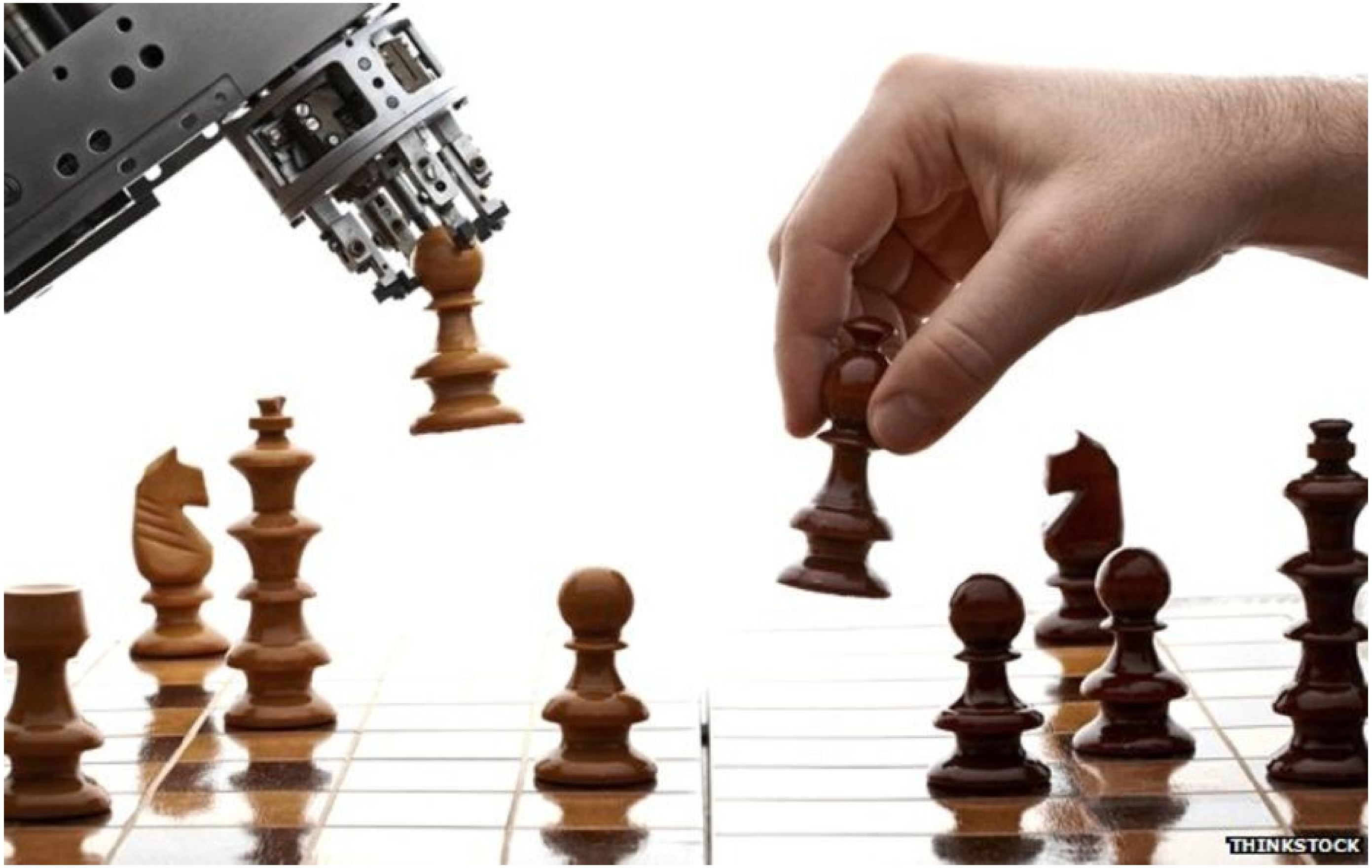


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

Professor Cheney

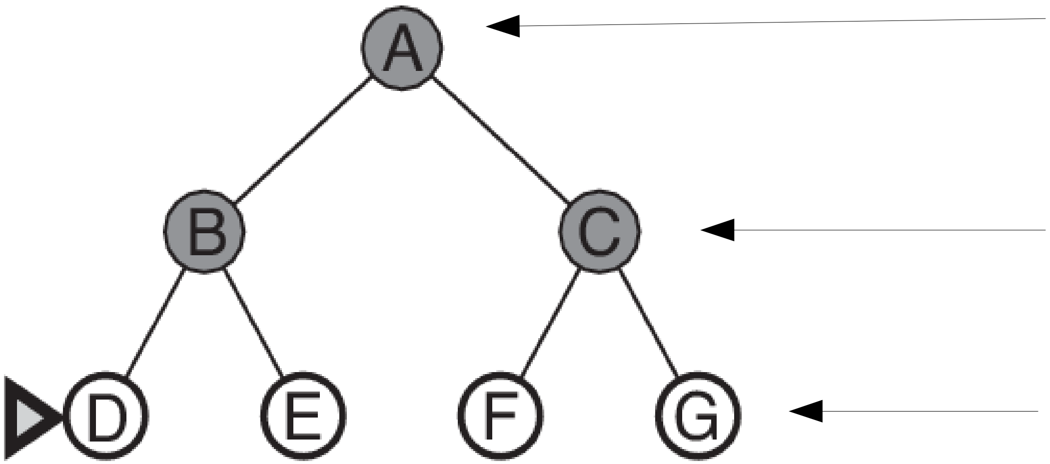
9/11/17

how best should we expand the frontier in search? (cont.)

breadth-first search

nodes visited:

$b = 2$



$1 = 2^0$

$2 = 2^1$

$4 = 2^2$

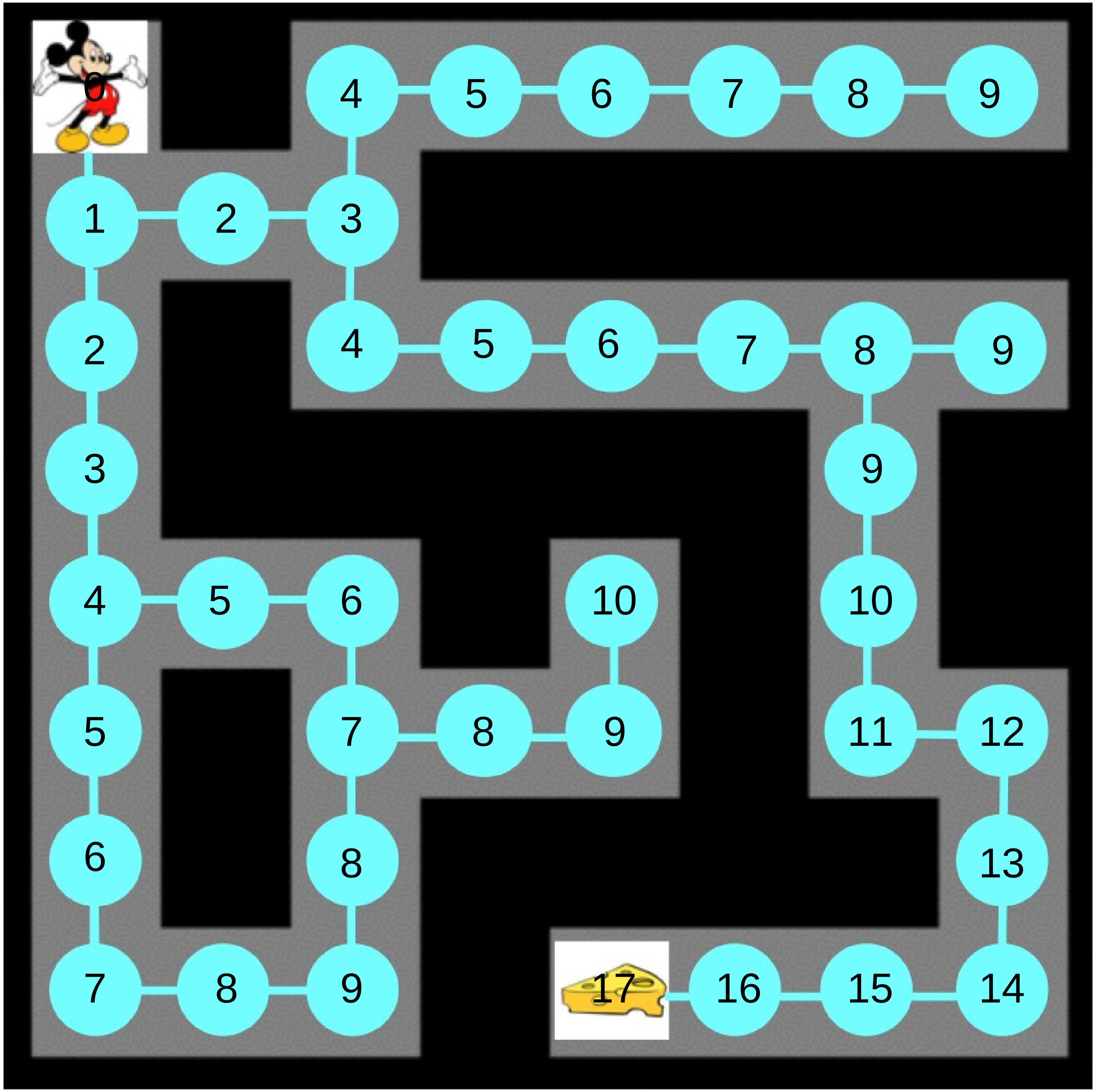
time complexity: $b^0 + b^1 + b^2 + \ldots + b^d = O(b^d)$

