A4B36ZUI - Introduction to ARTIFICIAL INTELLIGENCE

https://cw.fel.cvut.cz/wiki/courses/a4b33zui/start

Michal Pechoucek, Branislav Bosansky, Jiri Klema & Olga Stepankova

Department of Computer Science
Czech Technical University in Prague




OTEVŘENÁ INFORMATIKA

In parts based on cs121.stanford.edu & S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. 3rd edition, Prentice Hall, 2010

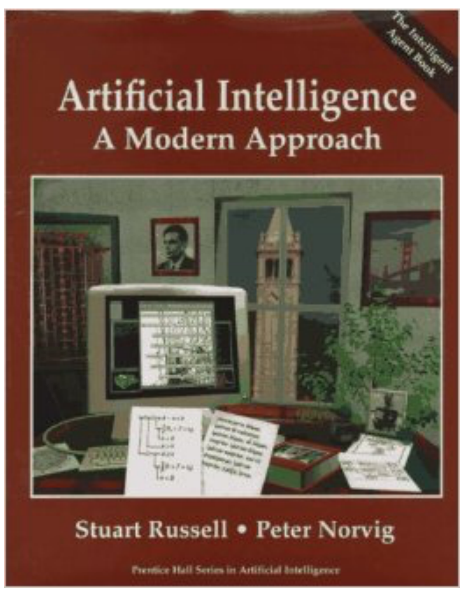
Přednášky z předmětu A4B33ZUI / Lectures for A4B33ZUI

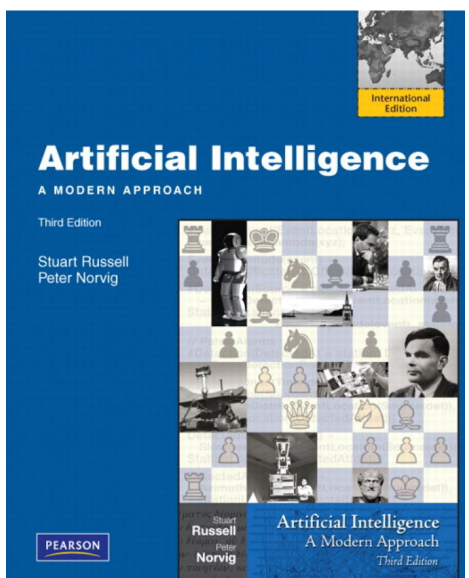
No.	Téma přednášky / Topic	Datum / Date	Čas / Time	Místnost / Room	Slidy / Slides	Staré slidy / Old Slides	Přednášející / Lecturer
1	Introduction, search problems, Uninformed search algorithms	21.2.2017	14:30	KN:E-107		1 2	Michal Pechoucek
2	Informed search algorithms, A*	28.2.2017	14:30	KN:E-107		3 3	Michal Pechoucek
3	Advanced A*	7.3.2017	14:30	KN:E-107			Michal Pechoucek
4	Two-player Games	14.3.2017	14:30	KN:E-107		PDF	Branislav Bosansky
5	Constraint Satisfaction Programming	21.3.2017	14:30	KN:E-107		4 EN BIN LS	Michal Pechoucek
6	Two-player Games II	28.3.2017	14:30	KN:E-107		5 2014_slides	Branislav Bosansky
7	Knowledge representation - introduction	4.4.2017	14:30	KN:E-107		7	Jiri Klema
8	Knowledge representation in FOL	11.4.2017	14:30	KN:E-107		8	Jiri Klema
9	Rational decisions under uncertainty	18.4.2017	14:30	KN:E-107		9	Jiri Klema
10	Sequential decisions under uncertainty	25.4.2017	14:30	KN:E-107		10	Jiri Klema
11	---	2.5.2017	—	—	Tuesday's Schedule		
12	Knowledge in Multiagent Systems	9.5.2017	14:30	KN:E-107		11	Olga Stepankova
13	Formal system for MOL, Temporal Logic for Real-life Problems	17.5.2017	14:30	KN:E-107		12 13a 13b-LTL	Olga Stepankova
14	AI an Applications	23.5.2017	14:30	KN:E-107			Michal Pechoucek

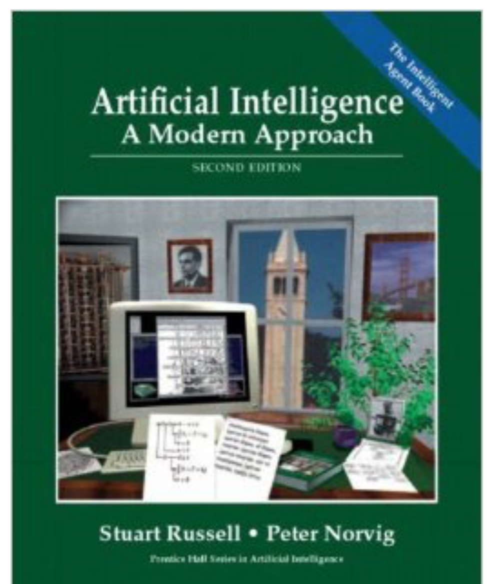
Přednášky z předmětu A4B33ZUI / Lectures for A4B33ZUI

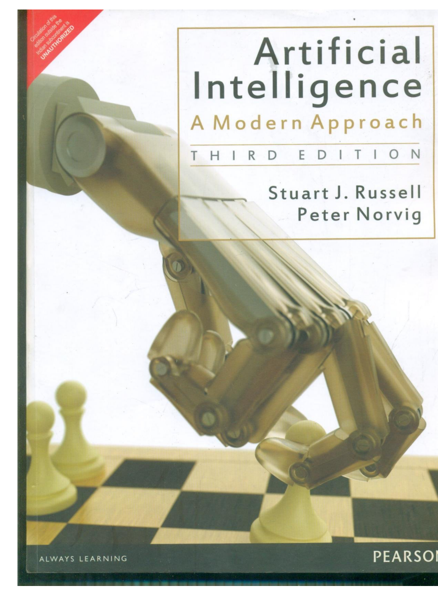
No.	Téma přednášky / Topic	Datum / Date	Čas / Time	Místnost / Room	Slid
1	Introduction, search problems, Uninformed search algorithms	21.2.2017	14:30	KN:E-107	
2	Informed search algorithms, A*	28.2.2017	14:30	KN:E-107	
3	Advanced A*	7.3.2017	14:30	KN:E-107	
4	Two-player Games	14.3.2017	14:30	KN:E-107	
5	Constraint Satisfaction Programming	21.3.2017	14:30	KN:E-107	
6	Two-player Games II	28.3.2017	14:30	KN:E-107	
7	Knowledge representation - introduction	4.4.2017	14:30	KN:E-107	
8	Knowledge representation in FOL	11.4.2017	14:30	KN:E-107	
9	Rational decisions under uncertainty	18.4.2017	14:30	KN:E-107	
10	Sequential decisions under uncertainty	25.4.2017	14:30	KN:E-107	
11	—-	2.5.2017	—	—	Tue
12	Knowledge in Multiagent Systems	9.5.2017	14:30	KN:E-107	
13	Formal system for MOL, Temporal Logic for Real-life Problems	17.5.2017	14:30	KN:E-107	
14	AI an Applications	23.5.2017	14:30	KN:E-107	

http://aima.cs.berkeley.edu









Introduction to AI
Uninformed Search

R&N: Chap. 3, Sect. 3.1–3.6

Why Search Matters in AI?

AI Today?

Why Search Matters in AI?

AI Today?

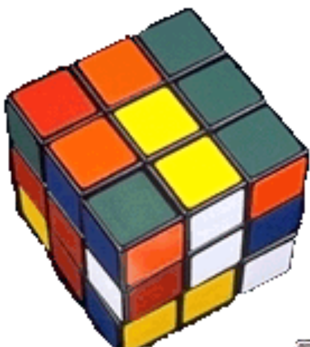
1. Machine Learning
   (Computational Statistics, Mathematical Optimisation)
   perception, understanding, prediction, classification
2. Automated Reasoning
   (Symbolic AI, Search based AI)
   problem solving, decision making, planning

Why Search Matters in AI?

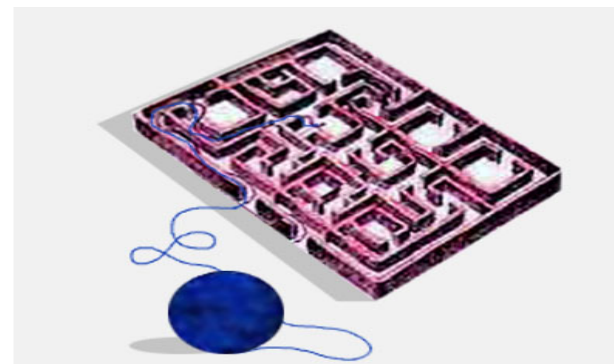
AI Today?

1. Machine Learning
   (Computational Statistics, Mathematical Optimisation)
   perception, understanding, prediction, classification
2. Automated Reasoning
   (Symbolic AI, Search based AI)
   problem solving, decision making, planning
3. Machine Learning + Automated Reasoning


MASH SOMATIC MAPS
PIXIE WAYSIDE ESAU
ODELL ARTICLE LIT
WILLIAMSHAKESPEARE
XS SENIOR
AMFM IRC RIGHT ANY
SALMANRUSHDIE ARSE
ALUMNI BUYIN RC
ONEI MO PH
ERNESTHEMINGWAY
ABE ES MOA
SMBD ULTRA YONDER
MISS ALLENGINSBERG
UN ODEAR EVE CRAB
TOILER SO
BARBARAKINGSOLVER
ELENA MALARIA MOIRE
REACT ONANIST PLASM
ROMEO KINESIS HALT




PUZZLE PICKS
60 PUZZLES
PUZZLE PICKS




Puzzle-Peg
Can YOU Solve It?
LUBBERS & BELL MFG. CO.
CLINTON, IOWA, U.S.A.
Puzzle-Peg


SOMETHING FISHY

Example: 8-Puzzle

8	2	
3	4	7
5	1	6

Initial state

1	2	3
4	5	6
7	8	

Goal state

State: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board

8-Puzzle: Successor Function

8	2	7
3	4	
5	1	6

SUCC(state) → subset of states

The successor function is knowledge about the 8-puzzle game, but it does not tell us which outcome to use, nor to which state of the board to apply it.

8	2	
3	4	7
5	1	6

8	2	7
3	4	6
5	1	

8	2	7
3		4
5	1	6

Search is about the exploration of alternatives

(n^2-1)-puzzle

8	2	
3	4	7
5	1	6

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

....

15-Puzzle

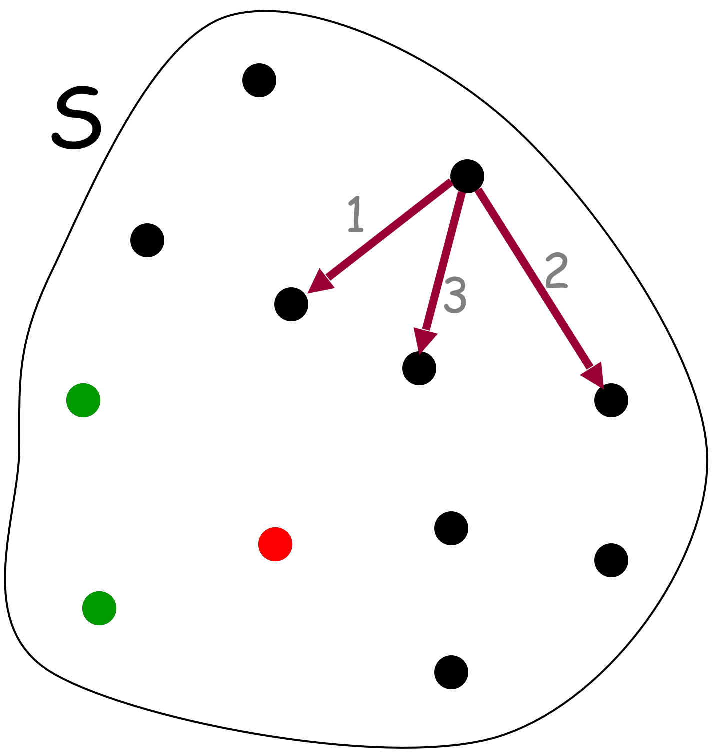
Sam Loyd offered $1,000 of his own money to the first person who would solve the following problem:

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

? →

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

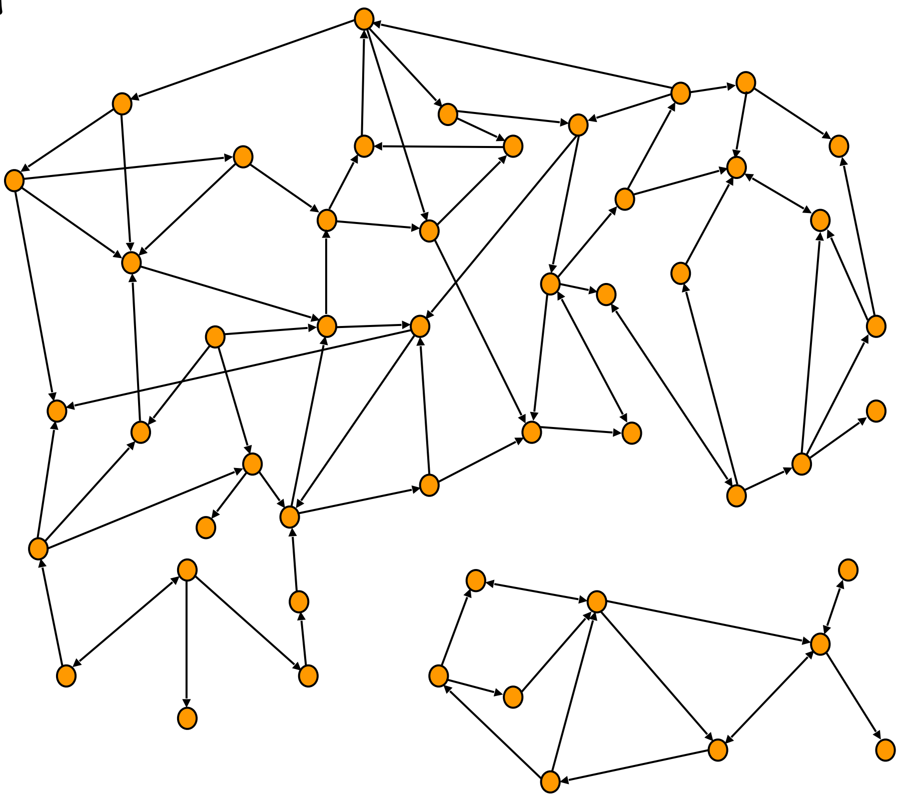
Stating a Problem as a Search Problem



- State space S
- Successor function:
  $x \in S \rightarrow$ SUCCESSORS$(x) \in 2^S$
- Initial state s_0
- Goal test:
  $x \in S \rightarrow$ GOAL?(x) = T or F
- Arc cost

