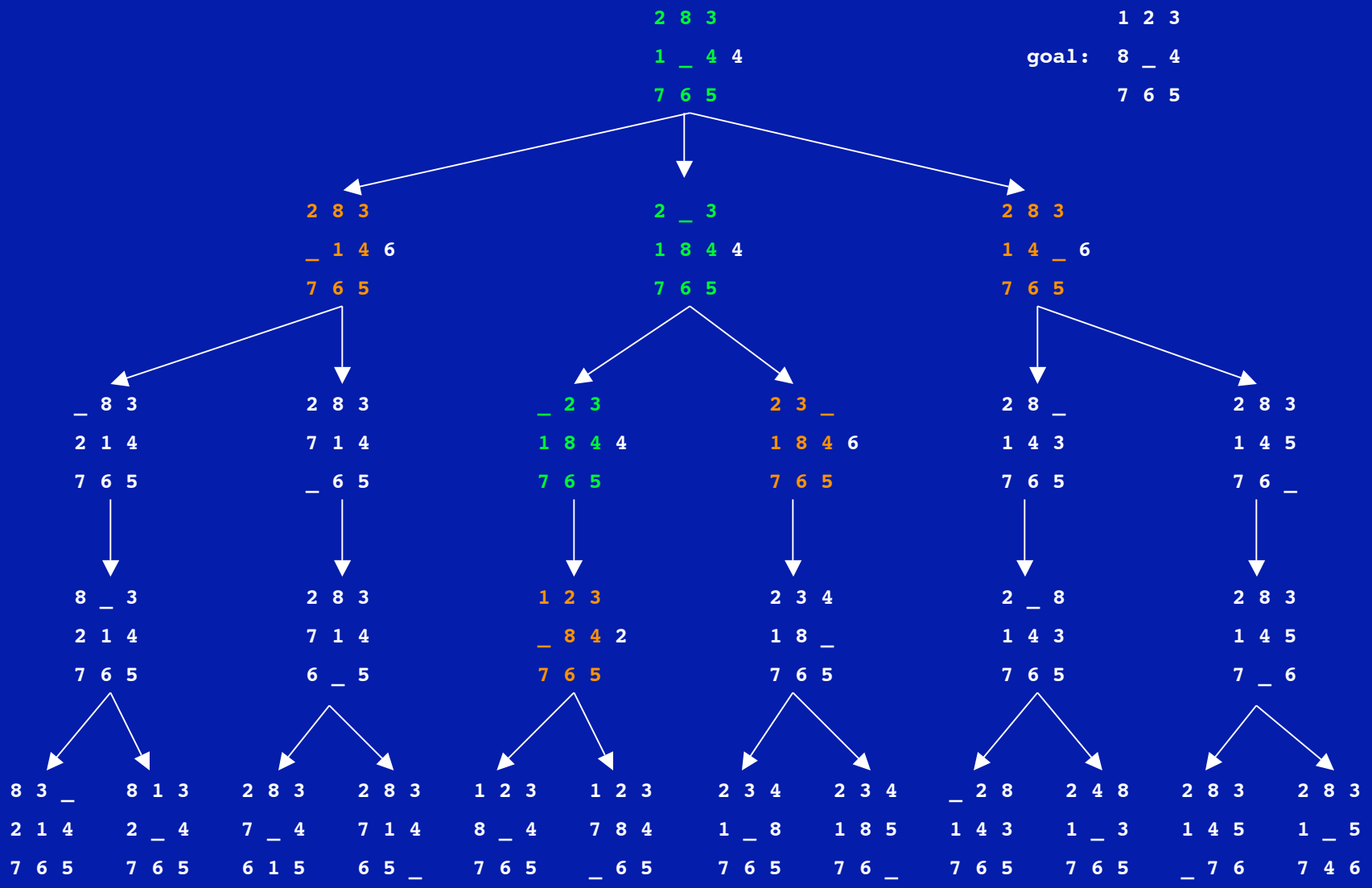
Heuristic best-first search - Manhattan distance