space complexity: $b^d = O(b^d)$

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

note: a typical chess game ~ 40-70 moves long
and has a branching factor ~ 35

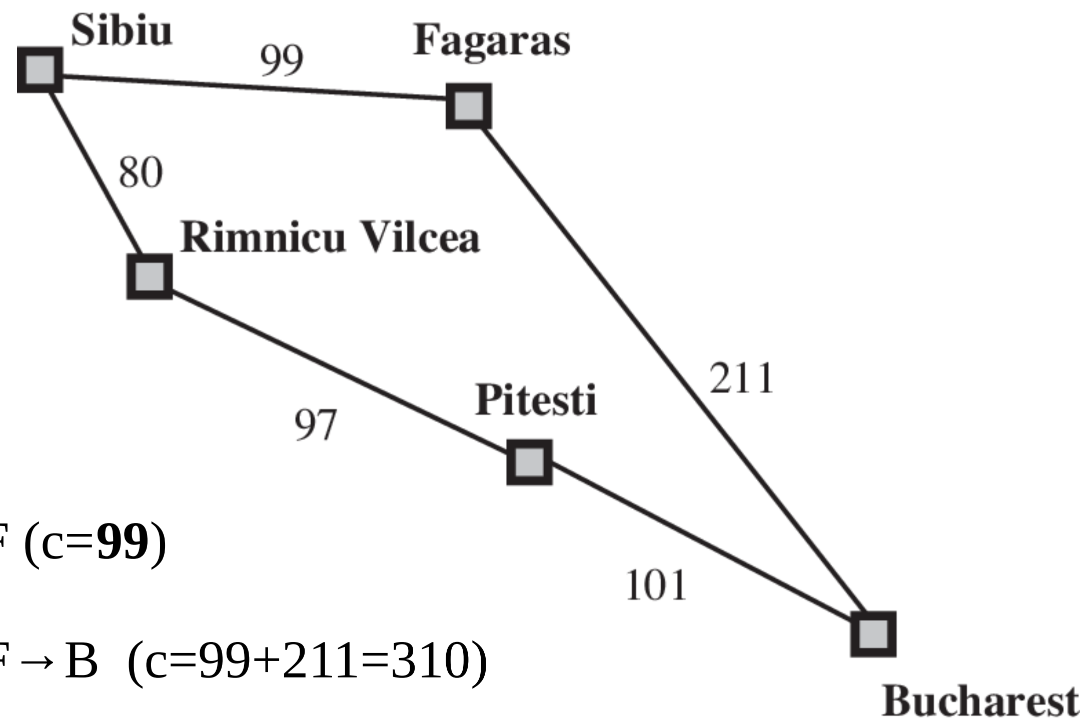

0
4 5 6 7 8 9
1 2 3
2 4 5 6 7 8 9
3 9
4 5 6 10 10
5 7 8 9 11 12
6 8 13
7 8 9 17 16 15 14

breadth-first search always finds the shallowest goal node

but shallowest is only optimal if all step-costs are equal

uniform-cost search

expand the frontier node with the current lowest cost



1) frontier: Sibiu (cost = **0**)

2) frontier: S → RV (c=**80**), S → F (c=99)

3) frontier: S → RV → P (c=80+97=177), S → F (c=**99**)

4) frontier: S → RV → P (c=80+97=**177**), S → F → B (c=99+211=310)

5) frontier: S → RV → P → **B** (c=80+97+101=**278**), S → F → B (c=99+211=310)

6) try to expand from goal node (B)??? → terminate search… we have a winner!

S → RV → P → B has lowest cost of 278

We expand from the least cost node/path

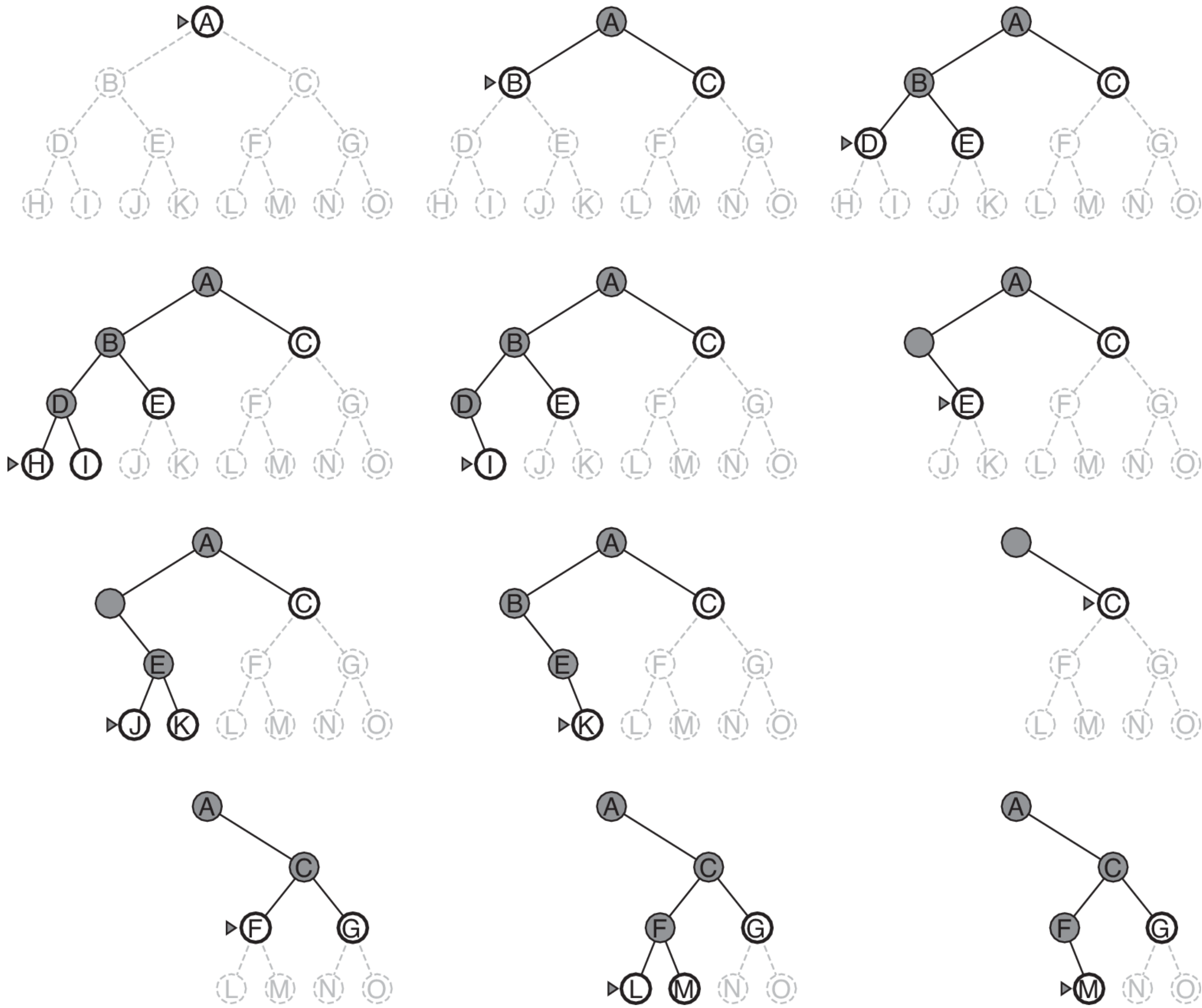
If we expand from a goal node,
that must be the least cost path to that node

"uniform-cost search expands nodes
in order of their optimal path"

depth-first search

expand the deepest frontier node

(LIFO)

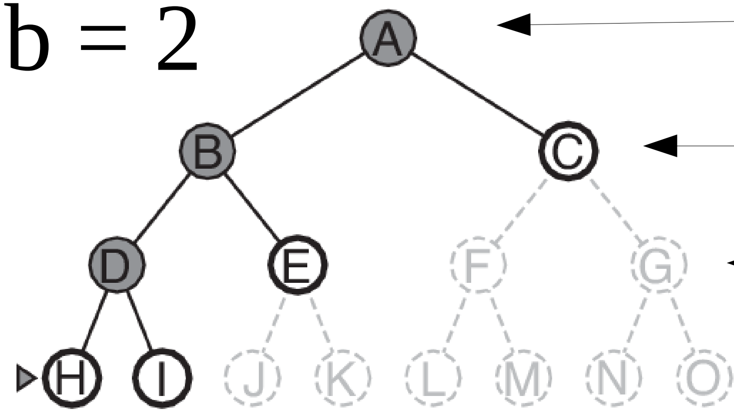


why??? we already know BFS finds shallowest goal node?

let's look at the complexity:

nodes visited:

$b = 2$



$1 = 2^0 = b^0$

$2 = 2^1 = b^1$

$4 = 2^2 = b^2$

$8 = 2^3 = b^3$

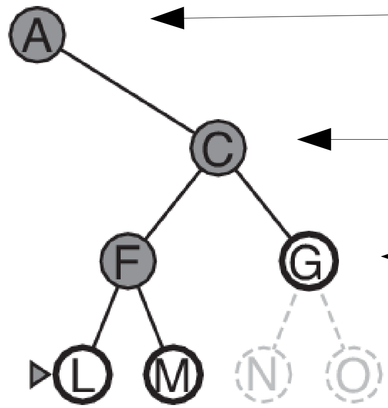
time complexity: $b^0 + b^1 + b^2 + b^3 + \ldots + b^m = O(b^m)$

(m is the max depth, d is the shallowest goal node)

this can't be better than BFS's $O(b^d)$ (since $m \geq d$)

nodes stored:

b = 2



$1 = 1$

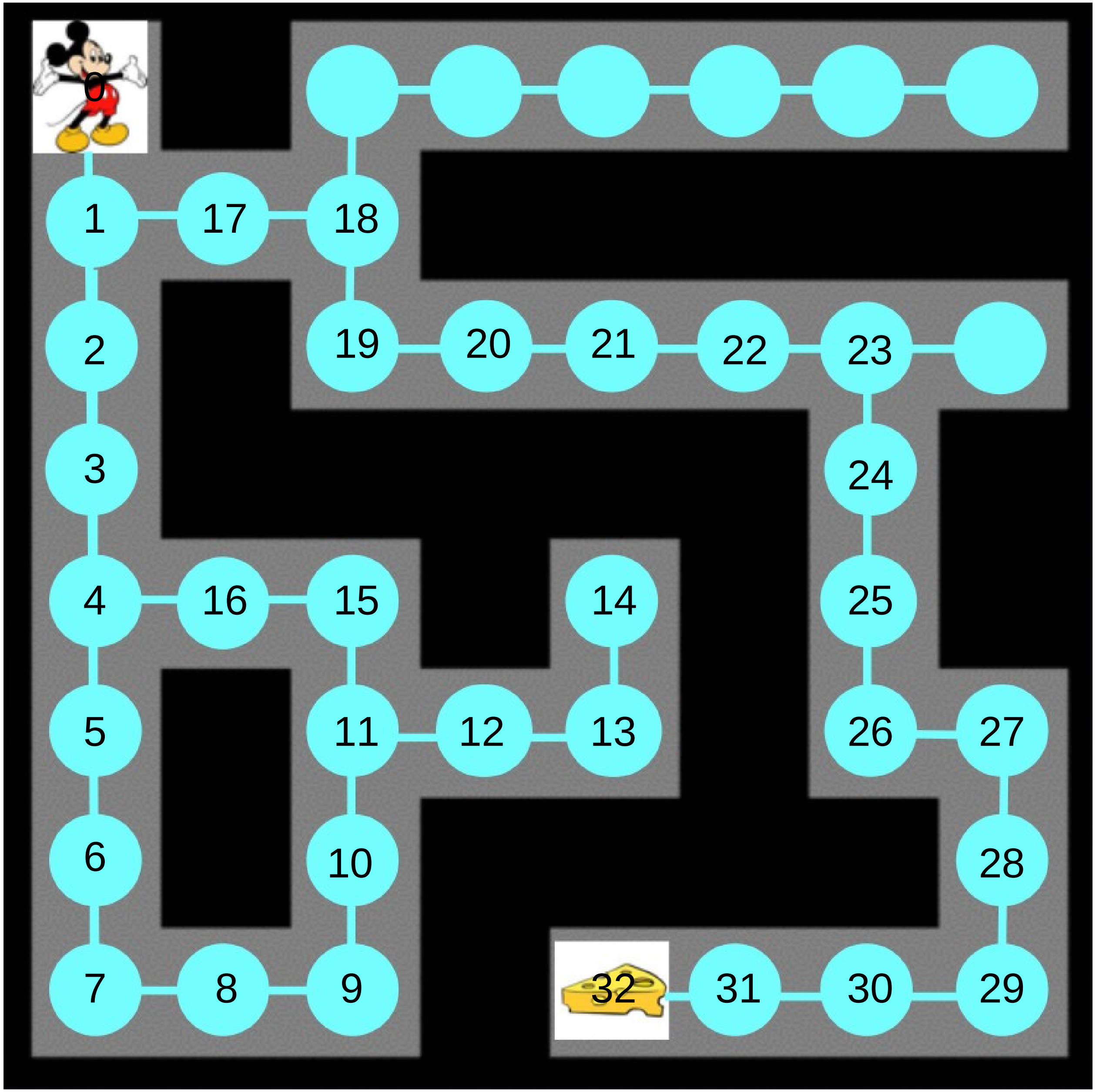
$2 = b$

$2 = b$

$2 = b$

space complexity: $1 + b + b + b + \ldots + b = O(bm)$

DFS has a better space complexity than BFS's $O(b^d)$!

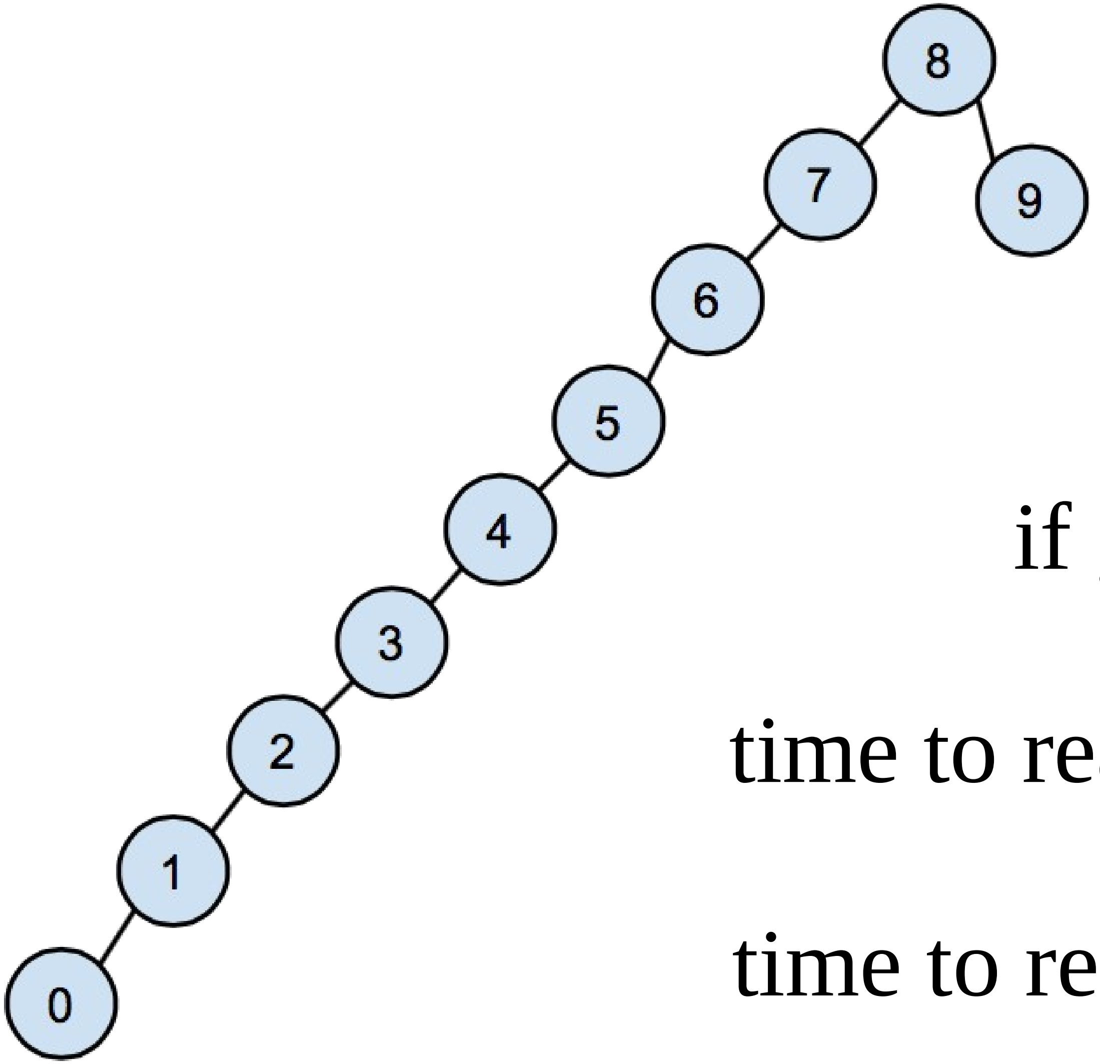

0
1
17
18
2
19
20
21
22
23
3
24
4
16
15
14
25
5
11
12
13
26
27
6
10
28
7
8
9
32
31
30
29

DFS has a seemingly similar time complexity and better space complexity than BFS

why don't we always use it? what can go wrong?

$$m >> d$$

unbalanced trees!