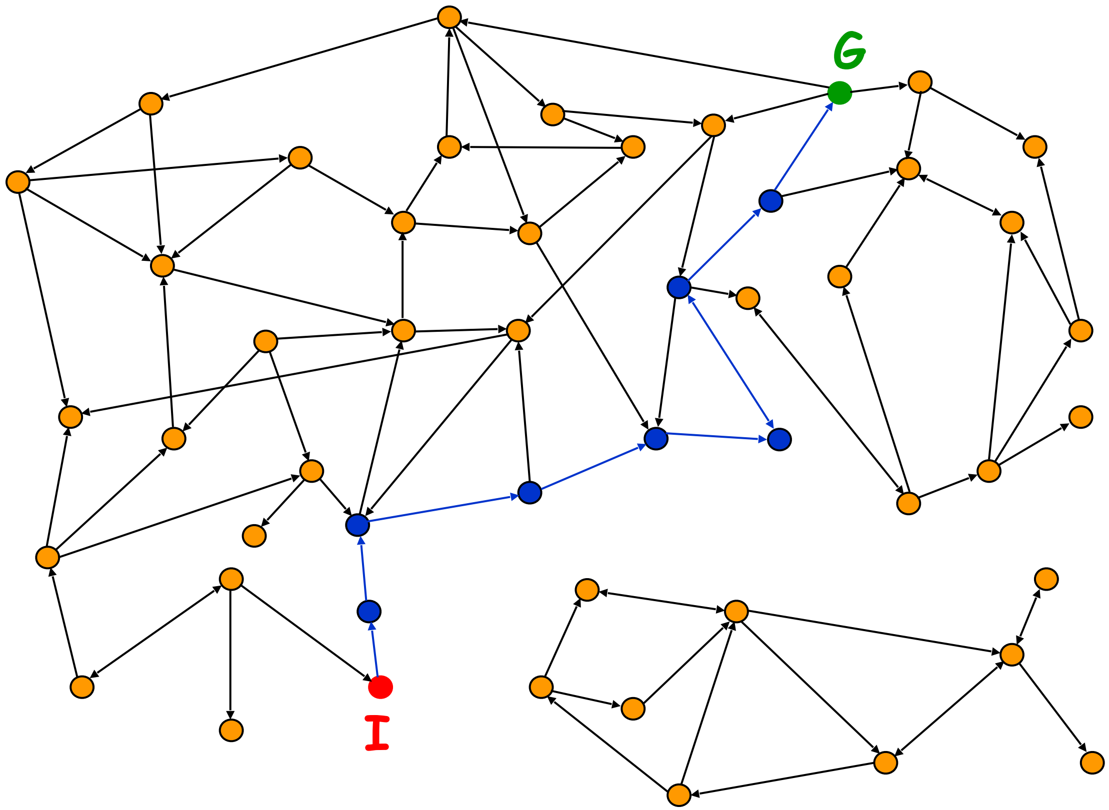
State Graph

- Each state is represented by a distinct node
- An arc (or edge) connects a node s
  to a node s' if
  s' $\in$ SUCCESSORS(s)
- The state graph may contain more than one connected component



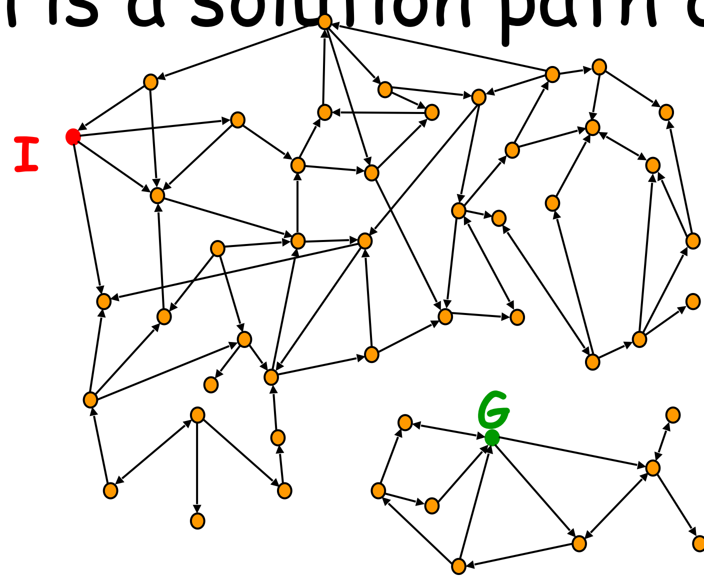
Solution to the Search Problem

- A solution is a path connecting the initial node to a goal node (any one)



Solution to the Search Problem

- A solution is a path connecting the initial node to a goal node (any one)
- The cost of a path is the sum of the arc costs along this path
- An optimal solution is a solution path of minimum cost
- There might be no solution !



How big is the state space of the (n^2-1)-puzzle?

- 8-puzzle → 9! = 362,880 states
- 15-puzzle → 16! ~ 2.09×10^{13} states
- 24-puzzle → 25! ~ 10^{25} states

But <u>only half</u> of these states are reachable from any given state
(but you may not know that in advance)

8-, 15-, 24-Puzzles

8-puzzle → 362,880 states

0.036 sec

15-puzzle → 2.09×10^{13} states

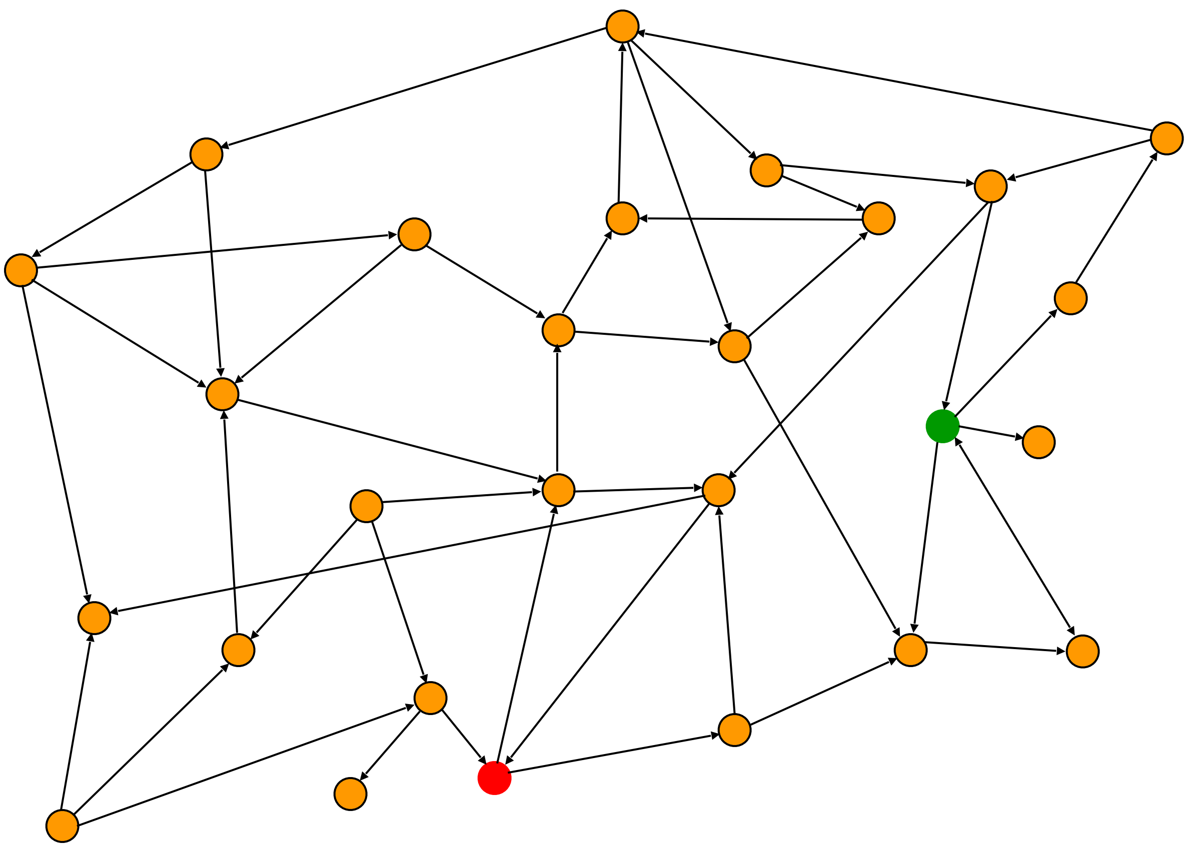
~ 55 hours

> 10^9 years

24-puzzle → 10^{25} states

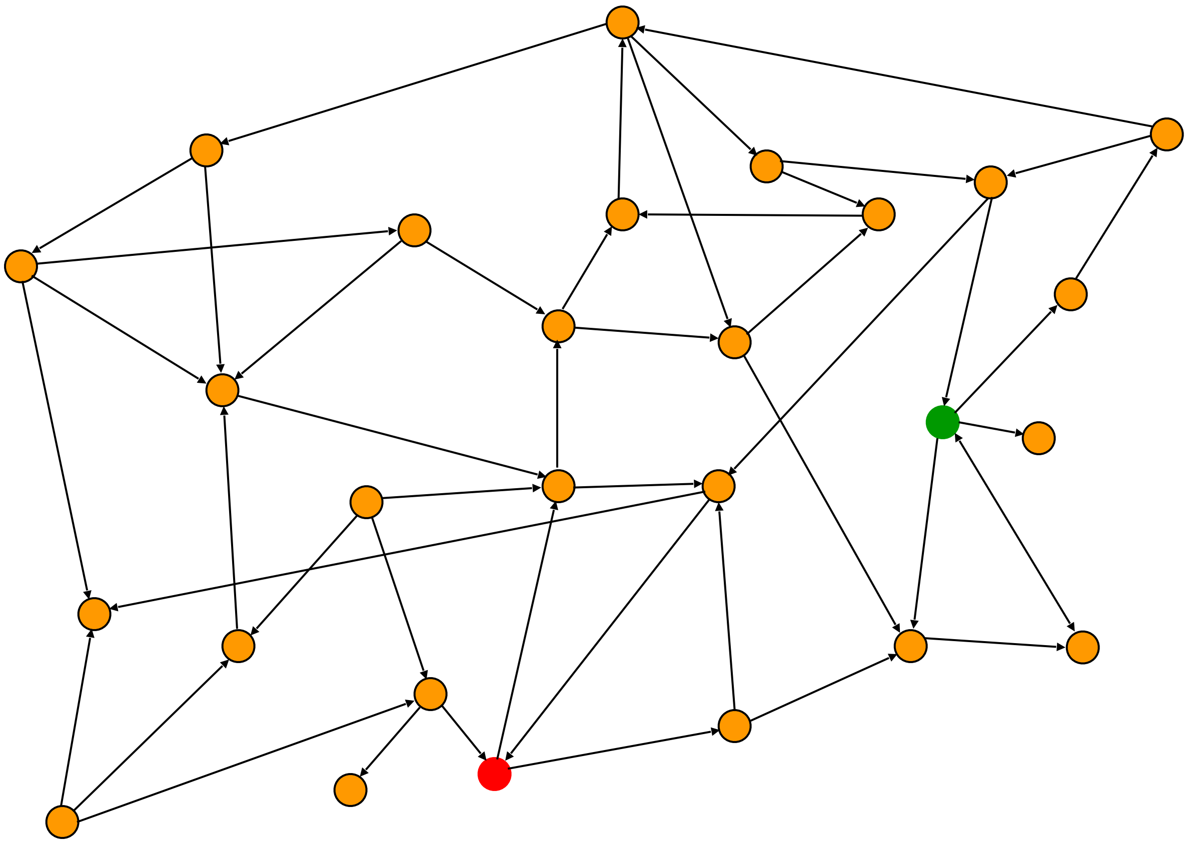
100 millions states/sec

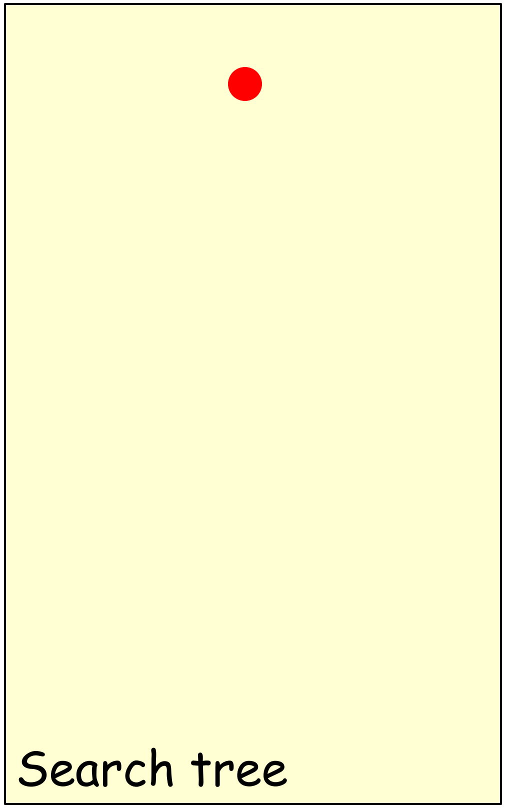
Searching the State Space



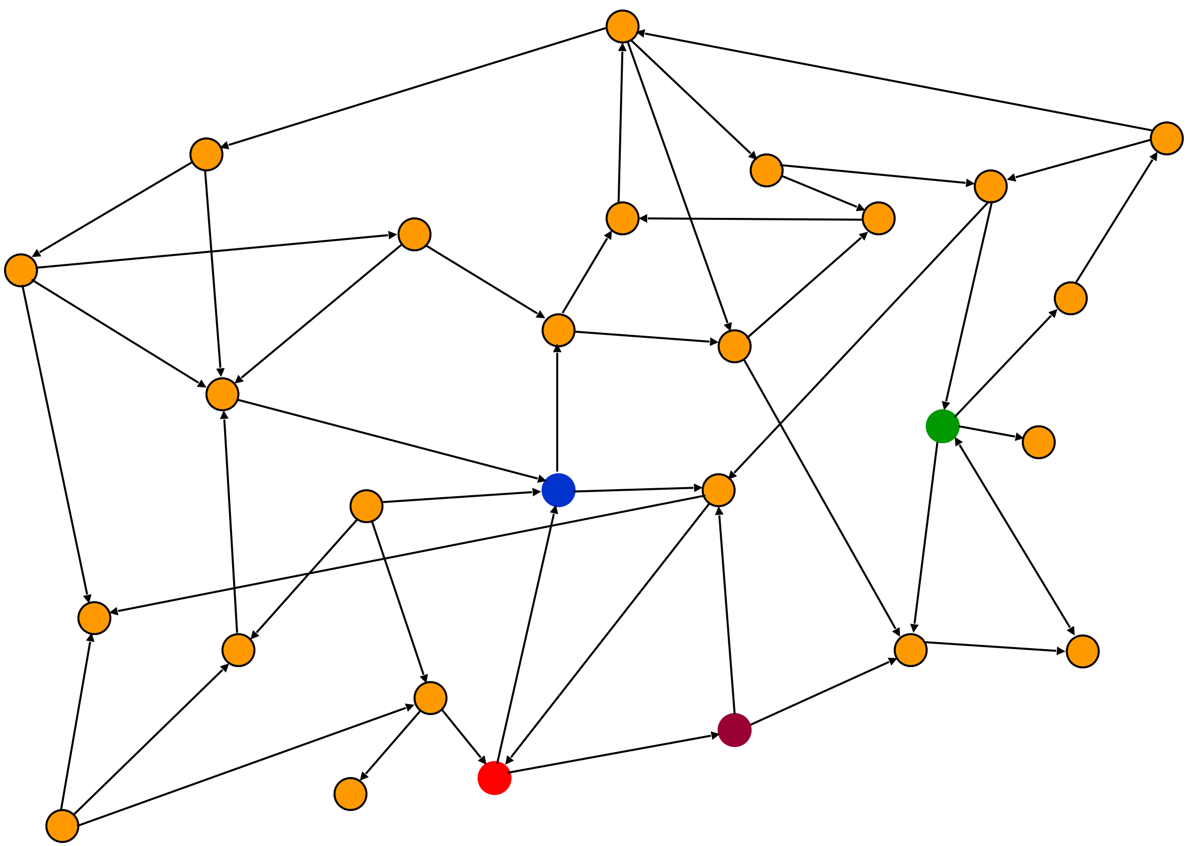
- Often it is not feasible (or too expensive) to build a complete representation of the state graph
- A problem solver must construct a solution by exploring a small portion of the graph