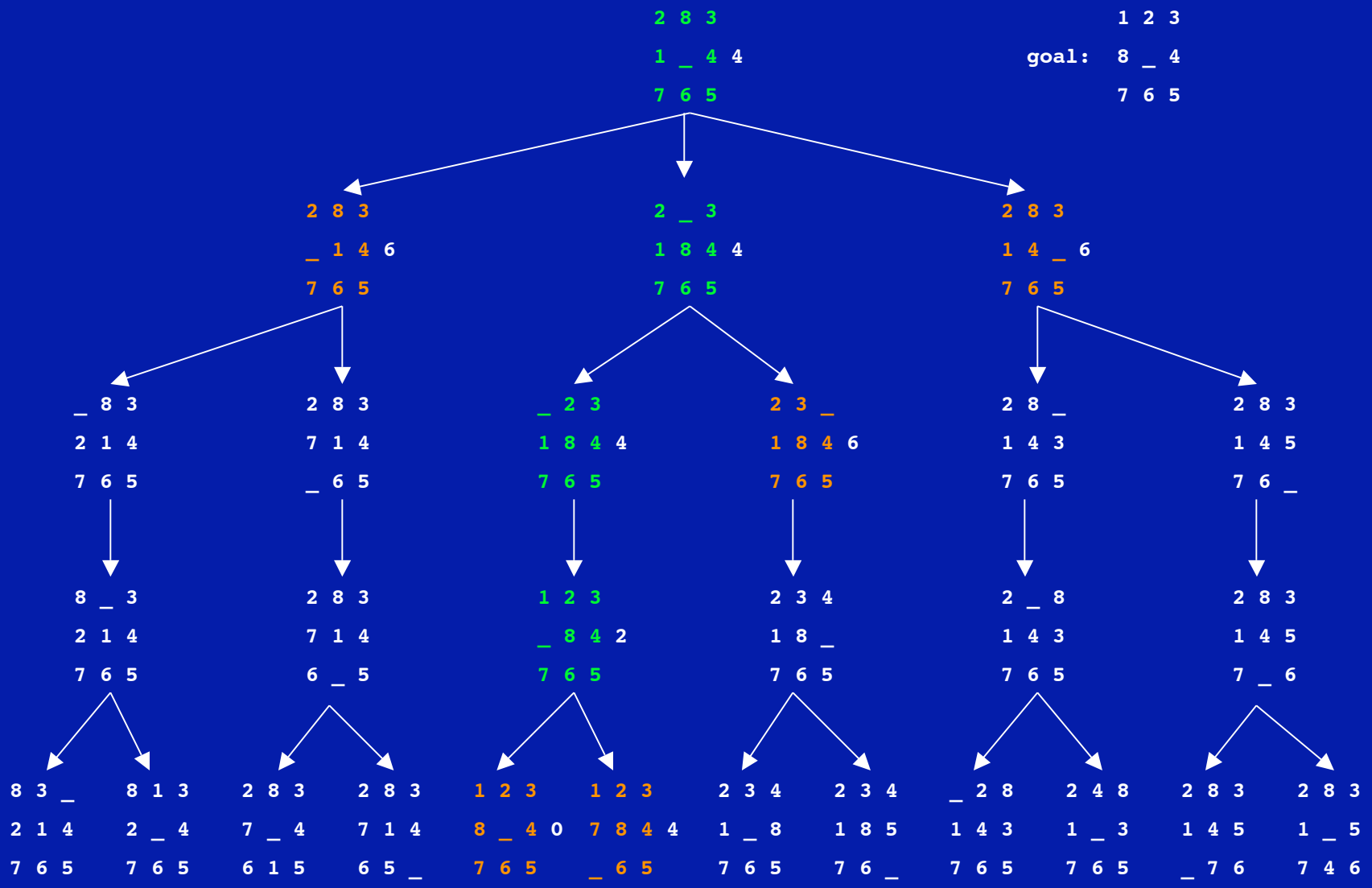
Heuristic best-first search - Manhattan distance