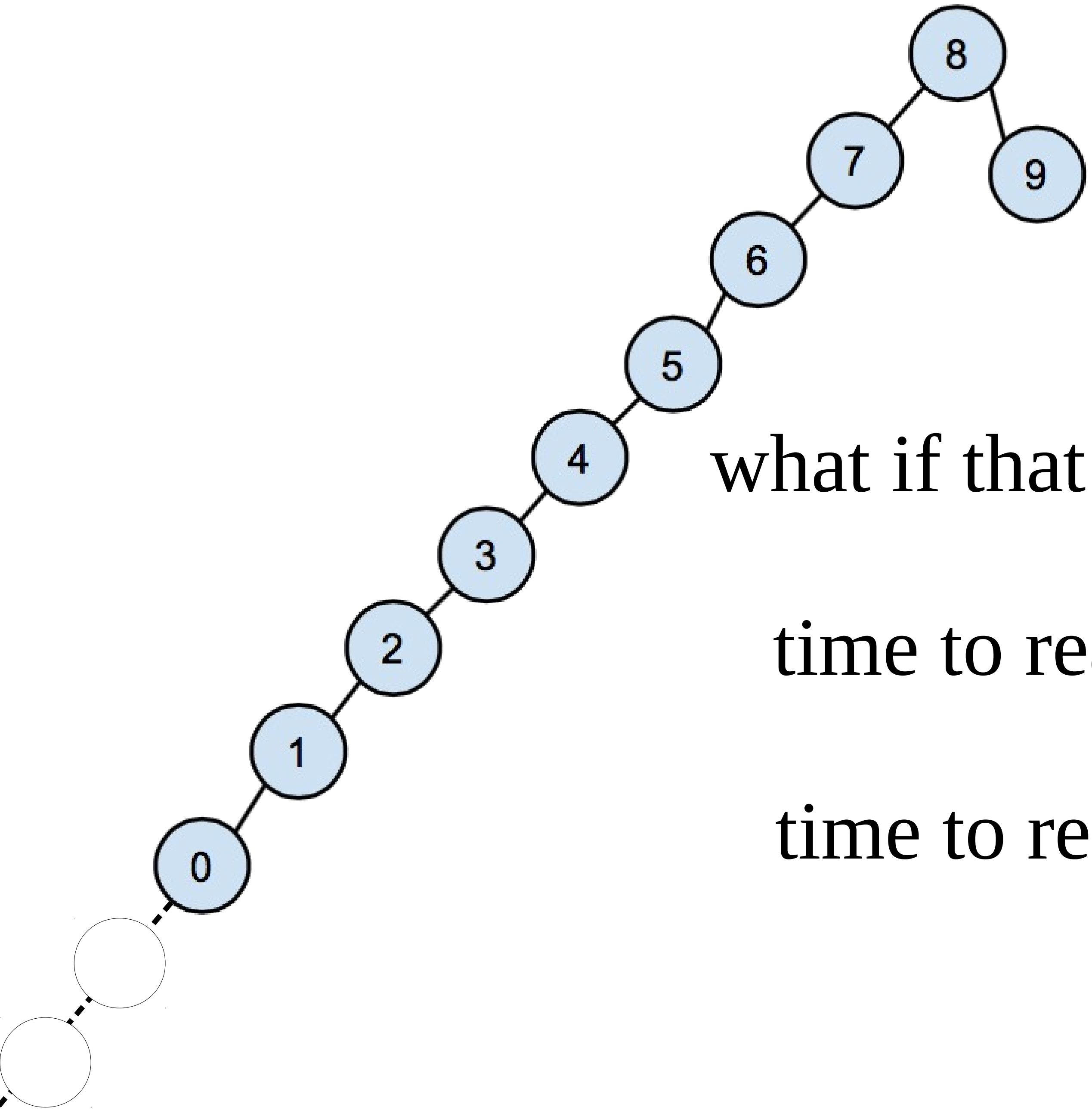


if goal node is #9?

time to reach goal w/ DFS: 10

time to reach goal w/ BFS: 3

unbalanced trees!



what if that branch never ended?

time to reach goal w/ DFS: ∞

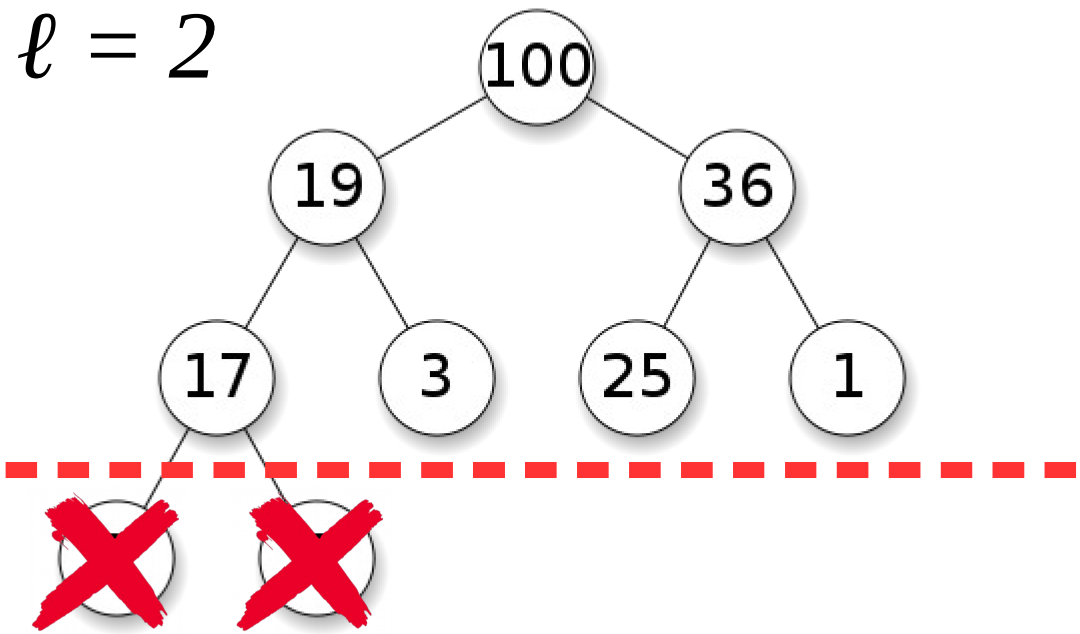
time to reach goal w/ BFS: 3

how can we limit the depth of branches in DFS?

depth-limited search

do DFS, but cut short any branches greater than ℓ edges deep

$\ell = 2$



now we are just doing DFS on a balanced(-ish) tree!

from before (but now with max depth $m = \ell$):

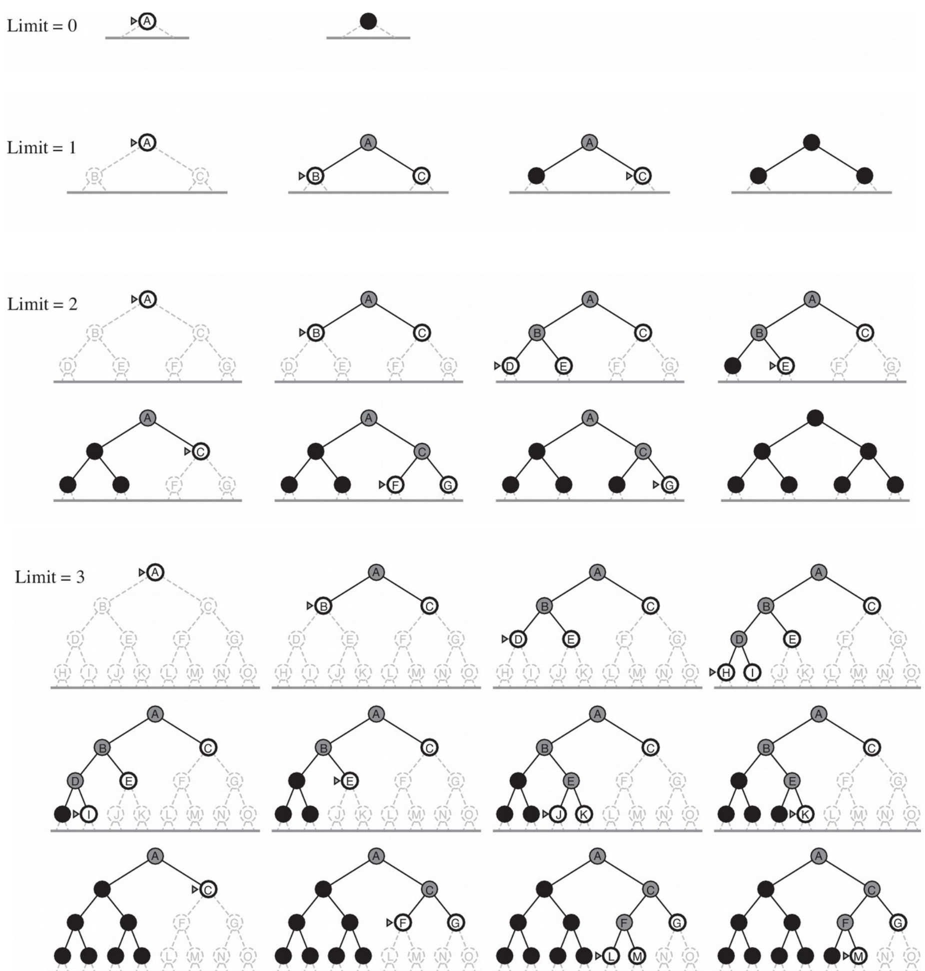
time complexity = $O(b^{\ell})$
space complexity = $O(b\ell)$

but what's the most efficient value of ℓ?

let's just do them all! (huh…?)