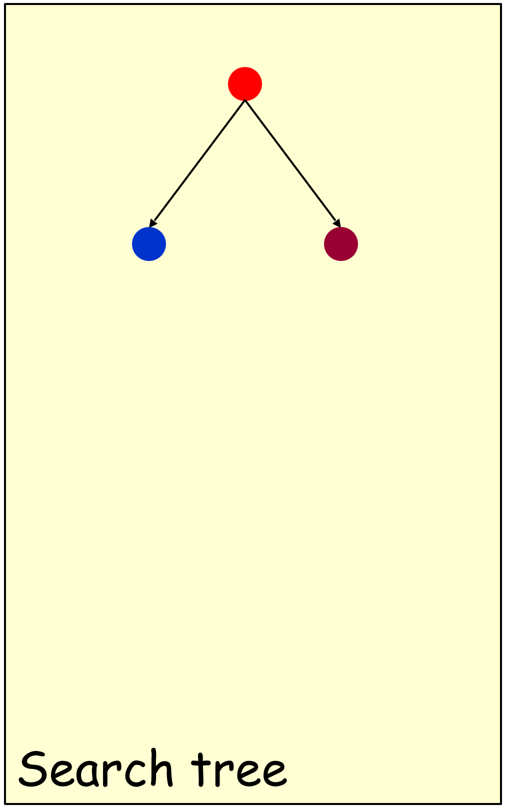
Searching the State Space



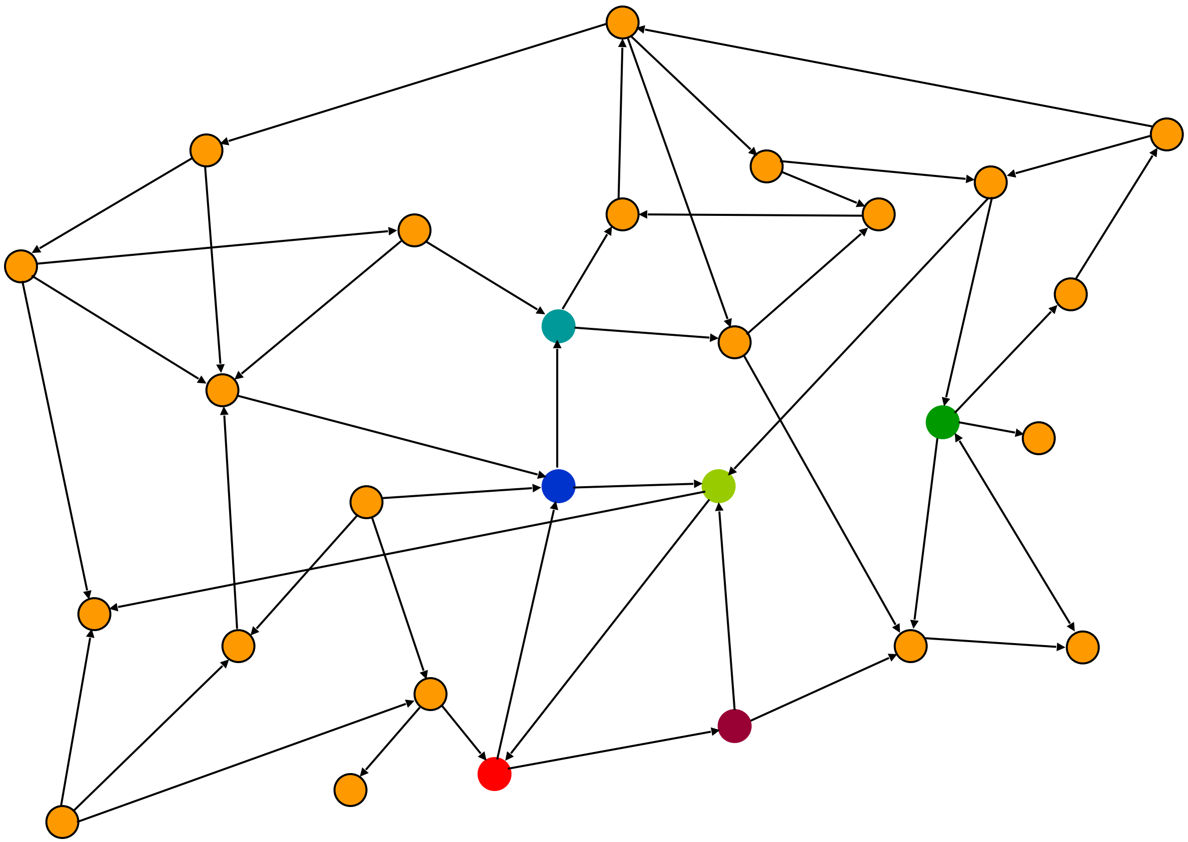


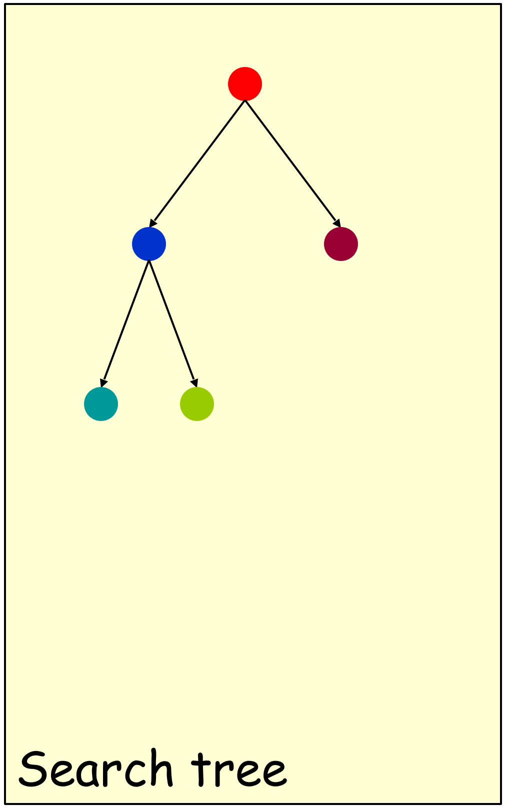
Searching the State Space



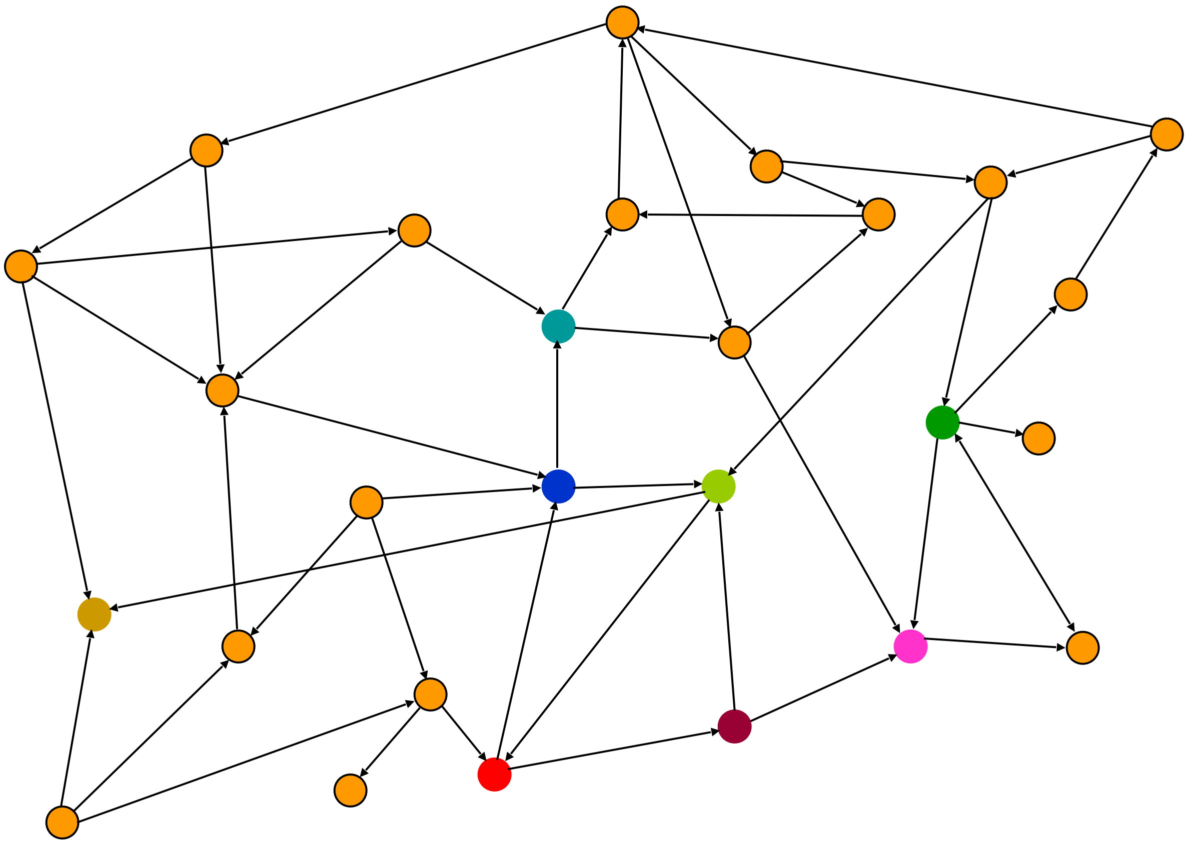


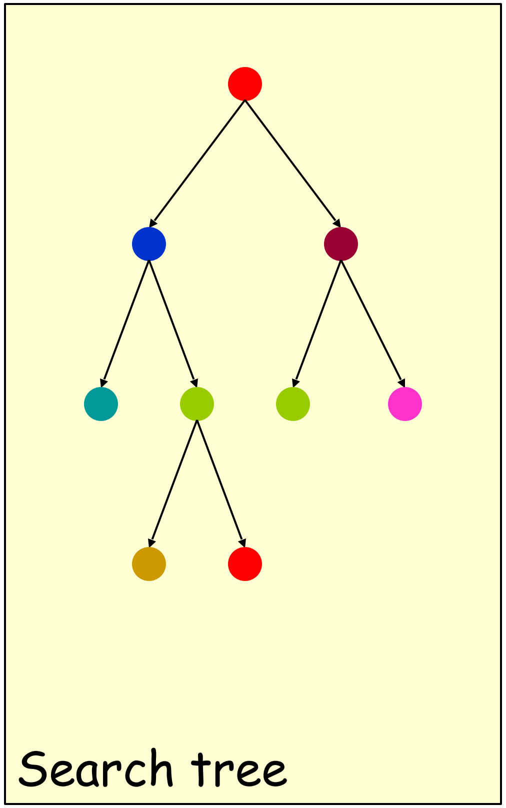
Searching the State Space



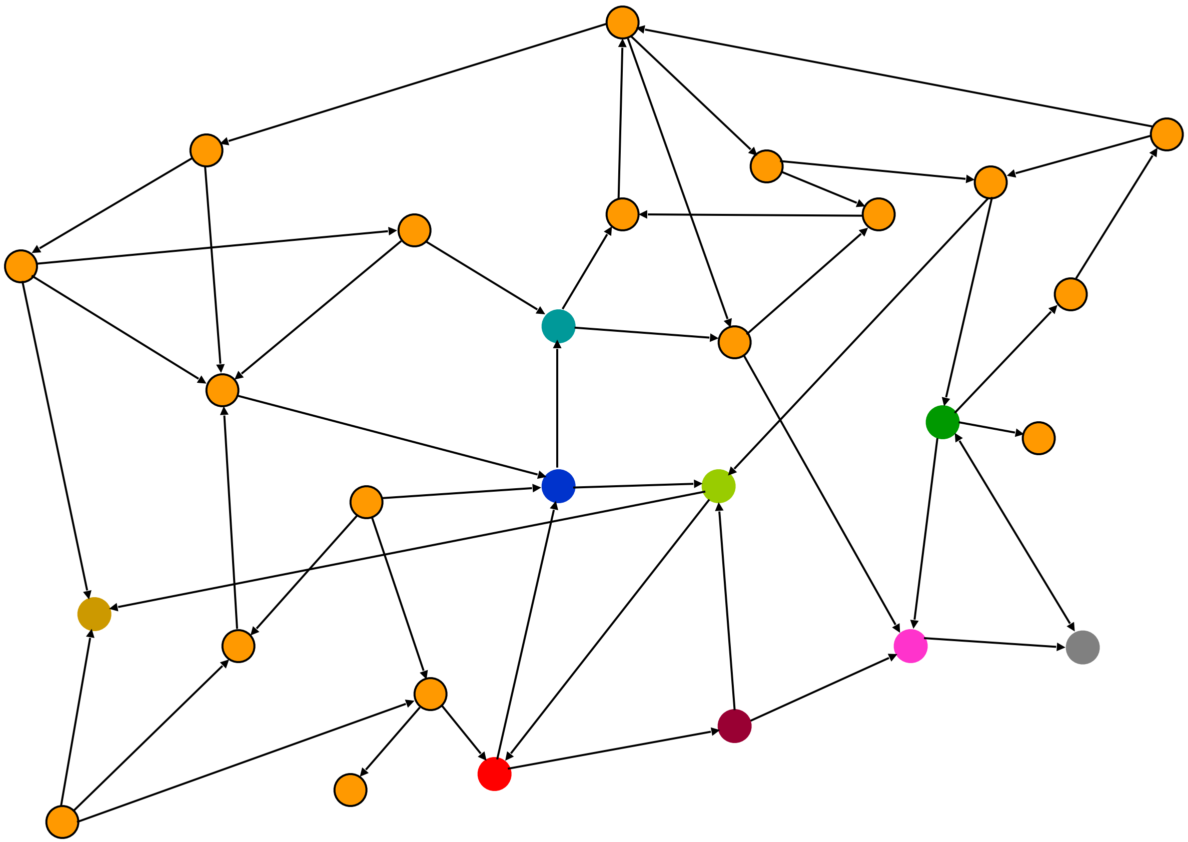


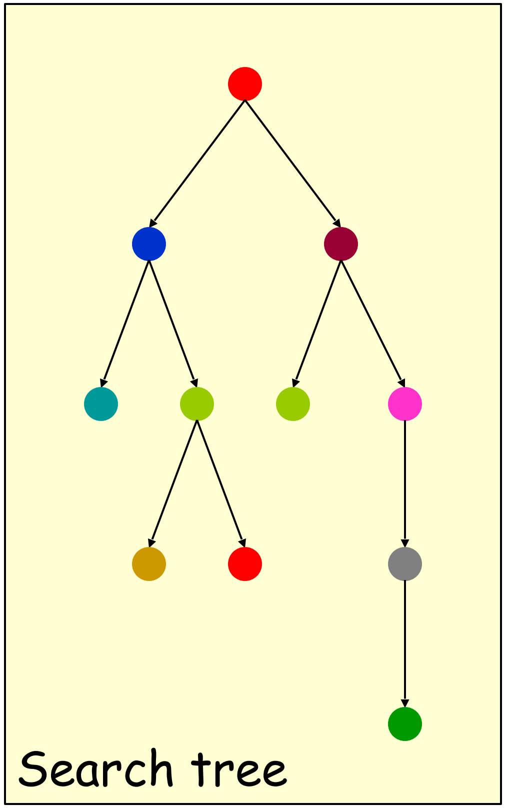
Searching the State Space



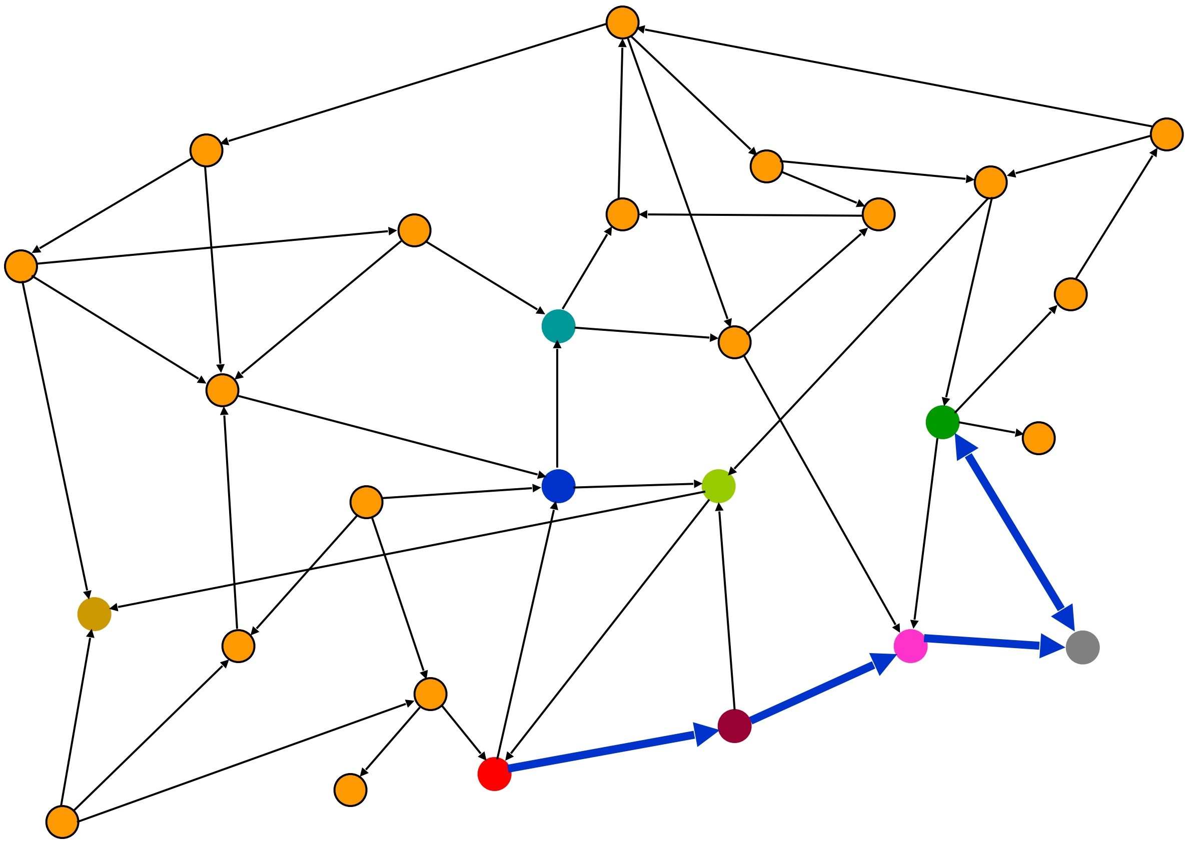


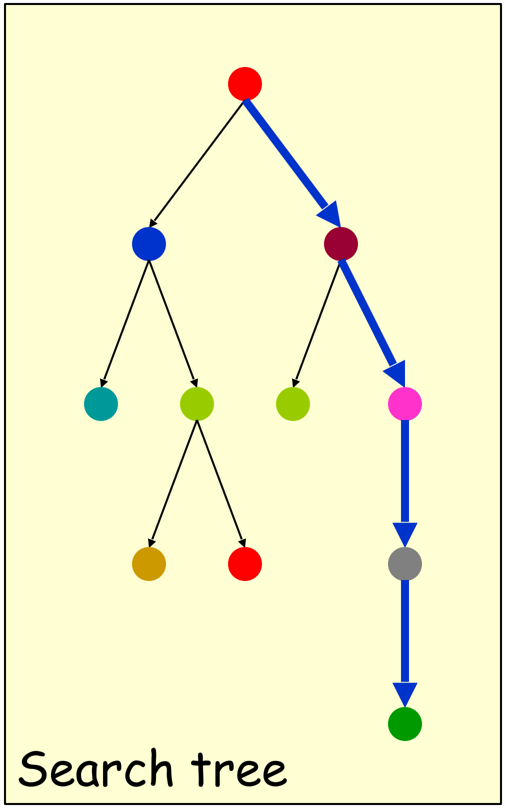
Searching the State Space





Searching the State Space

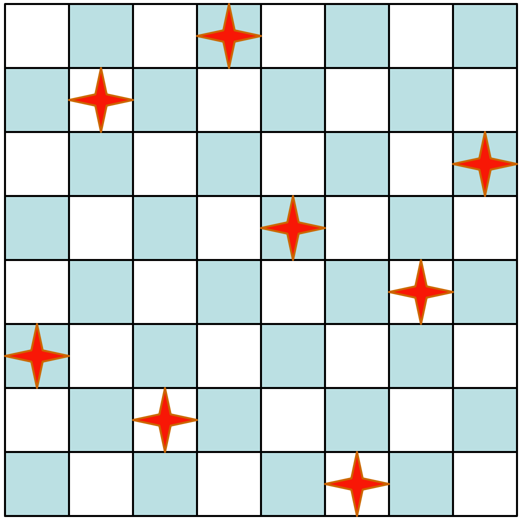




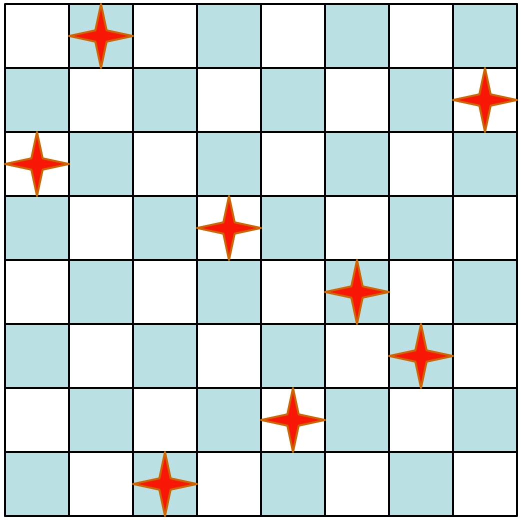
Other examples

8-Queens Problem

Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

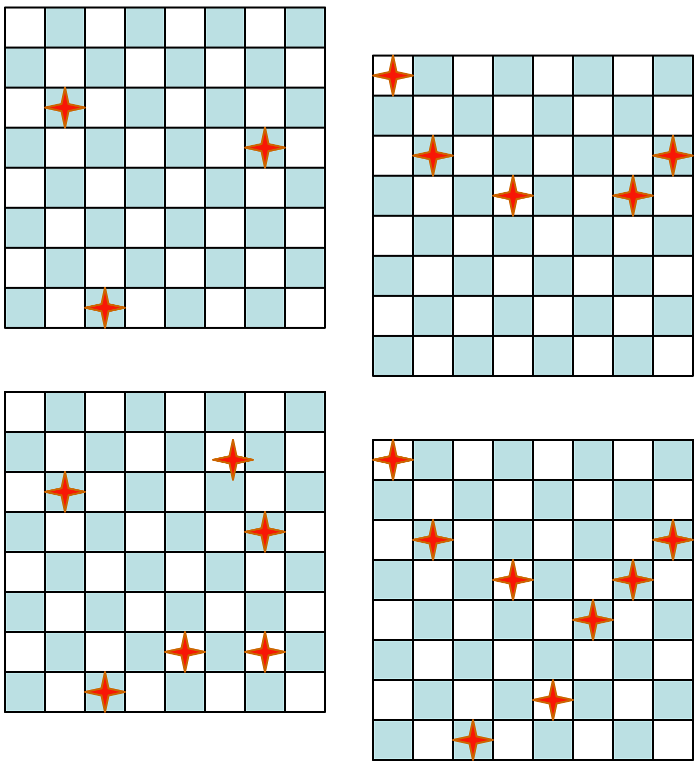


A solution



Not a solution

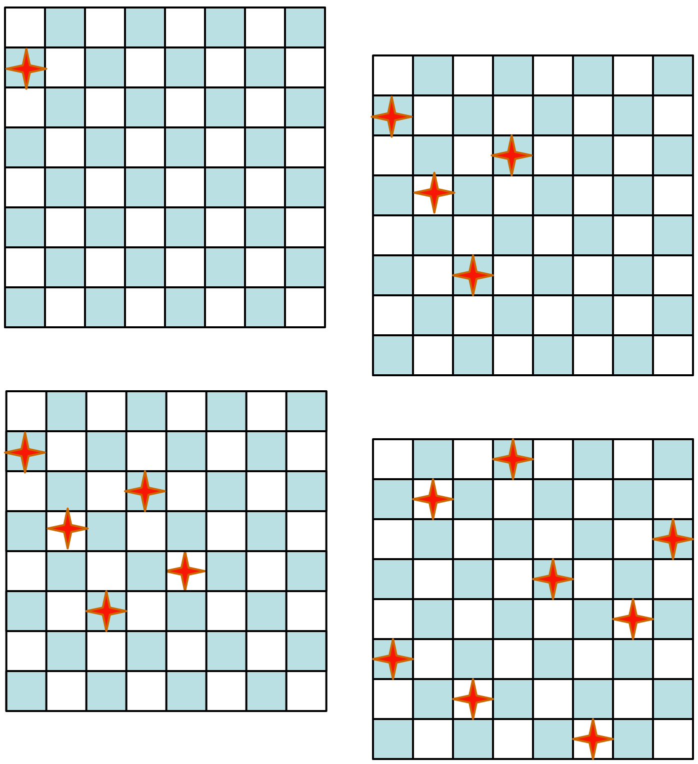
Formulation #1



- **States:** all arrangements of 0, 1, 2, ..., 8 queens on the board
- **Initial state:** 0 queens on the board