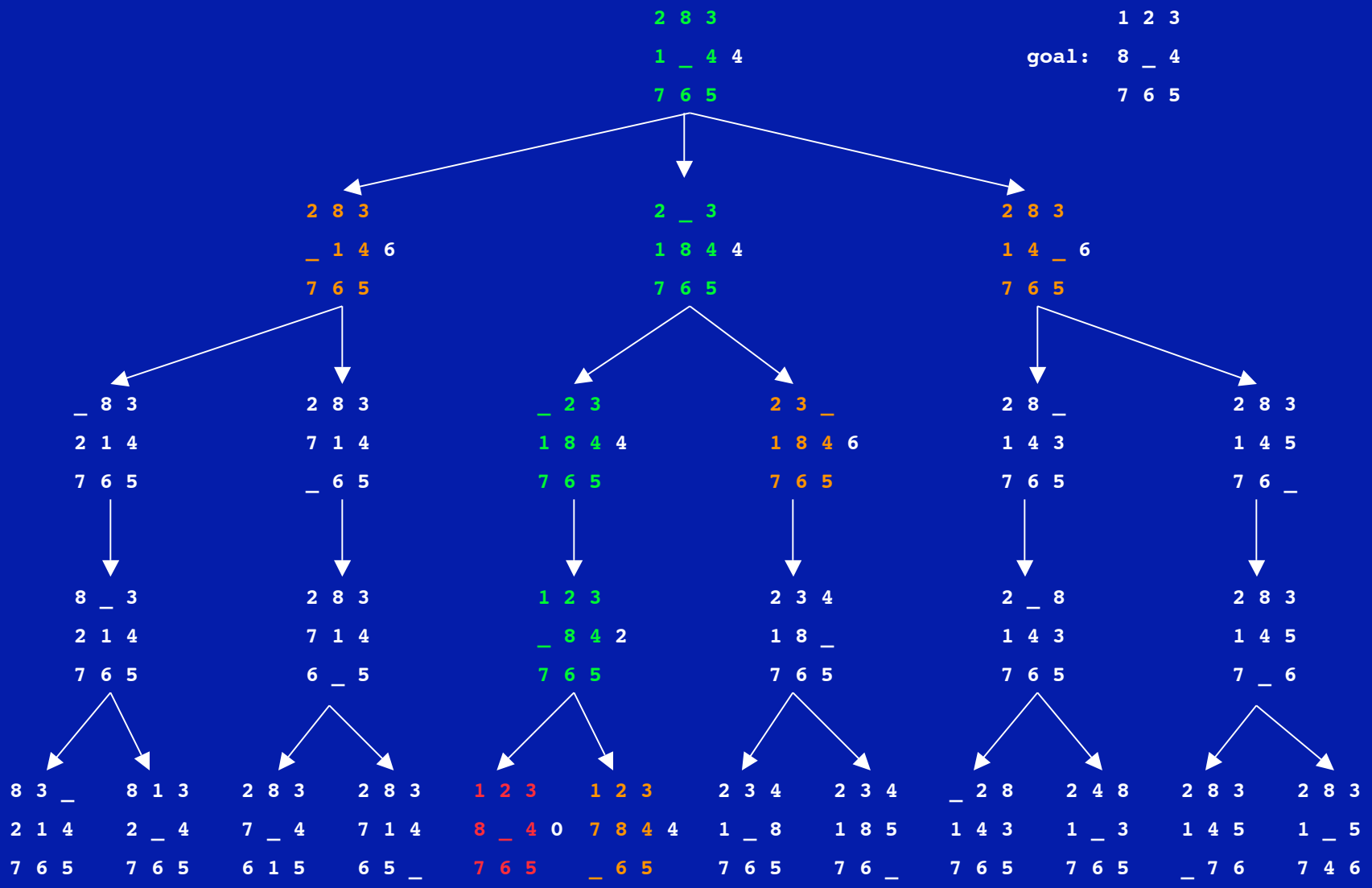
Heuristic best-first search - Manhattan distance



Heuristic best-first search - Manhattan distance



Heuristic best-first search - Manhattan distance



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 algorithm

Heuristic best-first search pursues multiple paths, expanding the node that seems to be nearest to the goal at any given time

Best-first search is like breadth-first search in that it's a gigantic space hog (something like $O(b^n)$ where b is the branching factor in the graph and n is the length of the path)

Not guaranteed to find a solution even if one exists.