iterative-deepening search

iteratively do depth-limited search for $\ell=0$, $\ell=1$, $\ell=2$, …



	time: $O(b^{\ell})$	space: $O(b\ell)$
$\ell=0$:	$O(1)$	$O(0)$
$\ell=1$:	$O(b)$	$O(b)$
$\ell=2$:	$O(b^2)$	$O(2b)$
$\ell=3$:	$O(b^3)$	$O(3b)$

so when do we stop?

when we hit the shallowest goal node!

this occurs when $\ell = d$

What's the complexity of all $\ell=0$, $\ell=1$, $\ell=2$, … $\ell=d$, together?

time complexity:

$$O(b^0) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d)$$

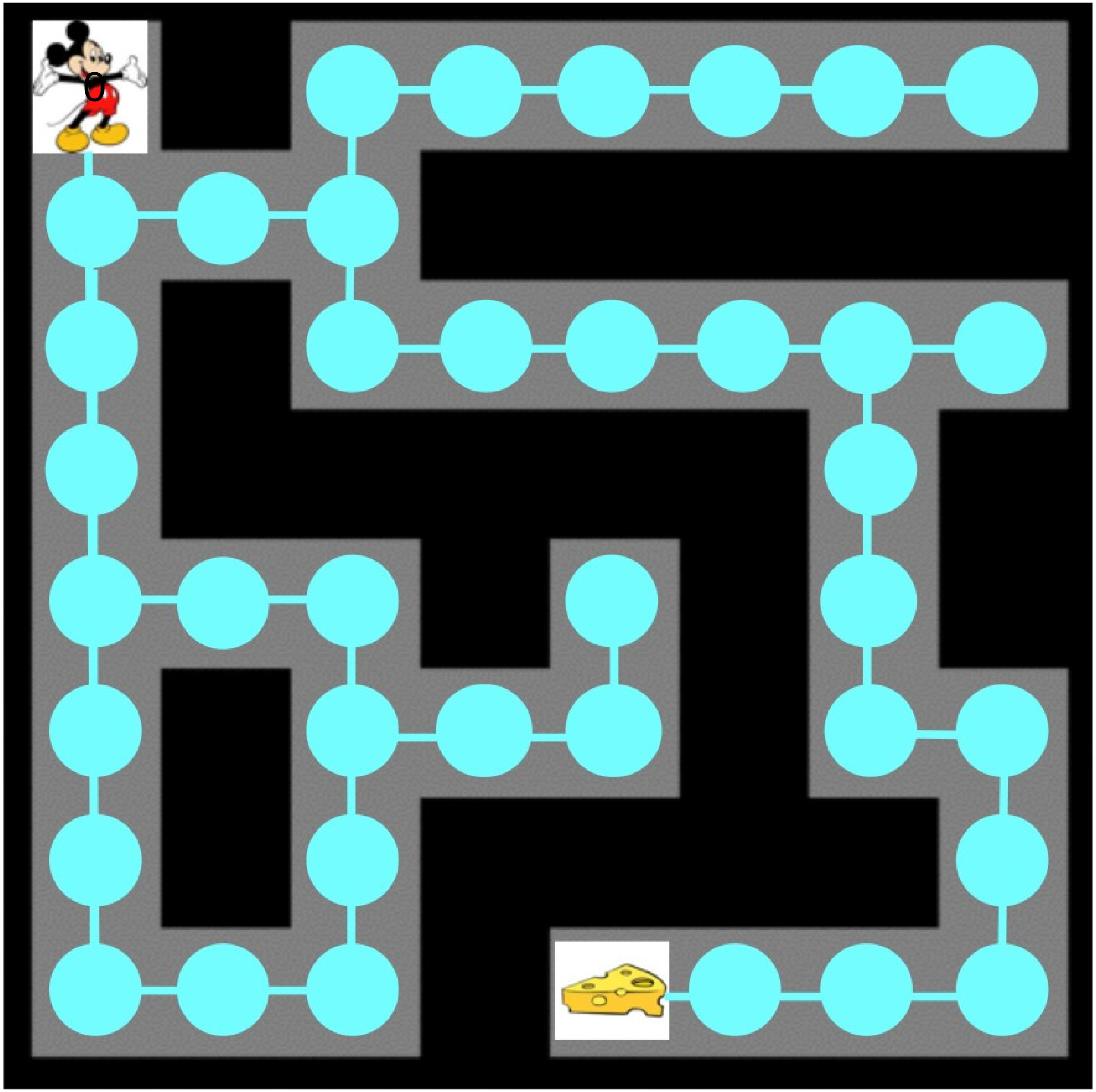
$\ell=0$ $\quad$ $\ell=1$

space complexity:

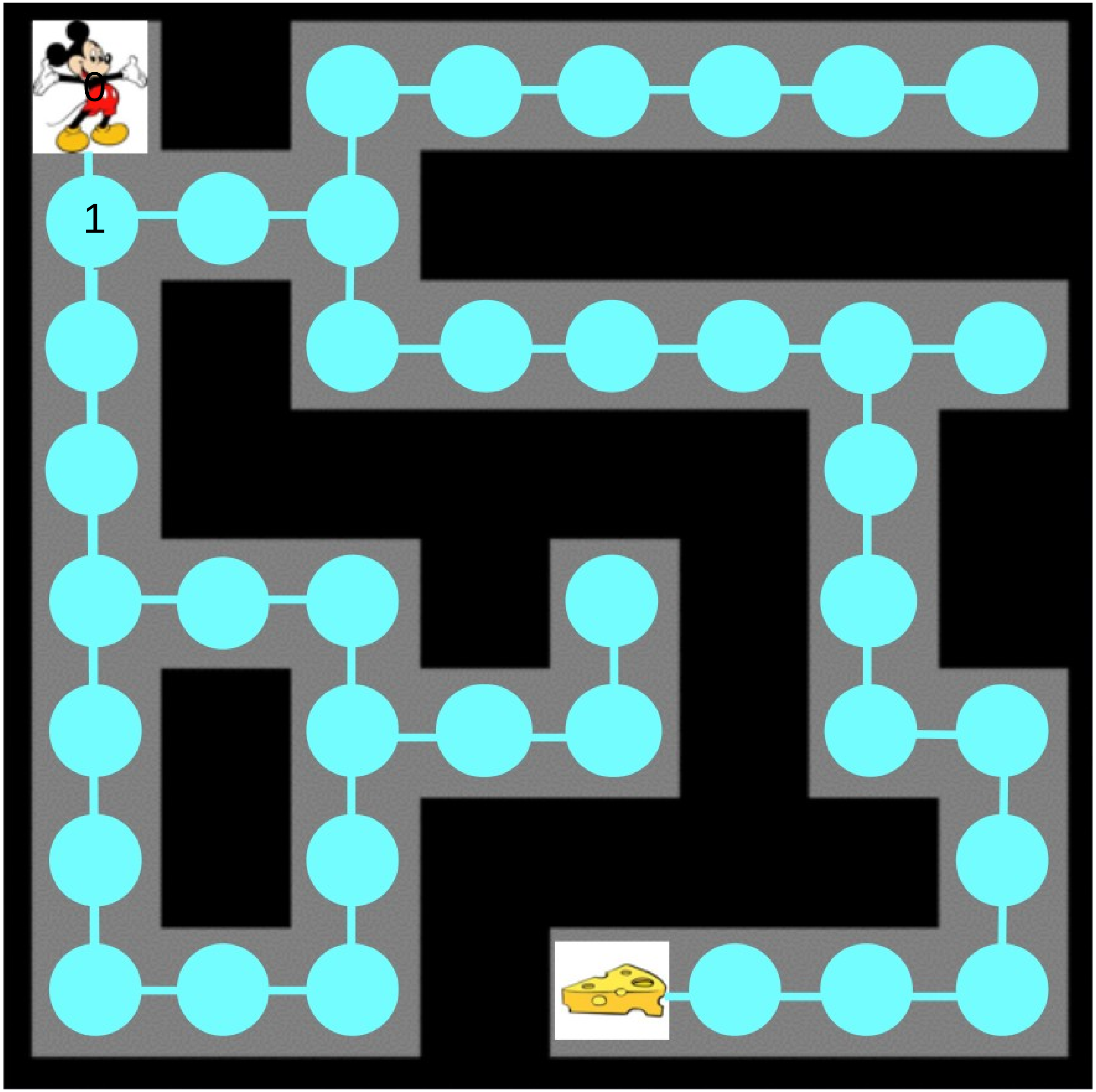
$$O(0) + O(b) + O(2b) + \ldots + O(db) = O(db)$$

iterative deepening search has better (or as good) time and space complexity compared to DFS and BFS!