- **Successor function:** each of the successors is obtained by adding one queen in an empty square
- **Arc cost:** irrelevant
- **Goal test:** 8 queens are on the board, with no queens attacking each other

→ ~ 64x63x...x57 ~ 3×10^{14} states

Formulation #2



- **States:** all arrangements of k = 0, 1, 2, ..., 8 queens in the k leftmost columns with no two queens attacking each other
- **Initial state:** 0 queens on the board
- **Successor function:** each successor is obtained by adding one queen in any square that is not attacked by any queen already in the board, in the leftmost empty column
- **Arc cost:** irrelevant
- **Goal test:** 8 queens are on the board

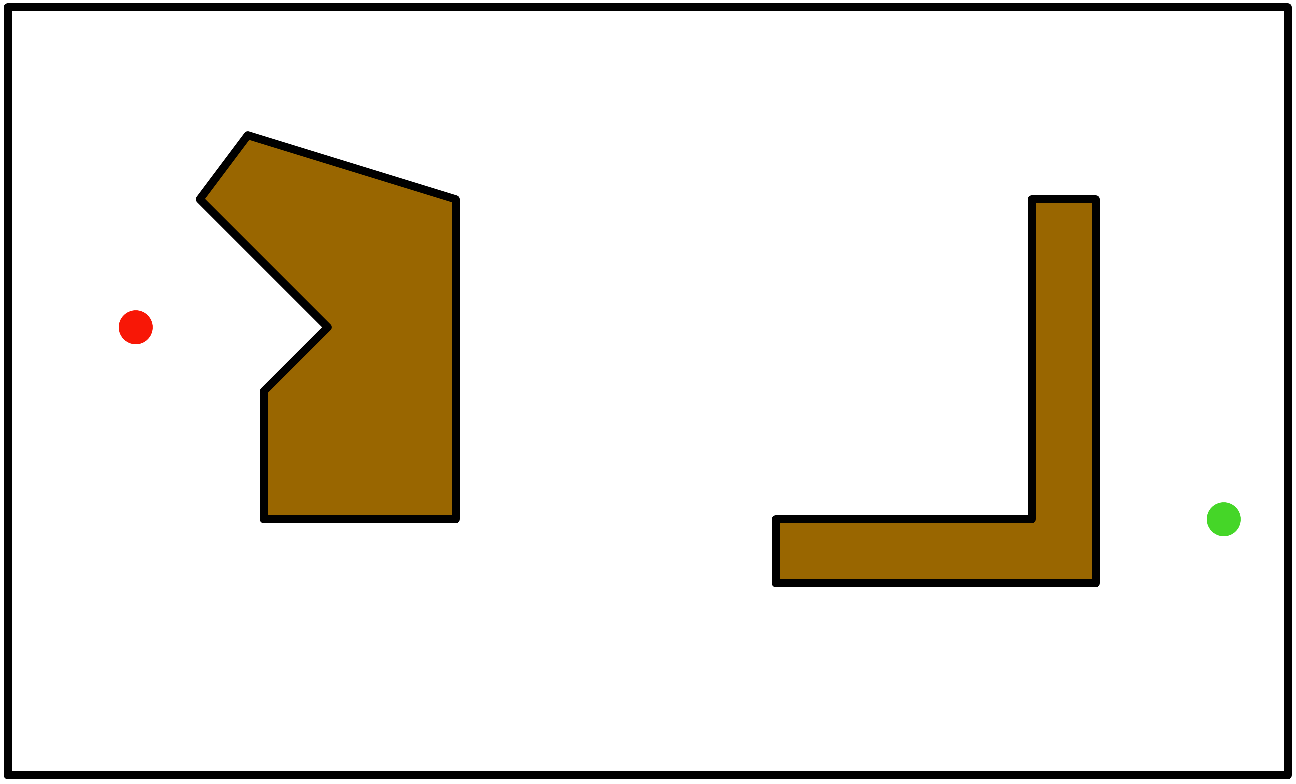
→ 2,057 states

n-Queens Problem

- A solution is a goal node, not a path to this node (typical of design problem)
- Number of states in state space:
  - 8-queens → 2,057
  - 100-queens → 10^{52}
- But techniques exist to solve n-queens problems efficiently for large values of n