Not guaranteed to find the shortest path first.

More heuristic search algorithms

There are others, and we'll get to them real soon,
but first let's address the second problem from earlier...

Two obstacles to overcome

For really interesting (i.e., big) problems, we need to find ways to...

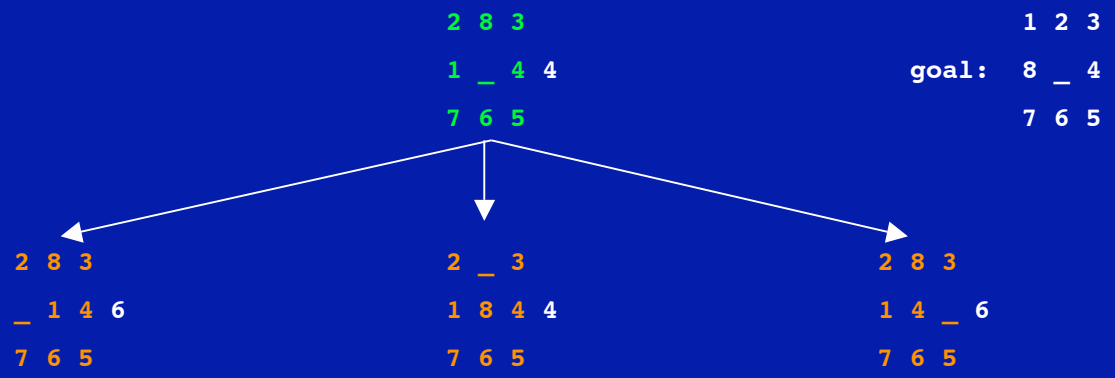
1. reduce the amount of time spent searching

2. reduce the amount of time spent pre-building the graph

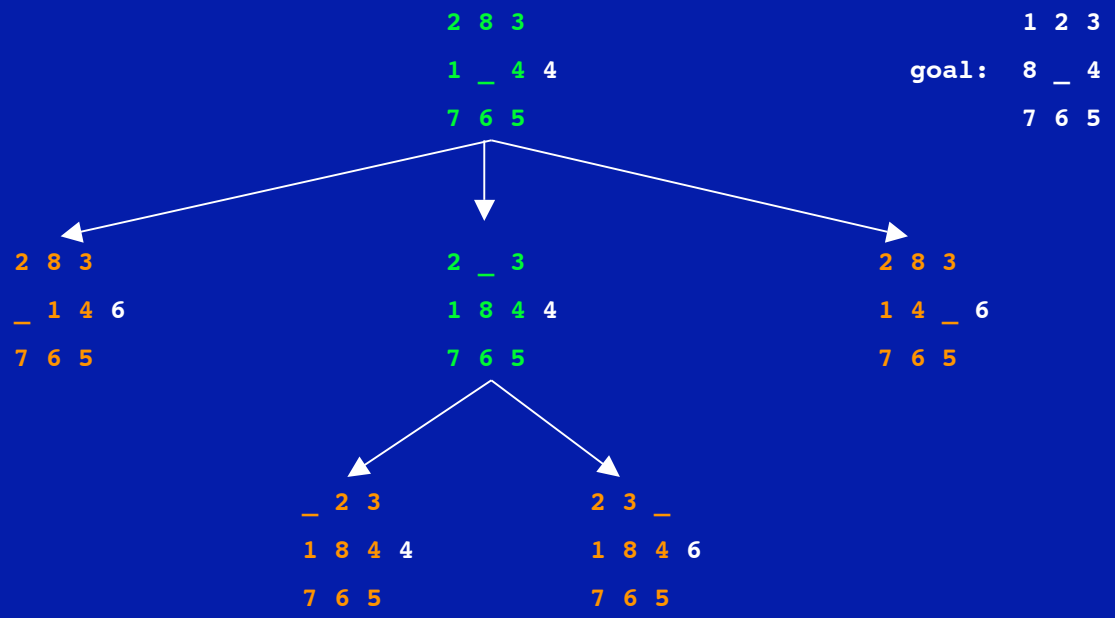
By the way, those reductions have to be significant (i.e., big)