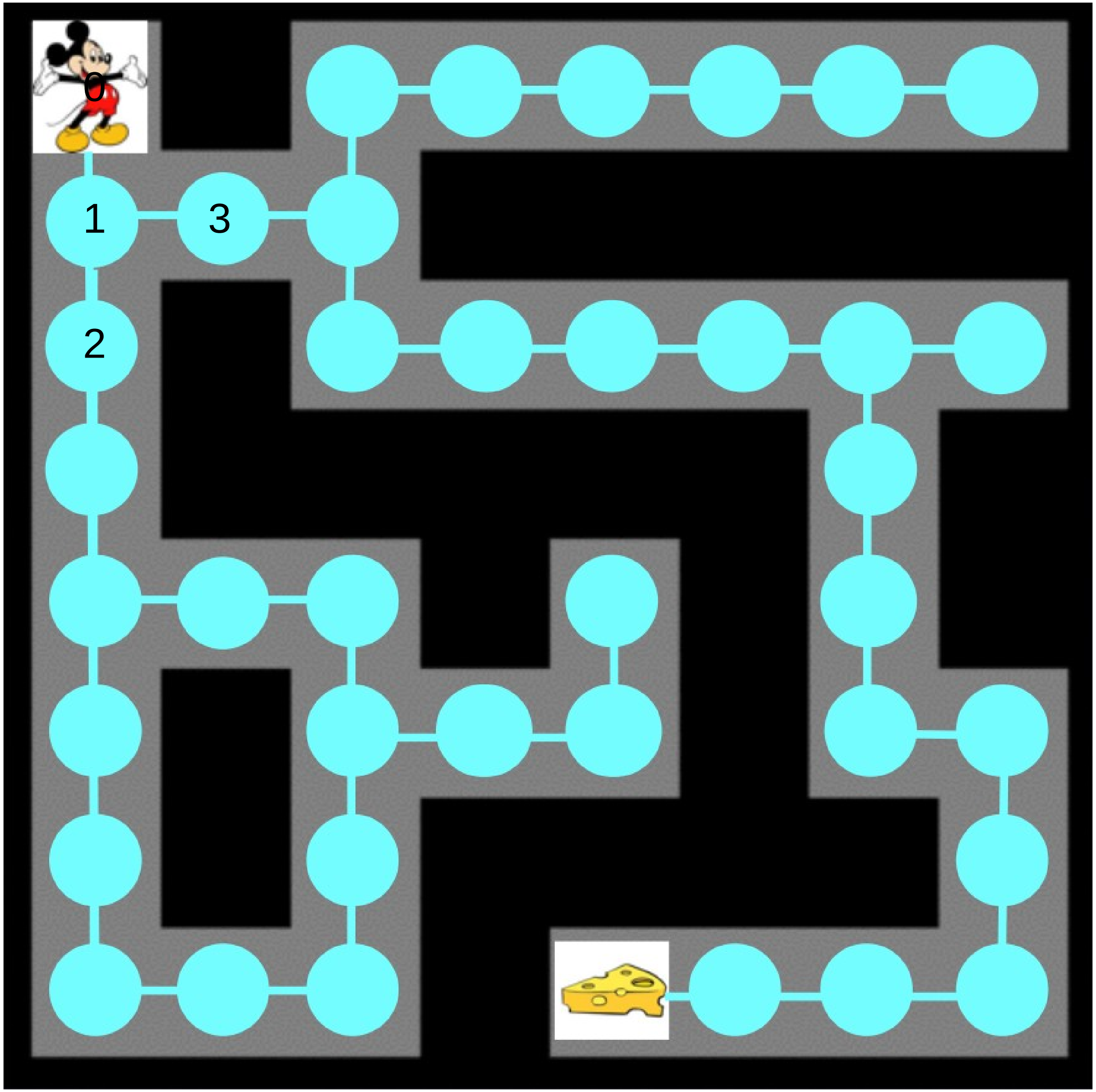
$\ell = 0$



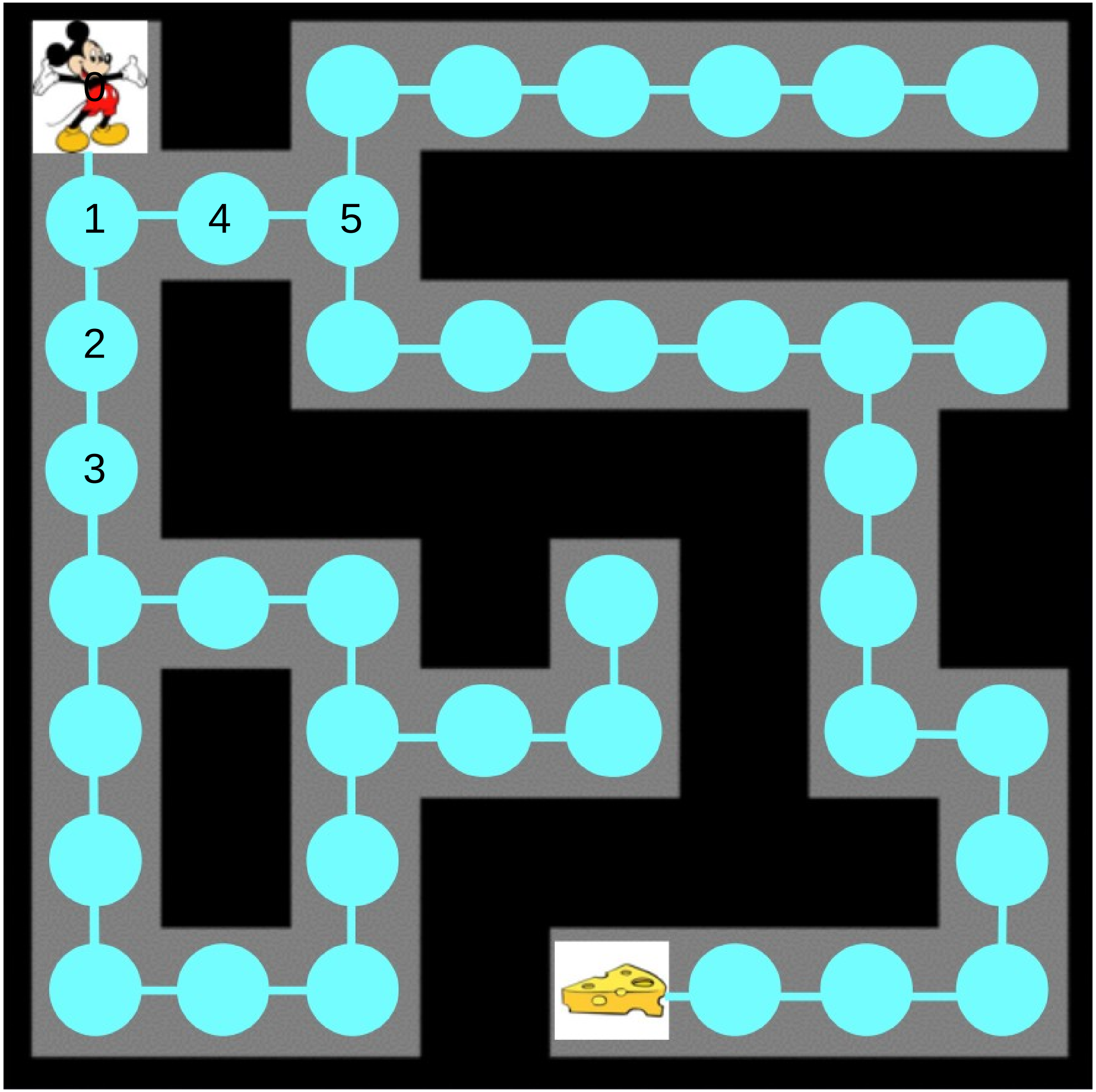
$\ell = 1$



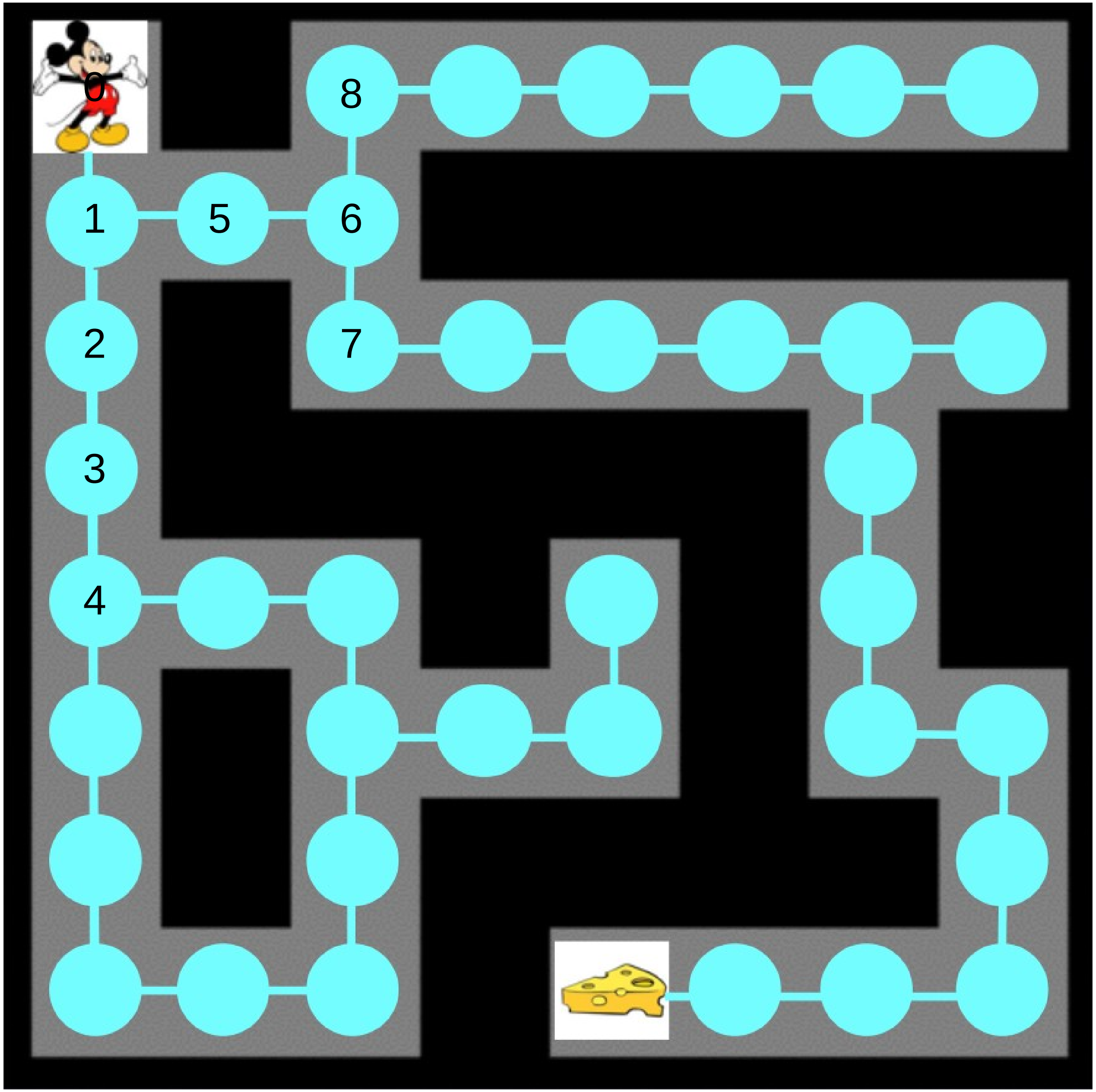
$\ell = 2$



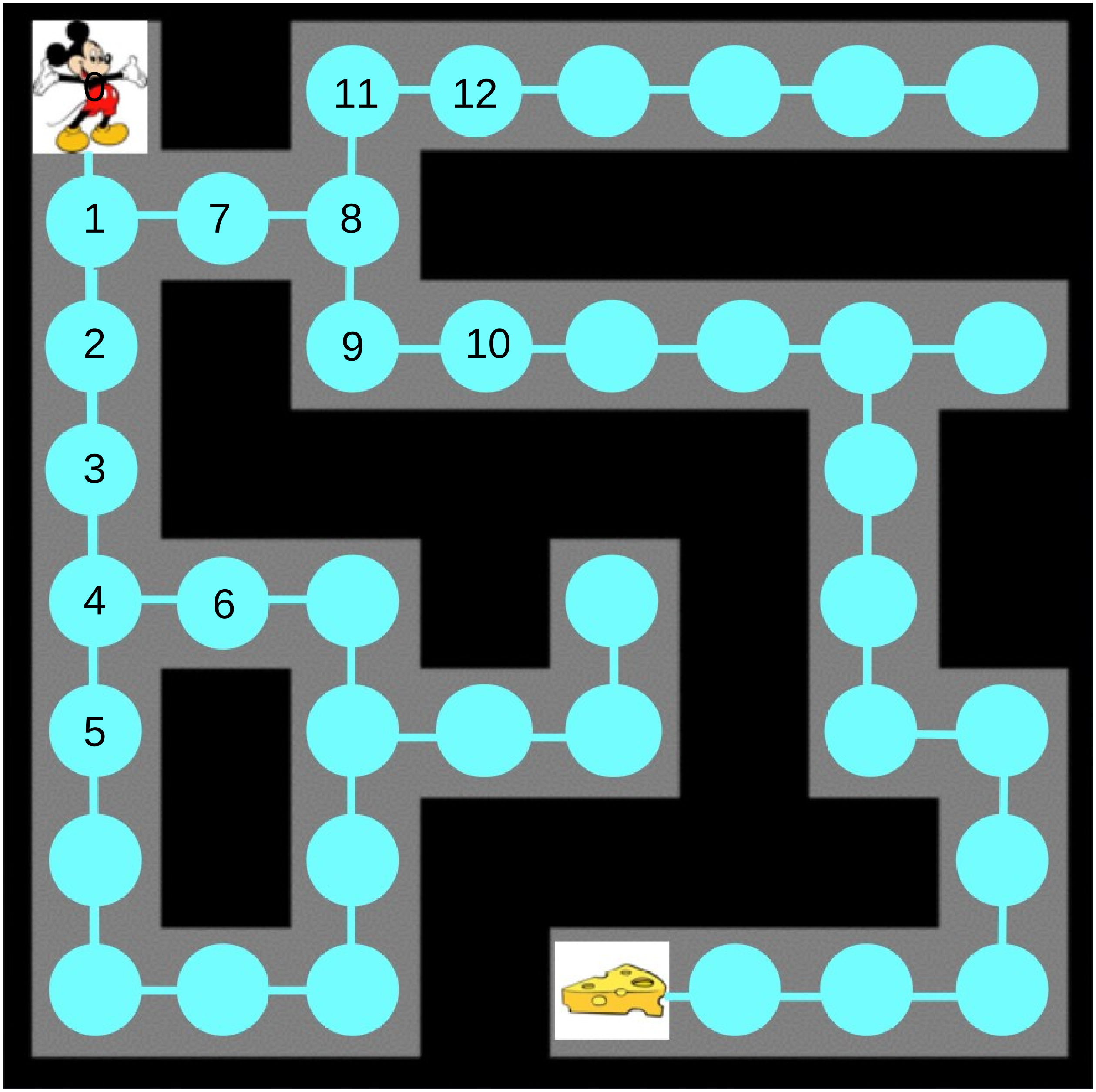
$\ell = 3$



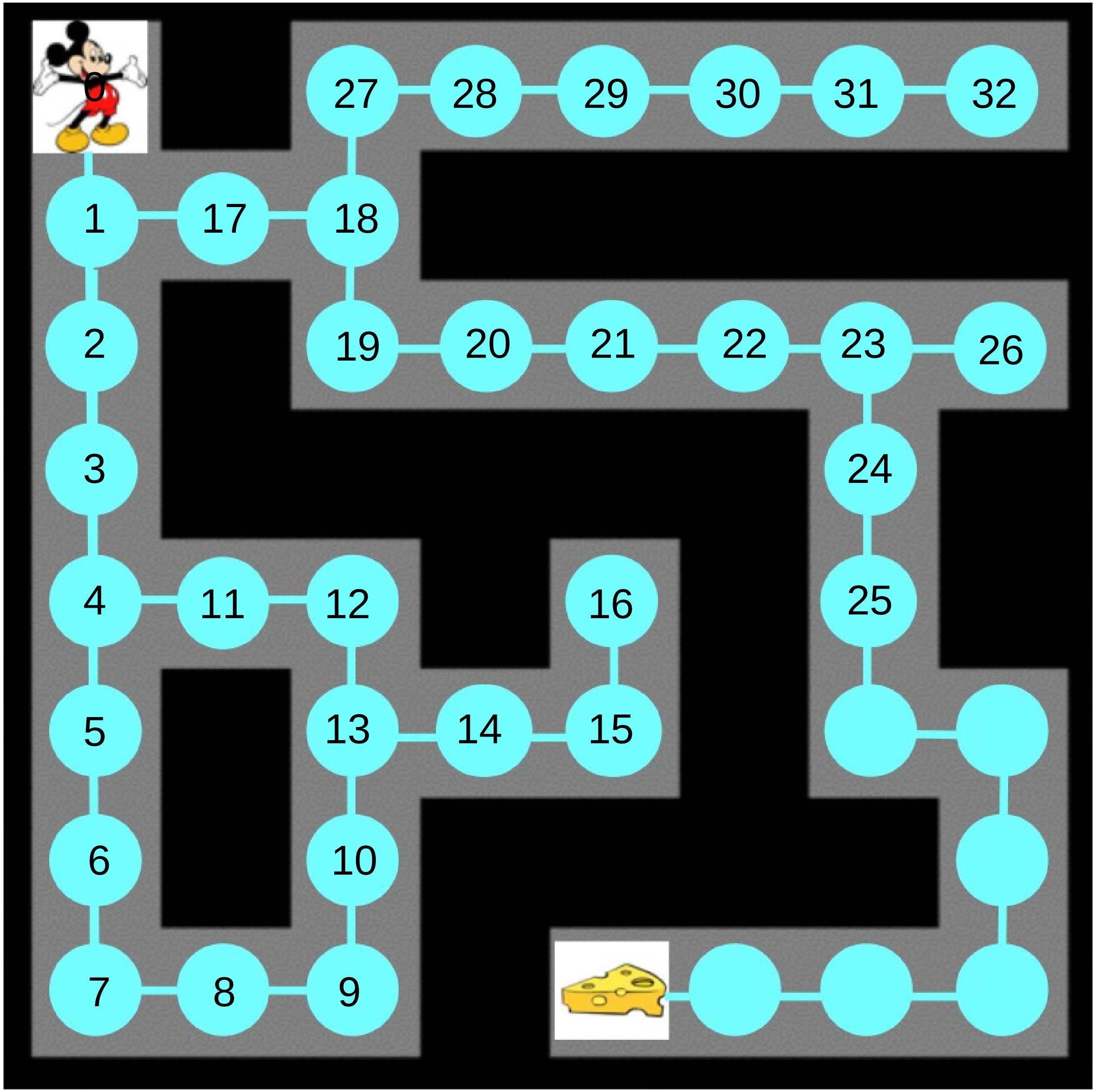
$\ell = 4$



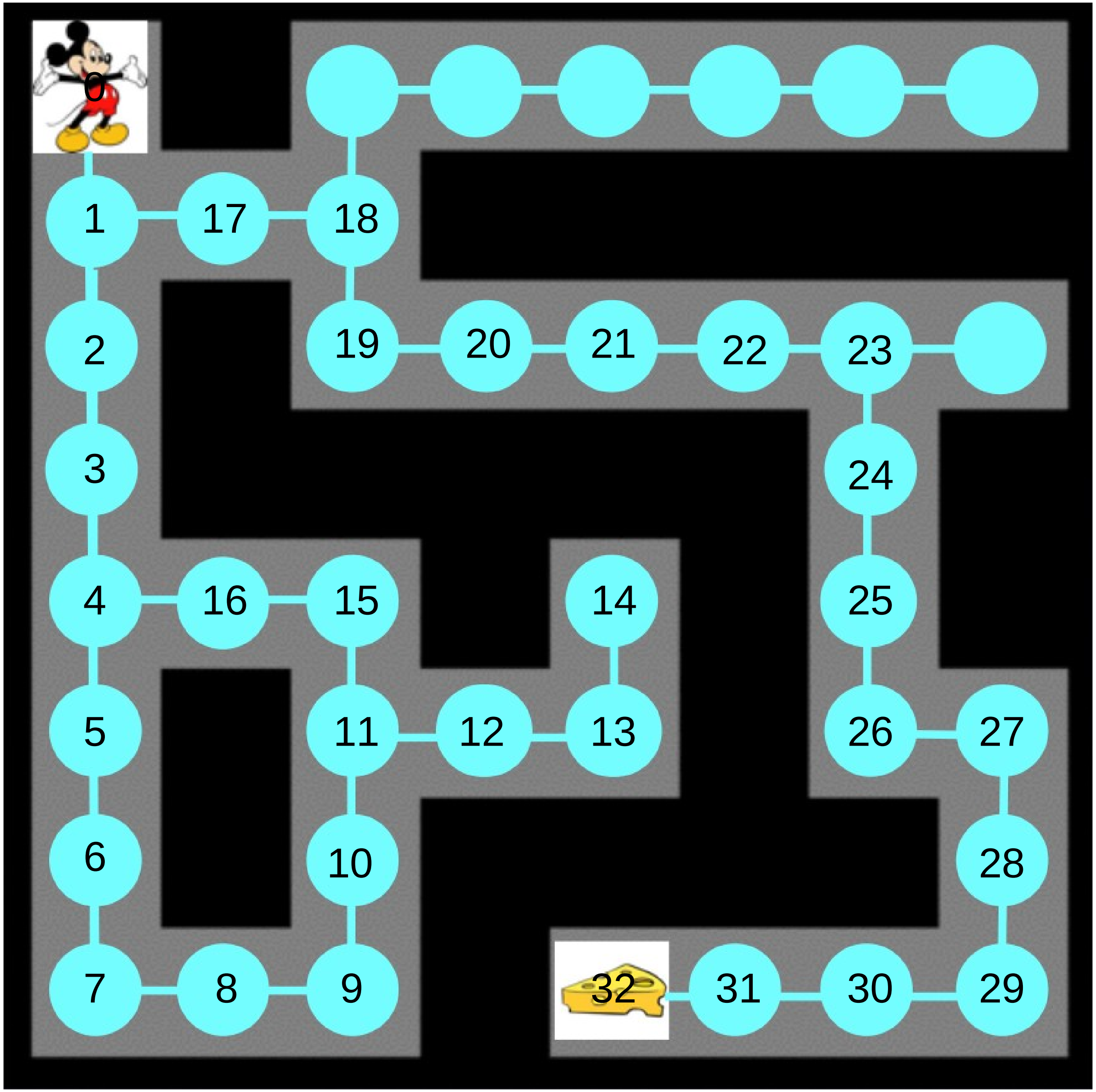
$\ell = 5$



$\ell = 10$


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

$\ell = 17$


0
1
17
18
19
20
21
22
23
2
3
24
4
16
15
14
25
5
11
12
13
26
27
6
10
28
7
8
9
32
31