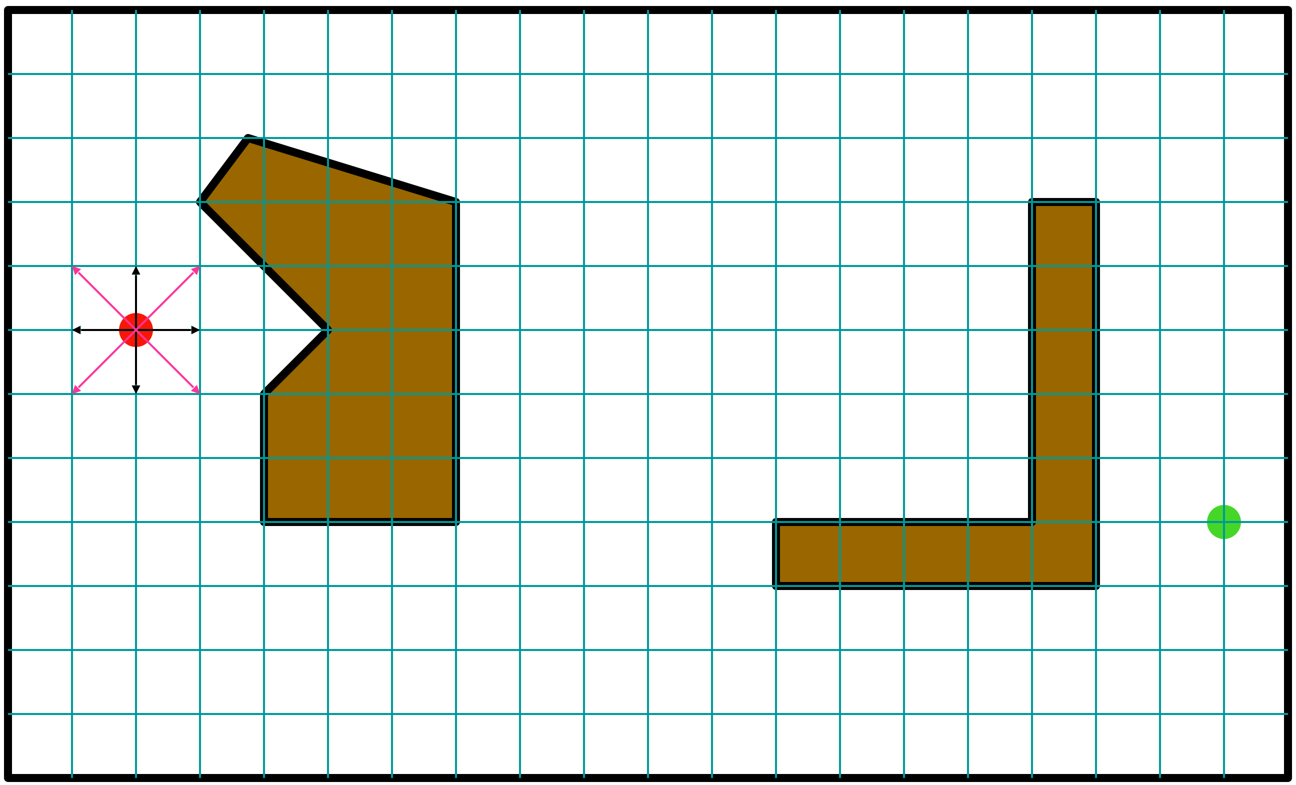
  They exploit the fact that there are many solutions well distributed in the state space

Path Planning



What is the state space?

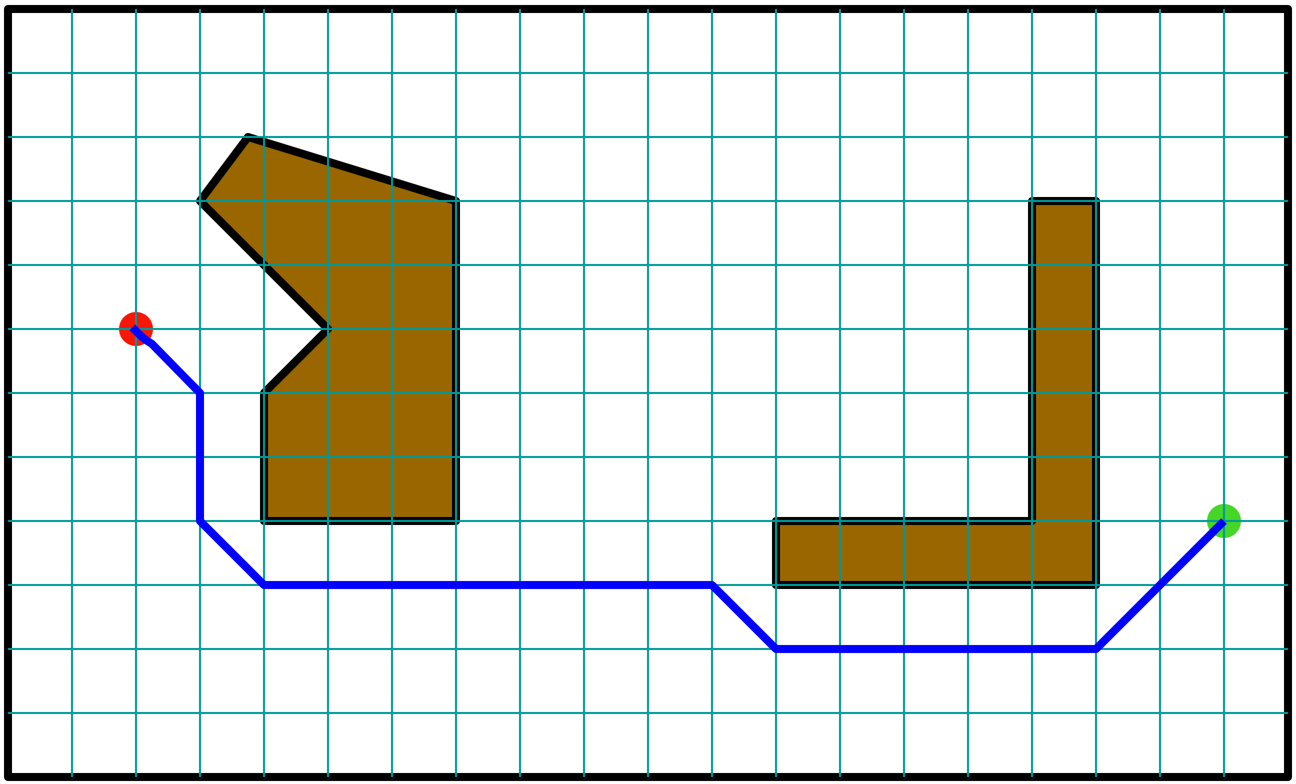
Formulation #1



Cost of one horizontal/vertical step = 1

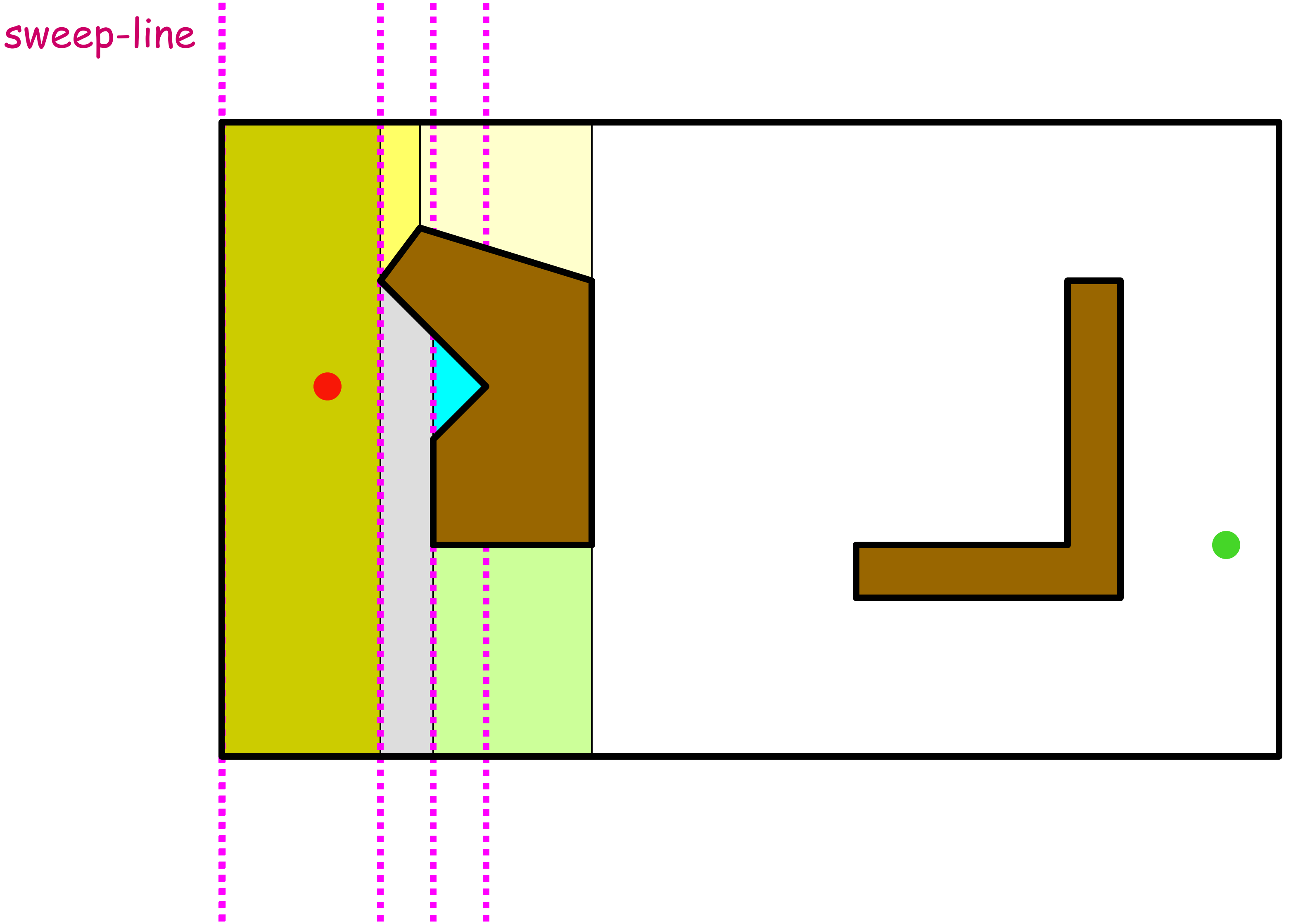
Cost of one diagonal step = √2

Optimal Solution

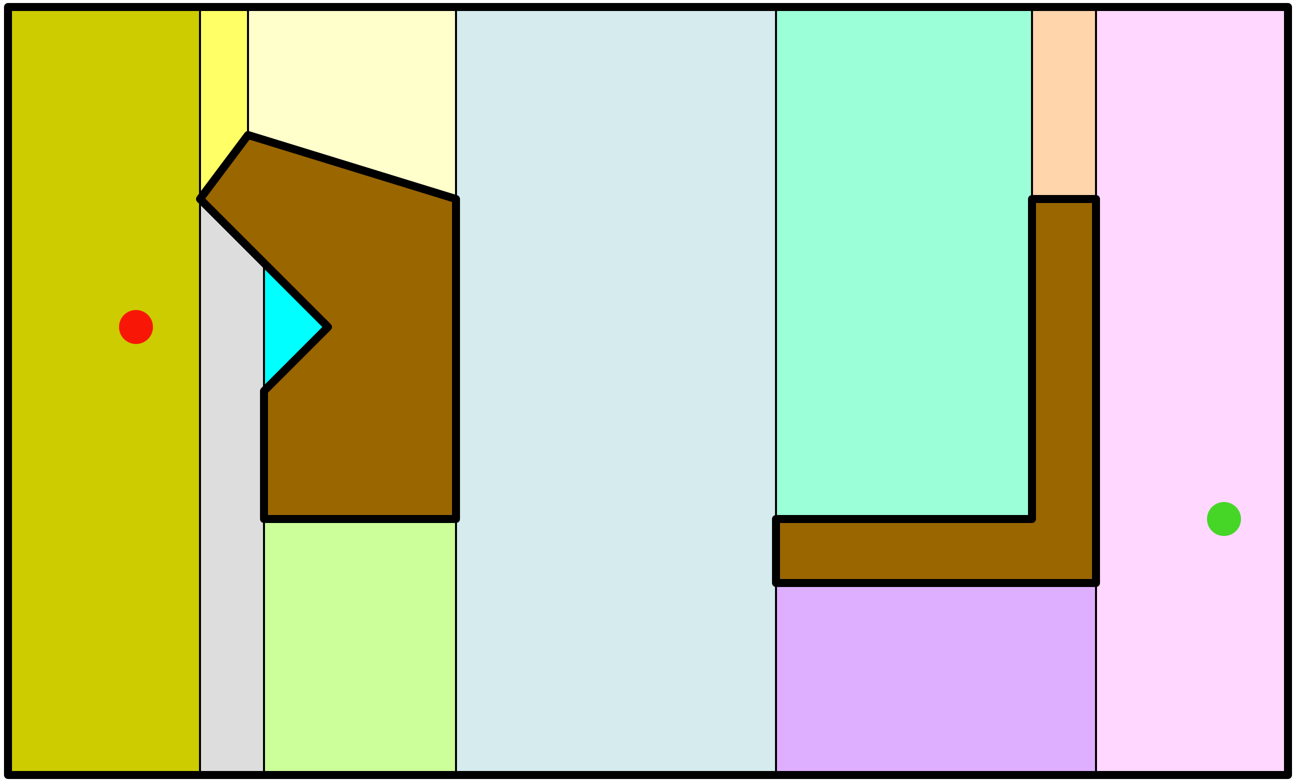


This path is the shortest in the discretized state space, but not in the original continuous space