Heuristic best-first search - Manhattan distance

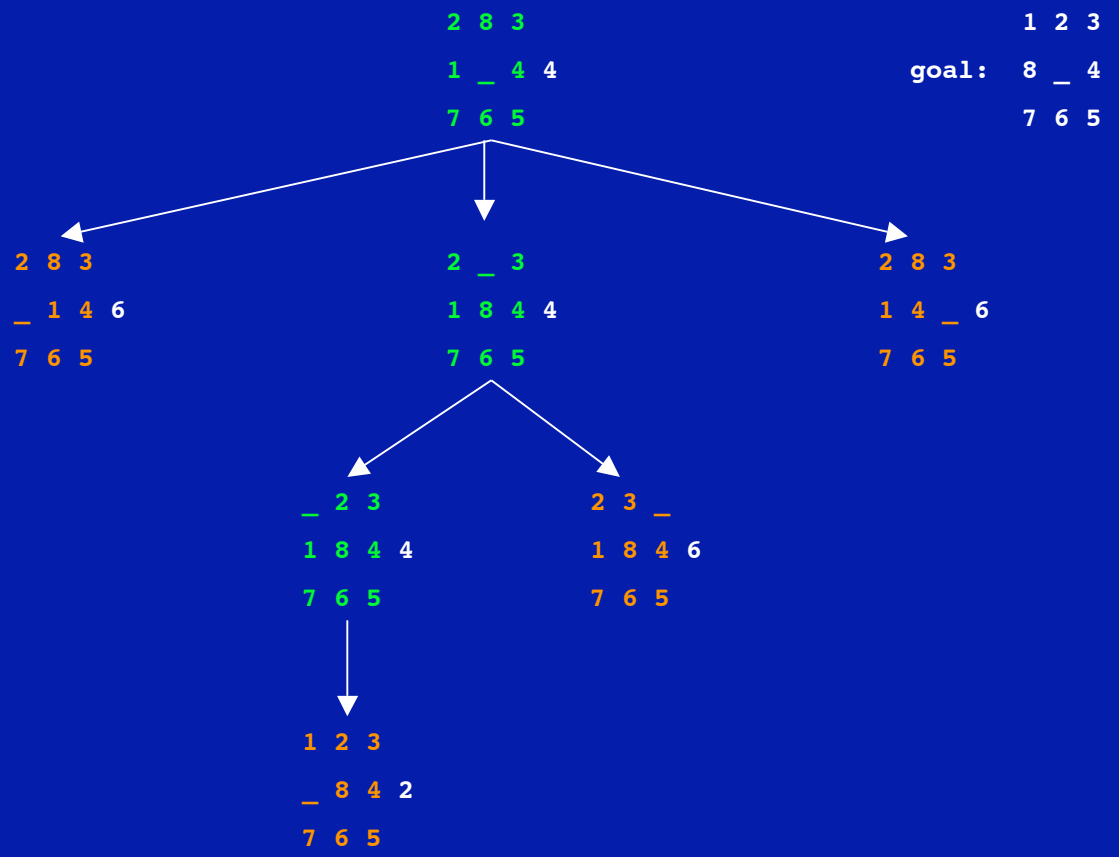
Heuristic best-first search - Manhattan distance