30
29

iterative-deepening search expands away from start node like BFS, but has memory efficiency like DFS

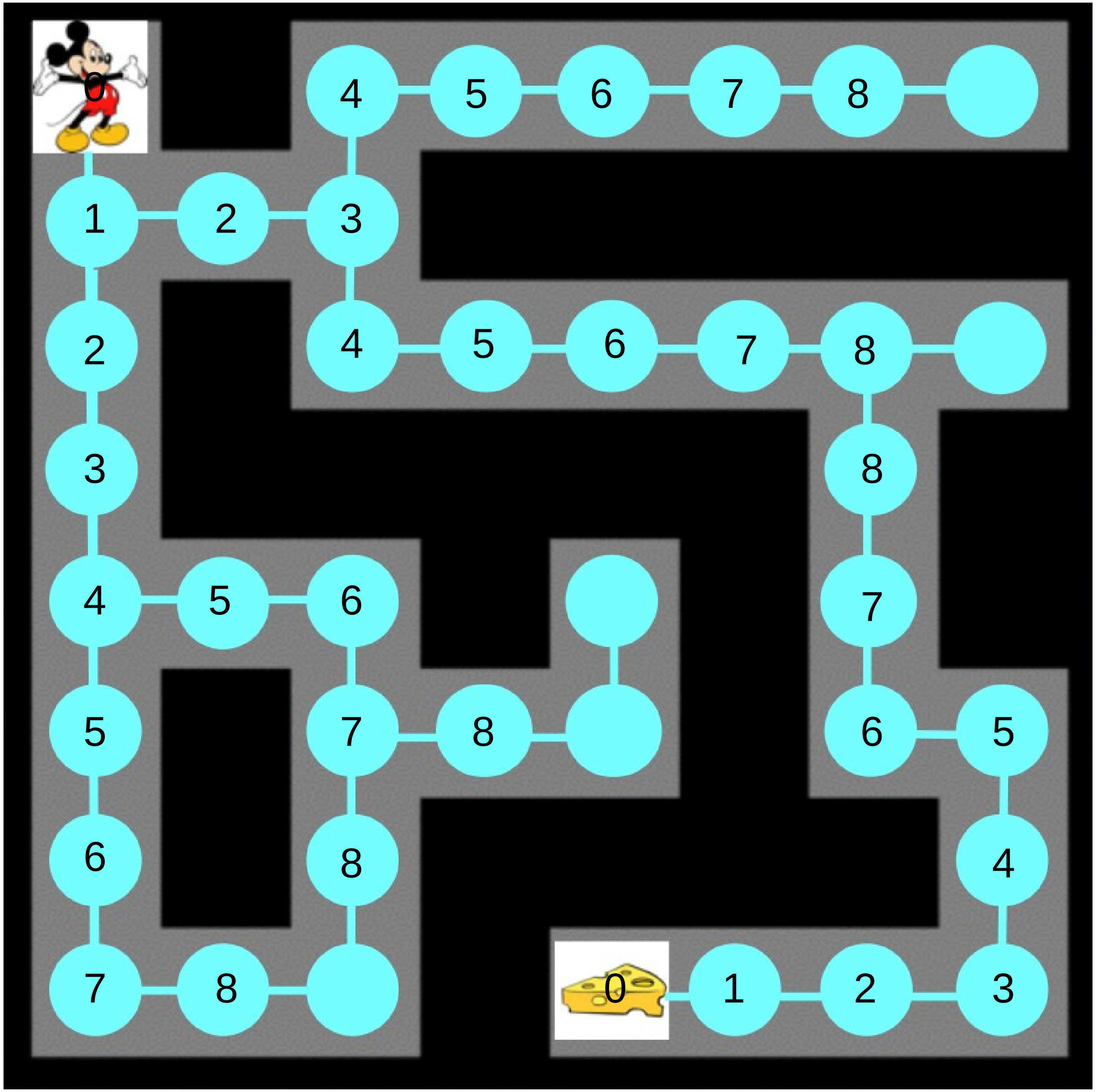
how can we make it even better?

the most expensive cases are long paths to the goal

(long paths are the most expensive for all algorithms, since the # of nodes grows exponentially with depth)

bidirectional search

if we know what our goal state is,
we can treat it as a second “start” state
and work backwards until the two searches meet


0
4 5 6 7 8
1 2 3
2
4 5 6 7 8
3
8
4 5 6
7
5
7 8
6 5
6
8
4
7 8
0 1 2 3

bidirectional breadth first search

if the total path length to the goal is *d*,
each search only has to go *d*/2 nodes until they meet

time complexity: $2*b^0 + 2*b^1 + 2*b^2 + \ldots + 2*b^{d/2} = O(b^{d/2})$

space complexity: $2*b^0 + 2*b^1 + 2*b^2 + \ldots + 2*b^{d/2} = O(b^{d/2})$

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

note: bidirectional search can cut the depth of most of these algorithms in half