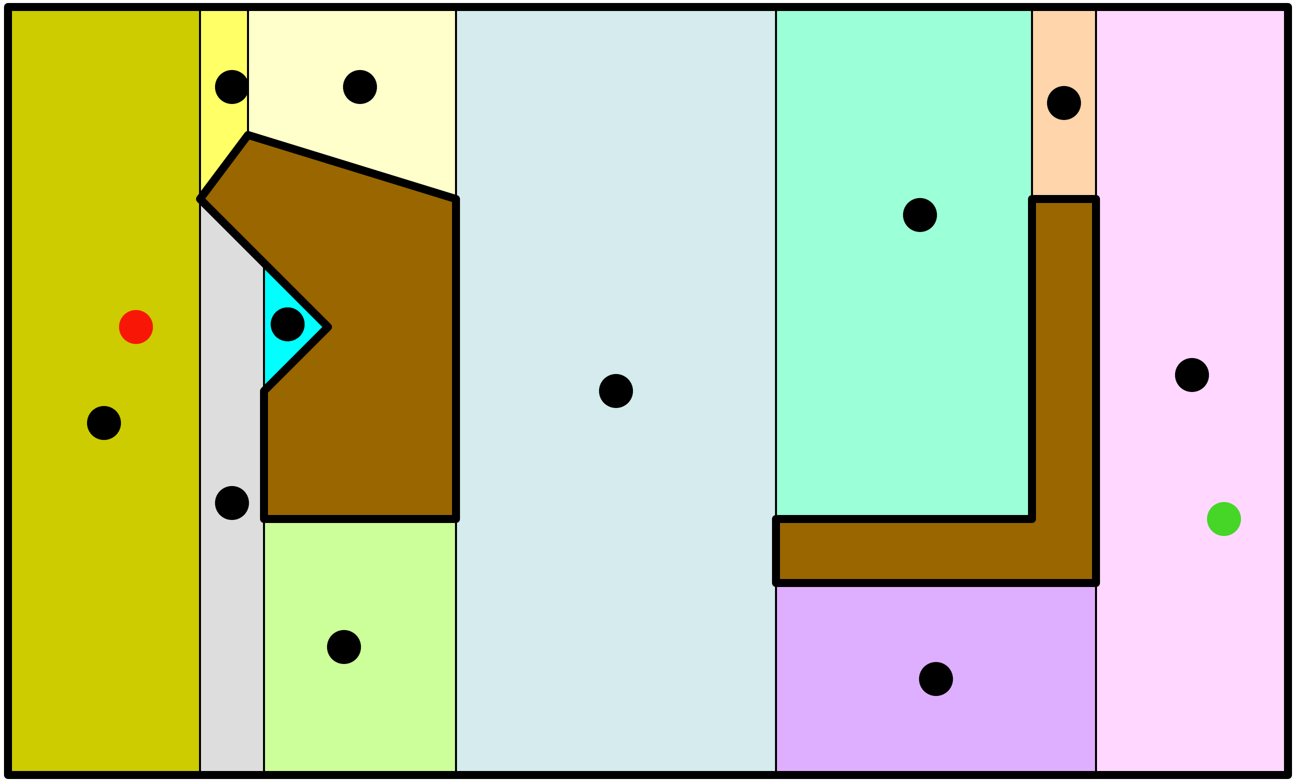
Formulation #2



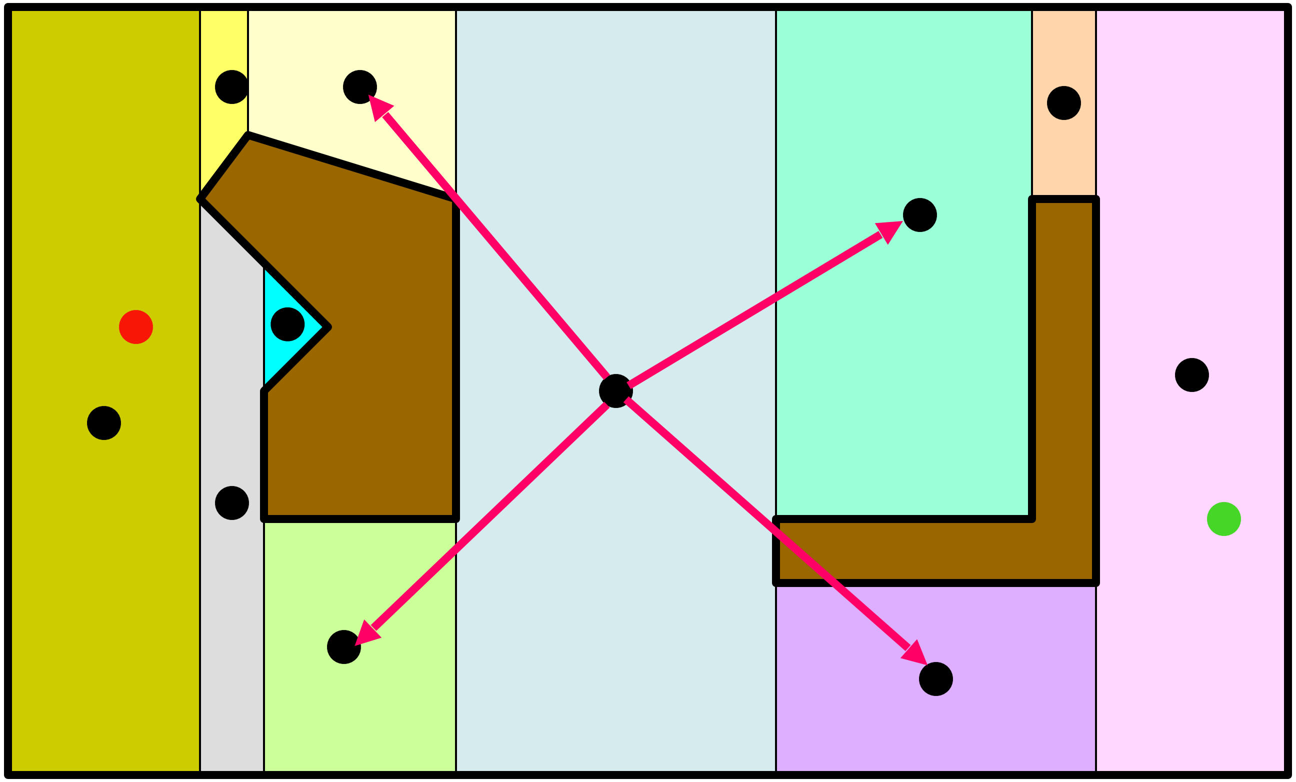
Formulation #2



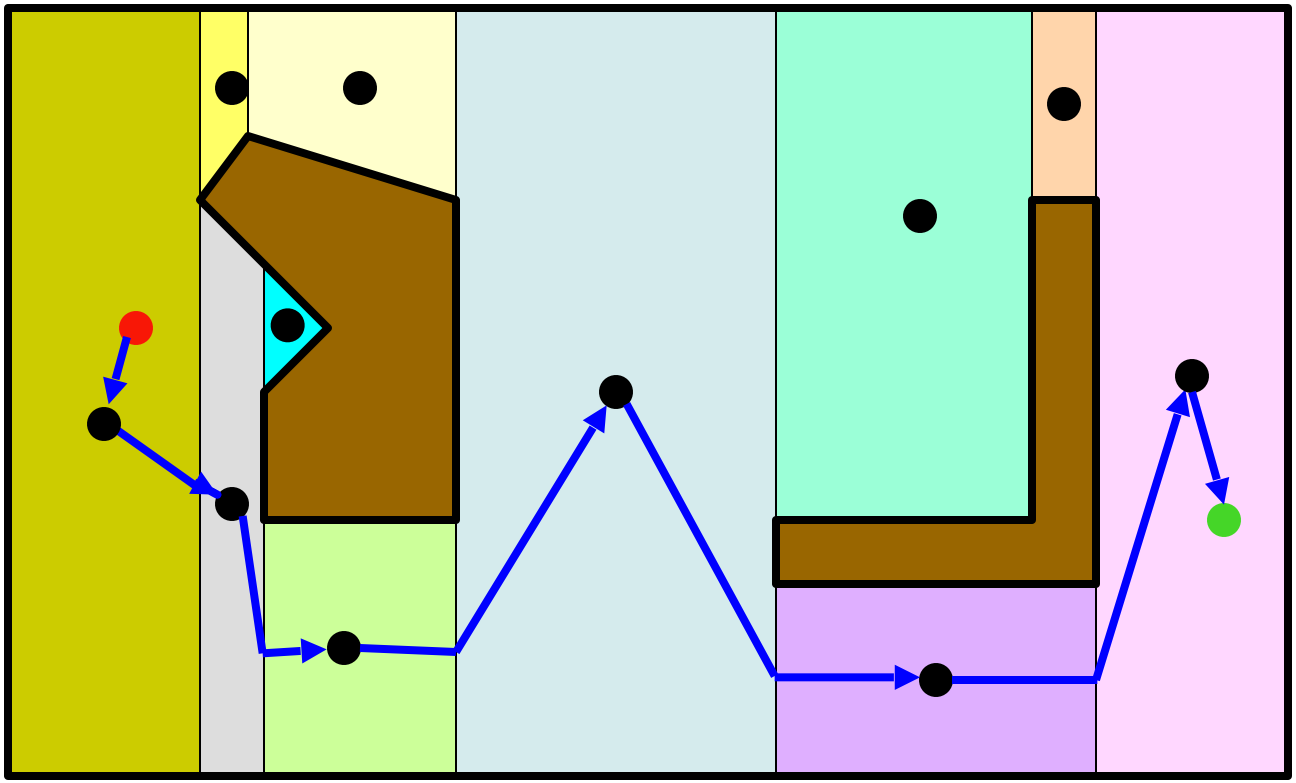
States



Successor Function

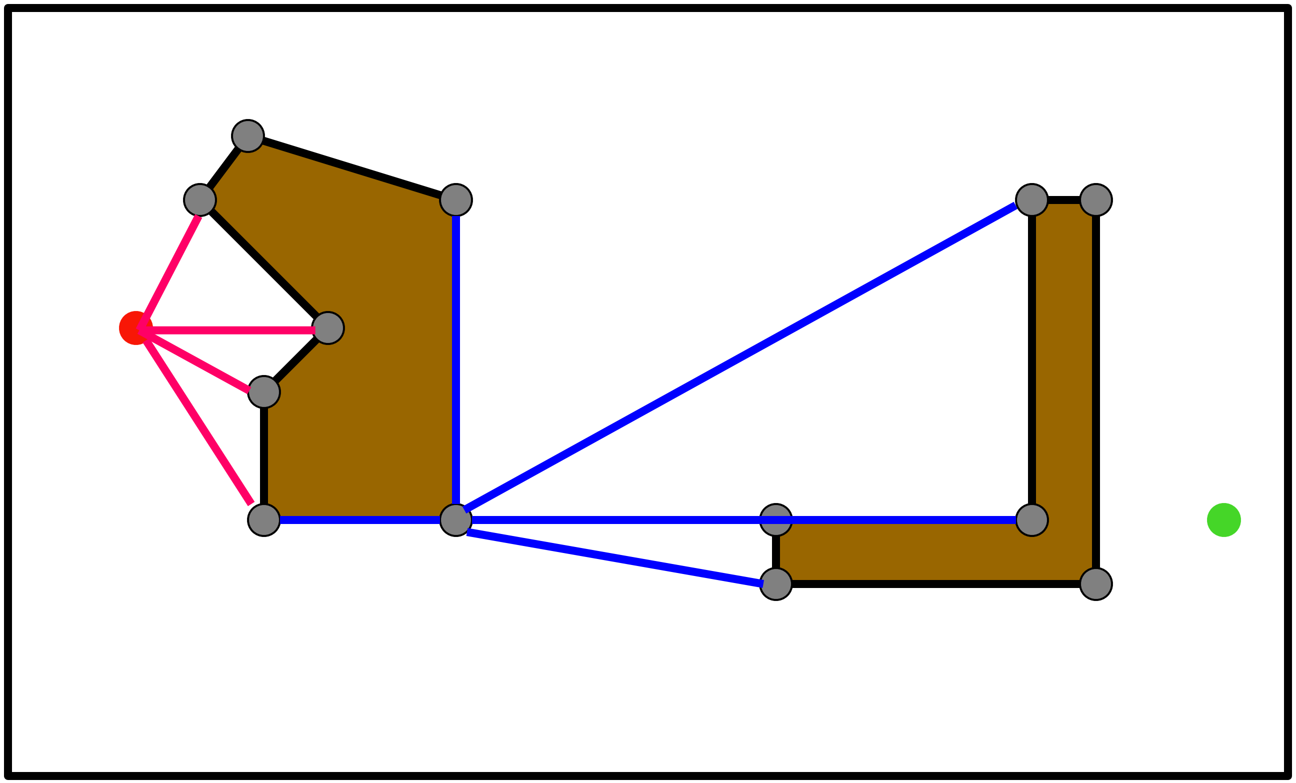


Solution Path



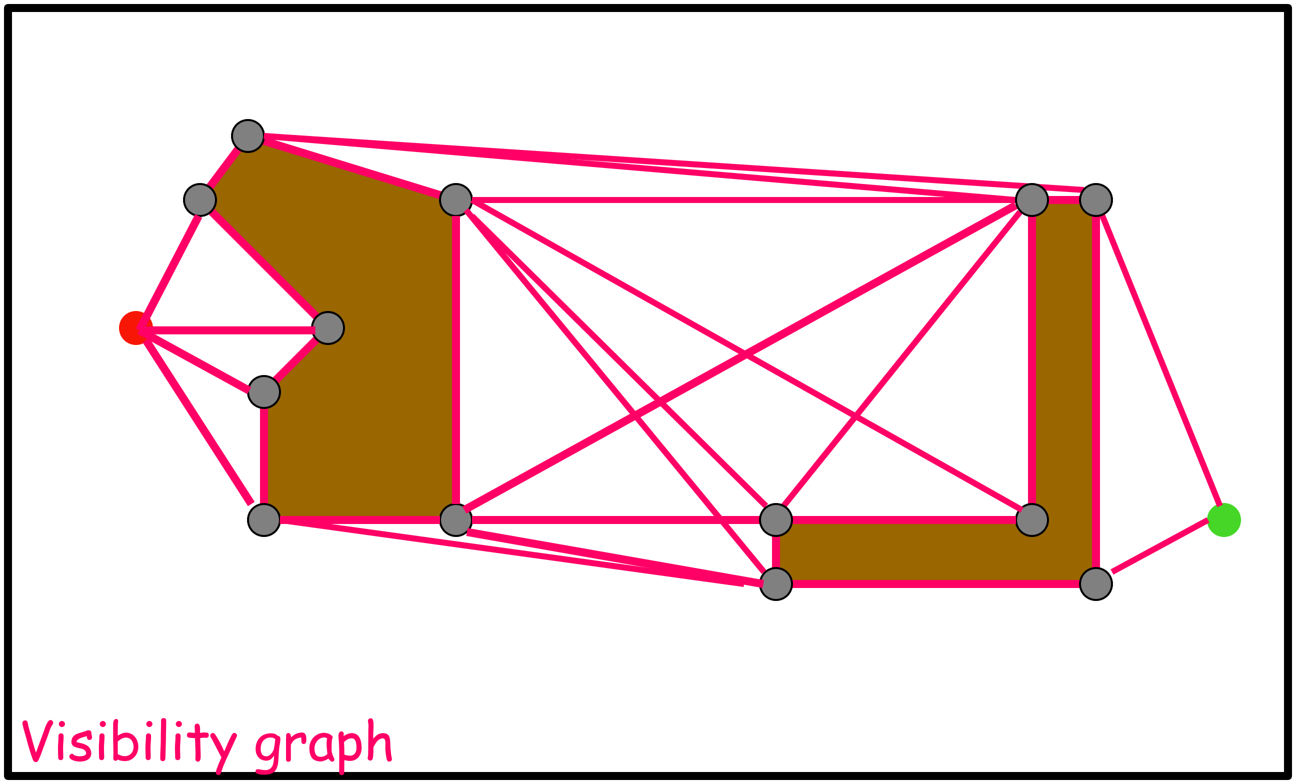
A path-smoothing post-processing step is usually needed to shorten the path further