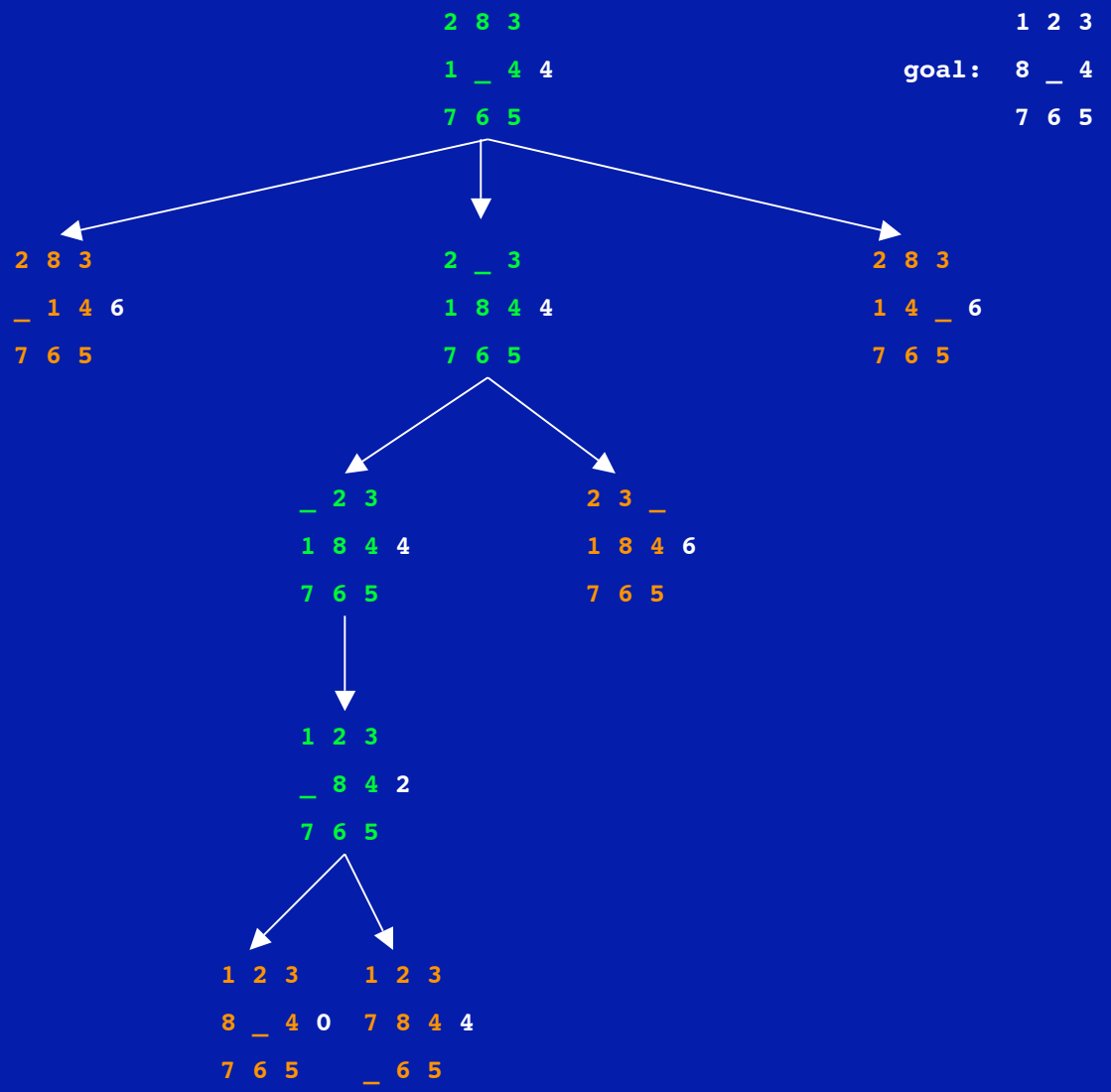
Heuristic best-first search - Manhattan distance