Formulation #3



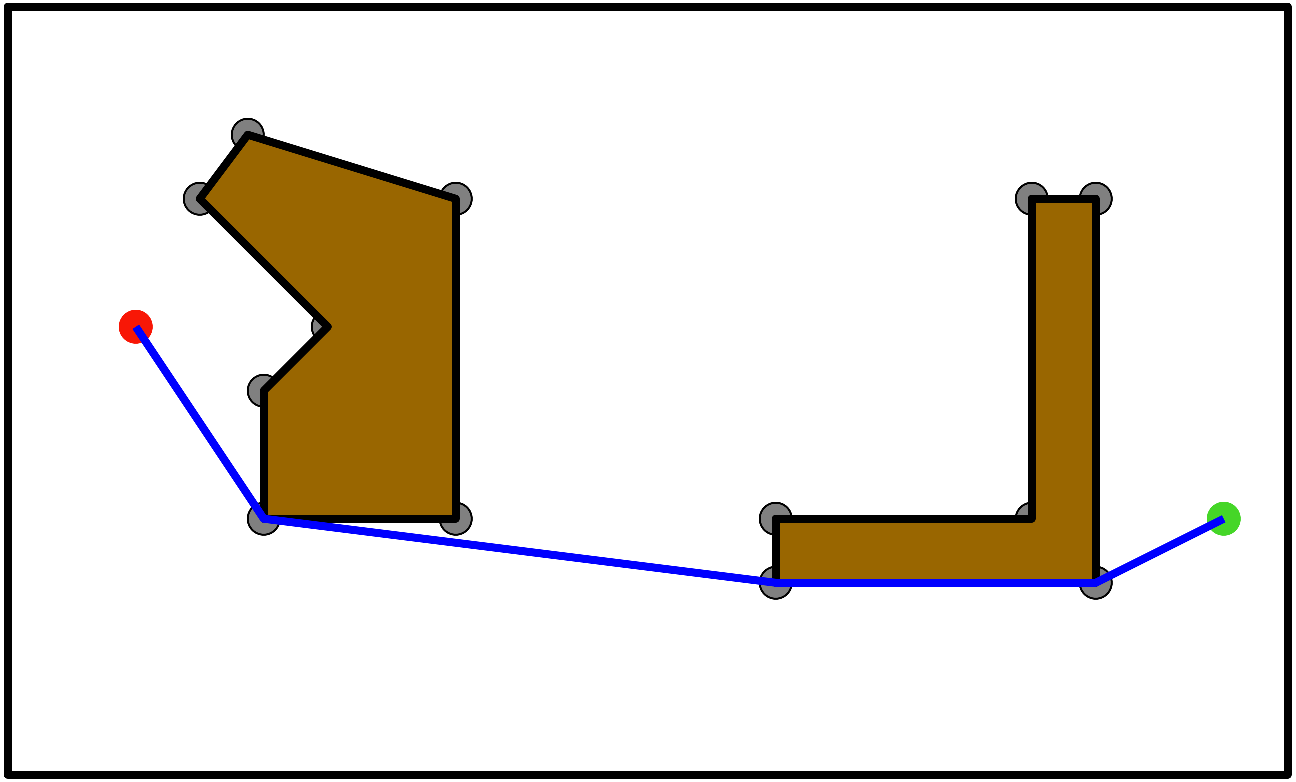
Cost of one step: length of segment

Formulation #3



Cost of one step: length of segment

Solution Path

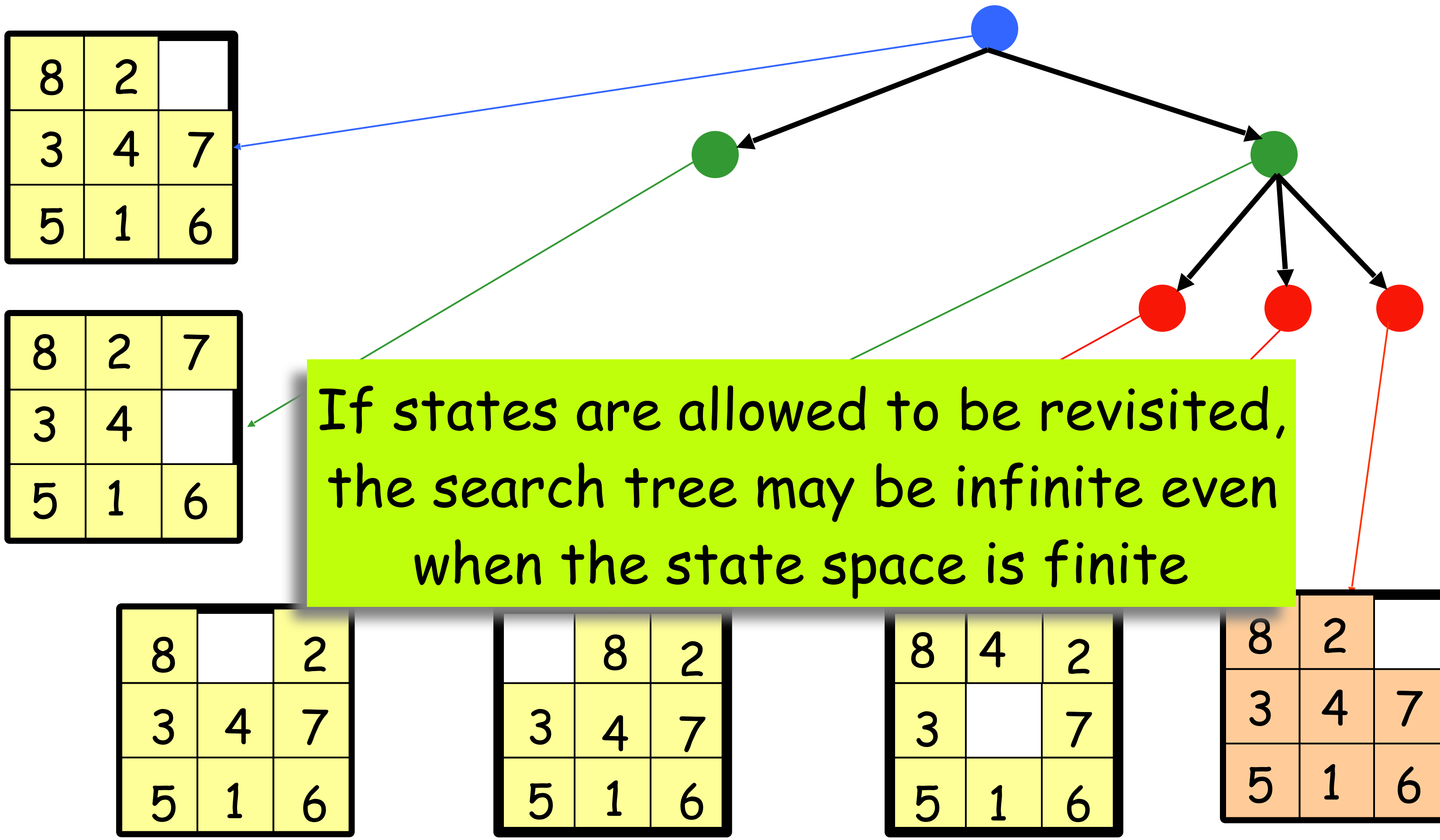


The shortest path in this state space is also the shortest in the original continuous space

Simple Problem-Solving-Agent

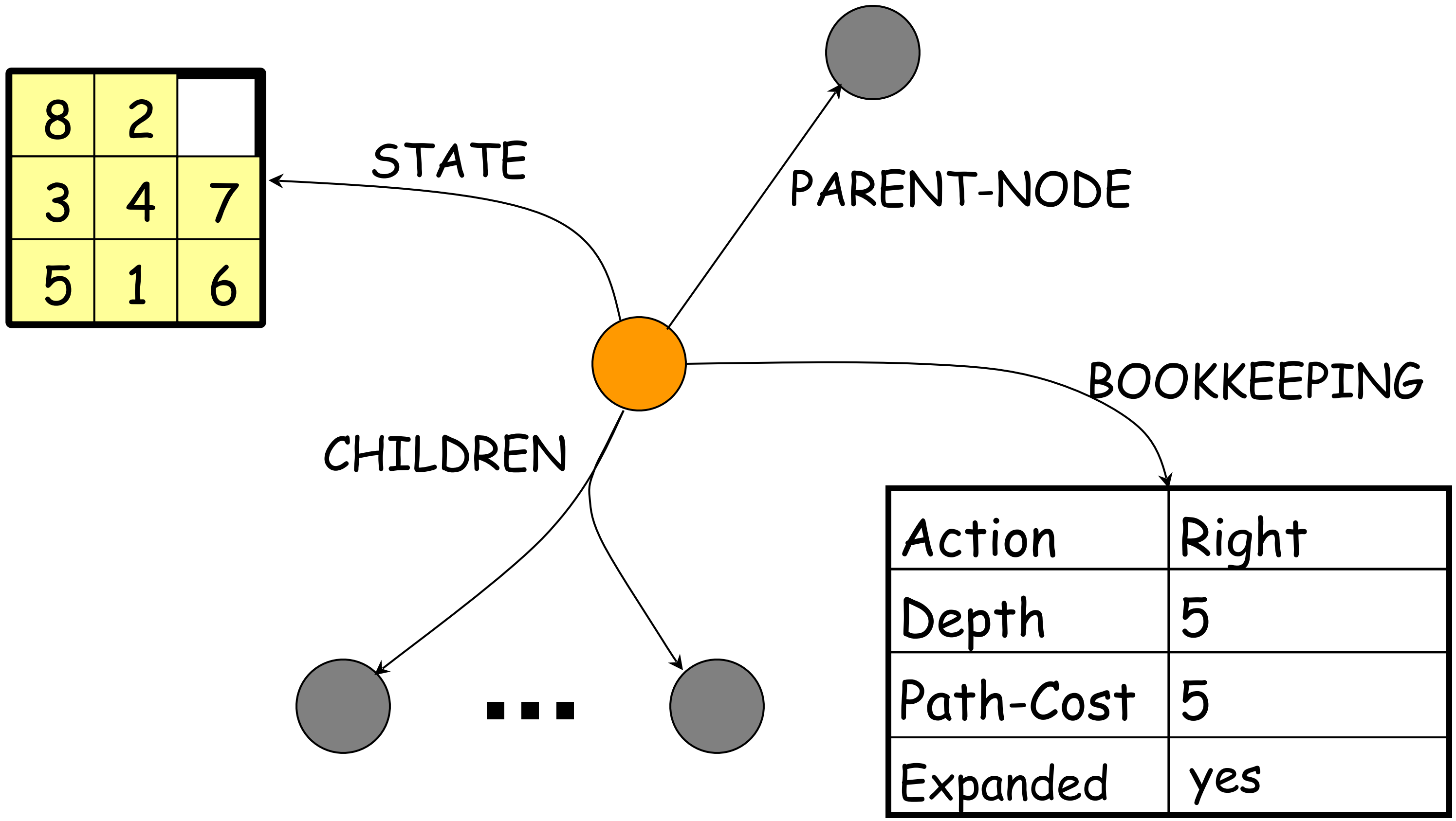
1. s_0 ← sense/read initial state
2. GOAL? ← select/read goal test
3. Succ ← read successor function
4. solution ← **search**(s_0, GOAL?, Succ)
5. perform(solution)

Search Nodes and States



If states are allowed to be revisited, the search tree may be infinite even when the state space is finite

Data Structure of a Node



Depth of a node N
= length of path from root to N

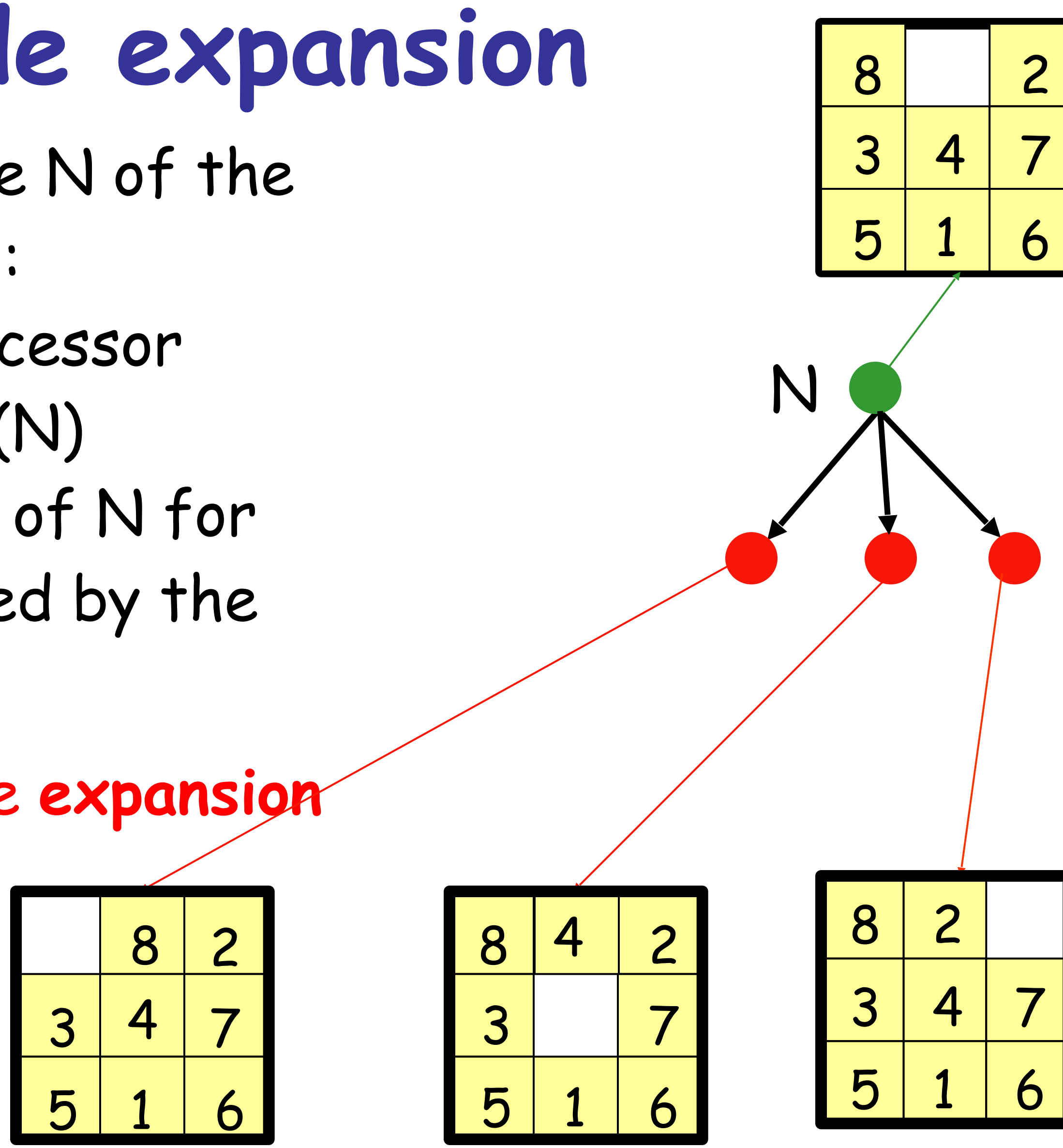
(depth of the root = 0)