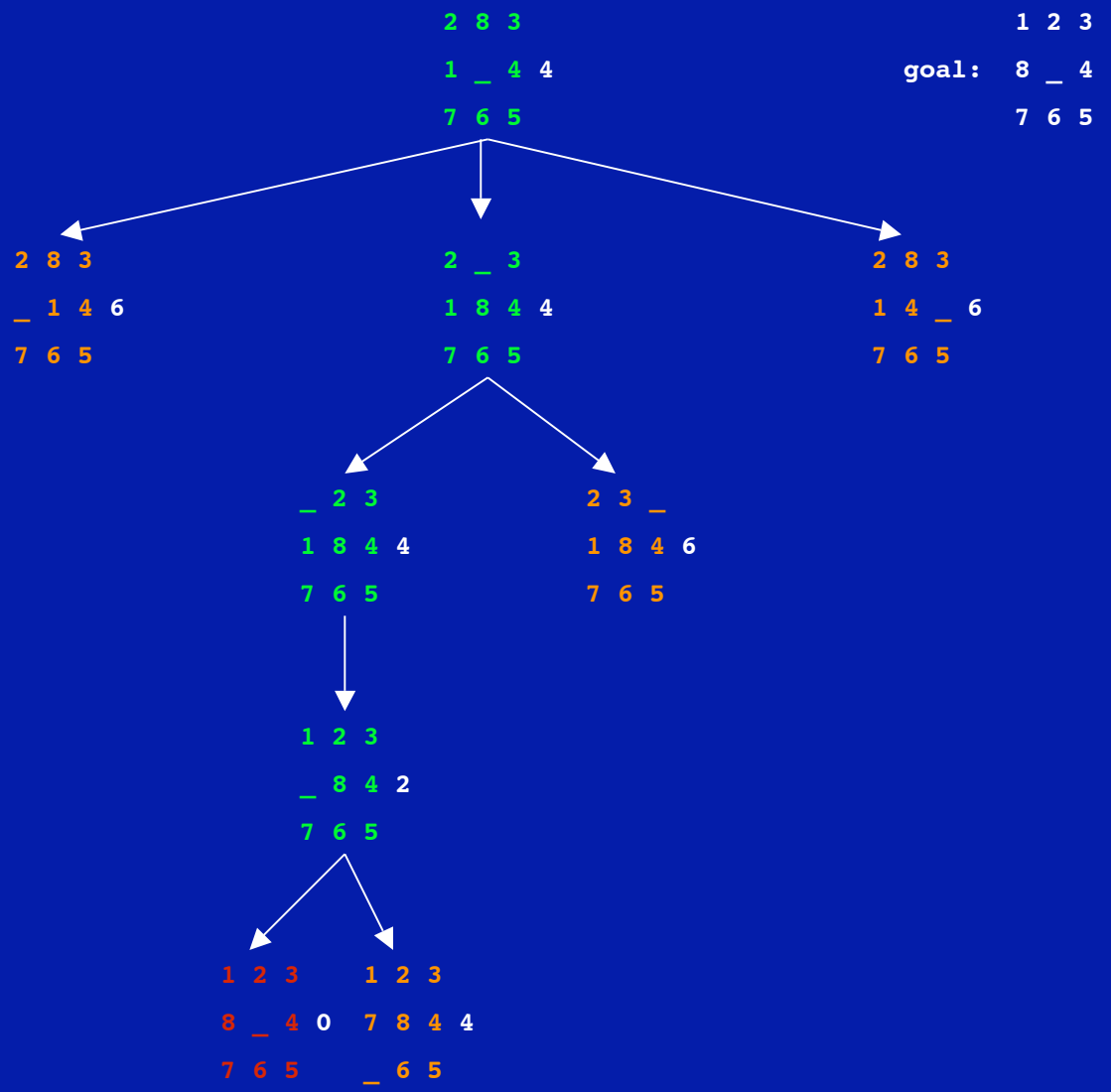
Heuristic best-first search - Manhattan distance



Heuristic best-first search - Manhattan distance



Heuristic best-first search - Manhattan distance



Move generation

```
move([0,X,C1,C2,C3,C4,C5,C6,C7],[X,0,C1,C2,C3,C4,C5,C6,C7]).
move([0,C1,C2,X,C3,C4,C5,C6,C7],[X,C1,C2,0,C3,C4,C5,C6,C7]).

/* ... and so on ... */
```

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 generate next nodes (neighbors)
        and put next nodes and h(n) values on frontier
        and sort frontier by h(n) values
end repeat

More to come Wednesday...

- add path collection
- add the ability to beat the world's best chess players

Questions?