Node expansion

The expansion of a node N of the search tree consists of:

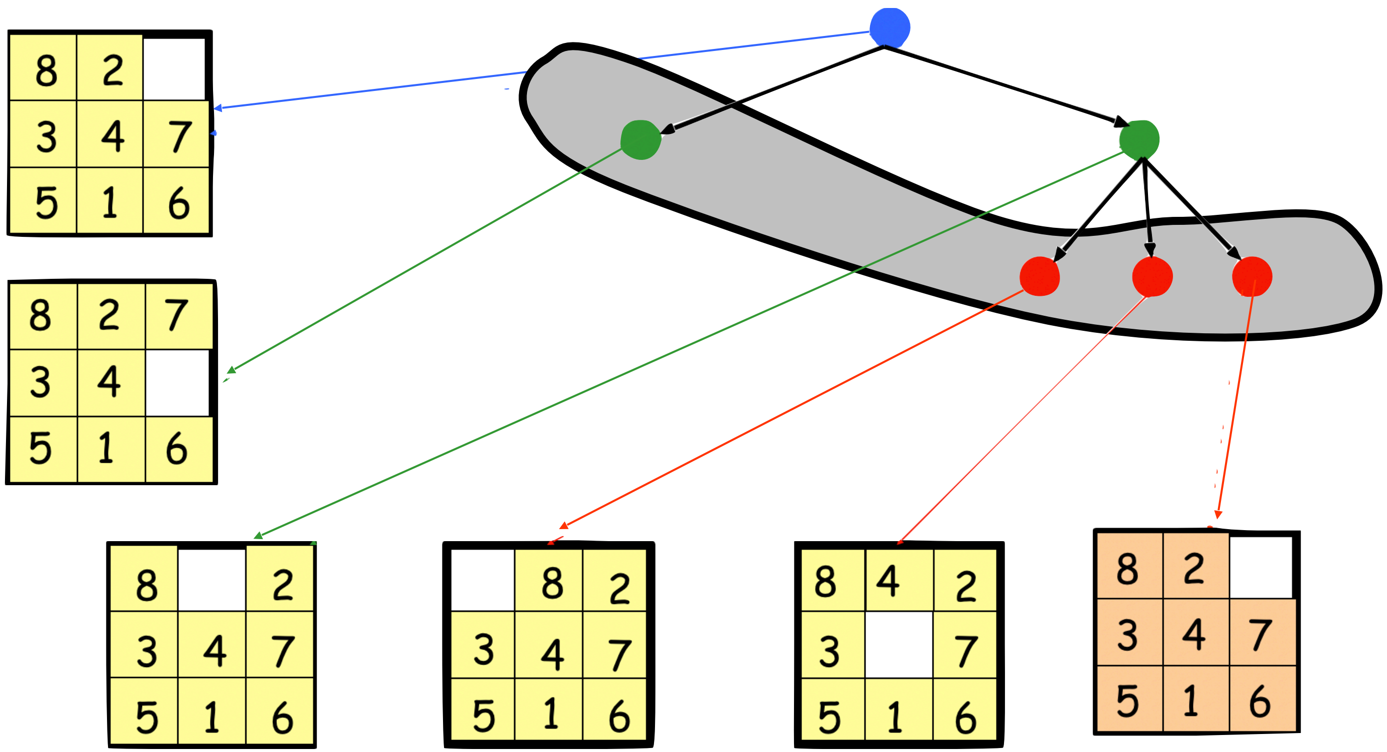
1) Evaluating the successor function on STATE(N)
2) Generating a child of N for each state returned by the function

node generation ≠ node expansion



Open-List of Search Tree

- The Open-List is the set of all search nodes that haven't been expanded yet



Search Strategy

- The Open-List is the set of all search nodes that haven't been expanded yet
- The Open-List is implemented as a priority queue
  - INSERT(node,Open-List)
  - REMOVE(Open-List)
- The ordering of the nodes in Open-List defines the search strategy

Search Algorithm #1

SEARCH#1

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,Open-List)
3. Repeat:
   a. If empty(Open-List) then return failure
   b. N ← REMOVE(Open-List)
   c. s ← STATE(N)
   d. For every state s' in SUCCESSORS(s)
      i. Create a new node N' as a child of N
      ii. If GOAL?(s') then return path or goal state
      iii. INSERT(N',Open-List)

Expansion of N

Performance Measures

- Completeness
  A search algorithm is complete if it finds a solution whenever one exists
  [What about the case when no solution exists?]
- Optimality
  A search algorithm is optimal if it returns a minimum-cost path whenever a solution exists
- Complexity
  It measures the time and amount of memory required by the algorithm

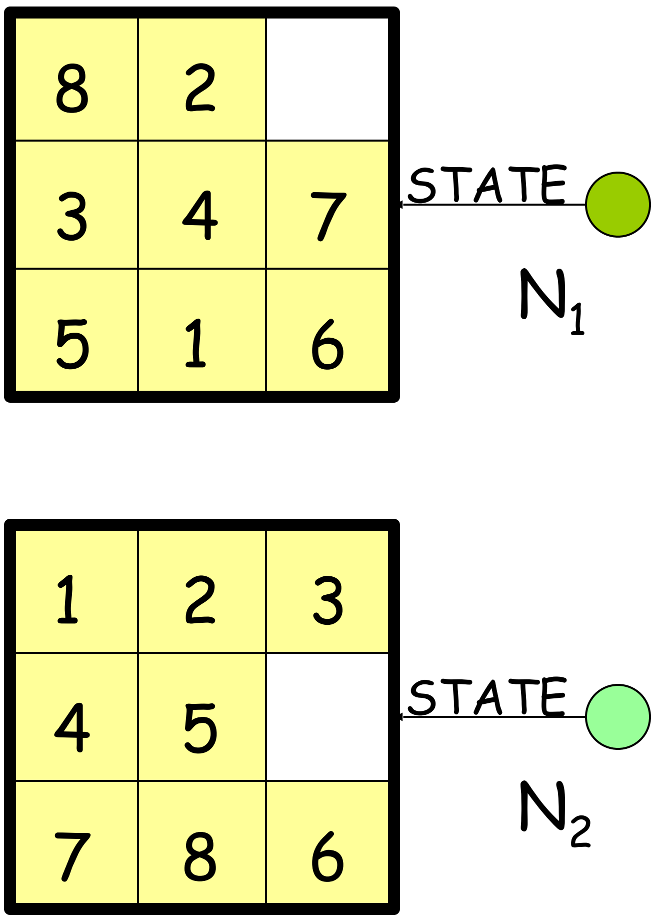
Blind vs. Heuristic Strategies

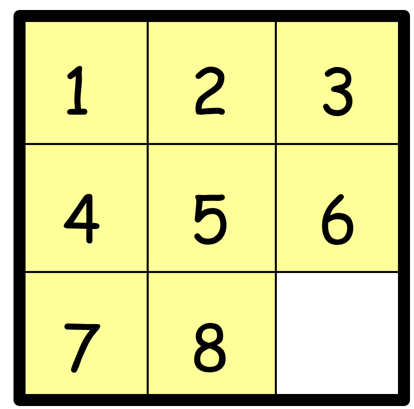
- Blind (or un-informed) strategies do not exploit state descriptions to order Open-List. They only exploit the positions of the nodes in the search tree

- Heuristic (or informed) strategies exploit state descriptions to order Open-List (the most "promising" nodes are placed at the beginning of Open-List)

Example

For a blind strategy, N_1 and N_2 are just two nodes (at some position in the search tree)

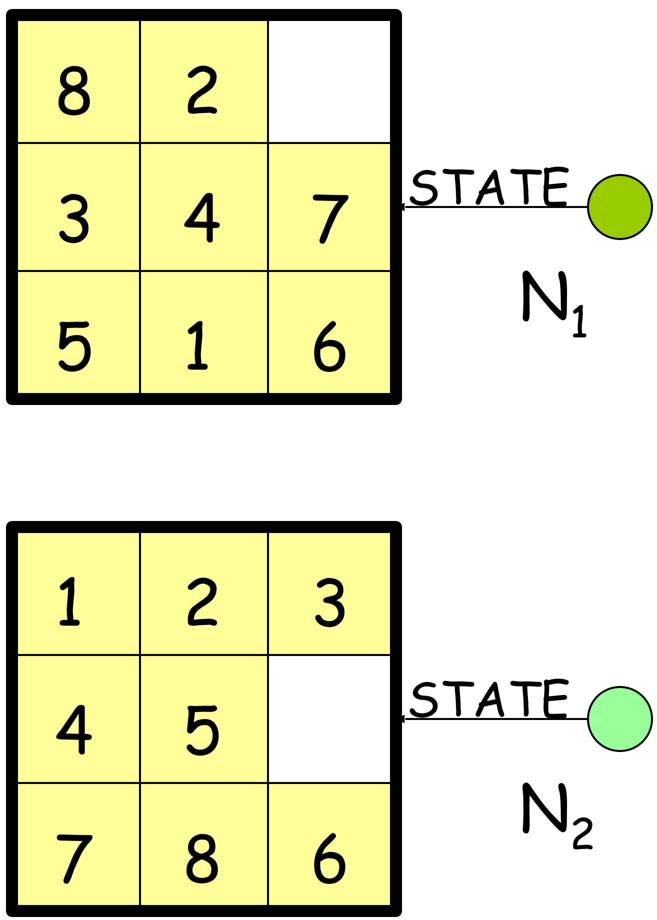


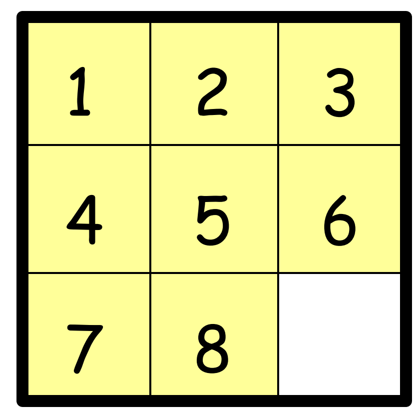


Goal state

Example

For a heuristic strategy counting the number of misplaced tiles, N_2 is more promising than N_1





Goal state

Remark

- Some search problems, such as the (n^2-1)-puzzle, are NP-hard
- One can't expect to solve all instances of such problems in less than exponential time (in n)
- One may still strive to solve each instance as efficiently as possible

→ This is the purpose of the search strategy

Blind Strategies

- Breadth-first
  - Bidirectional
- Depth-first
  - Depth-limited
  - Iterative deepening

} Arc cost = 1

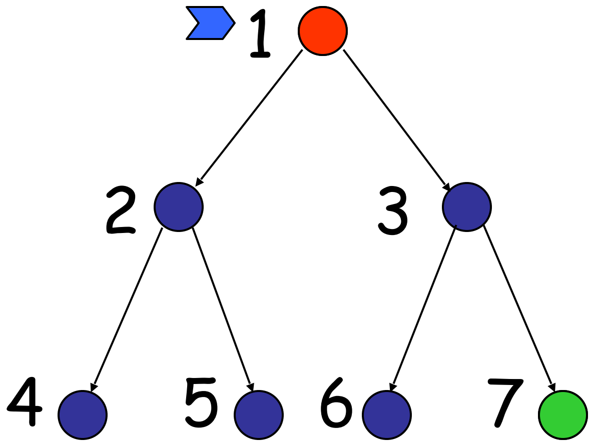
- Uniform-Cost

(variant of breadth-first)

} Arc cost = c(action) $\geq \varepsilon > 0$

Breadth-First Strategy

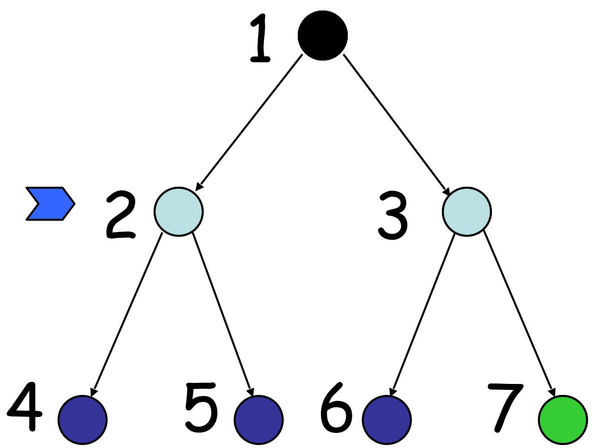
New nodes are inserted at the end of Open-List



Open-List = (1)

Breadth-First Strategy

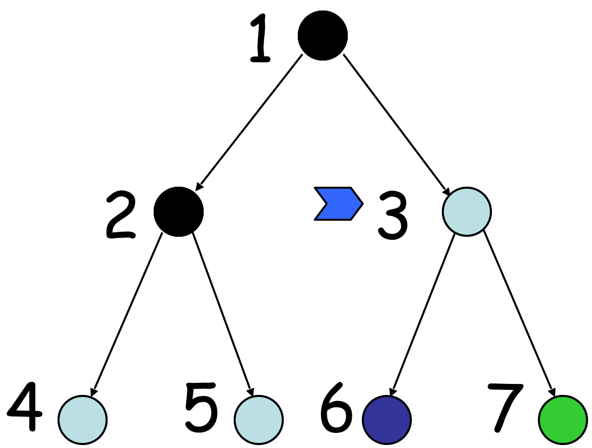
New nodes are inserted at the end of Open-List



Open-List = (2, 3)

Breadth-First Strategy

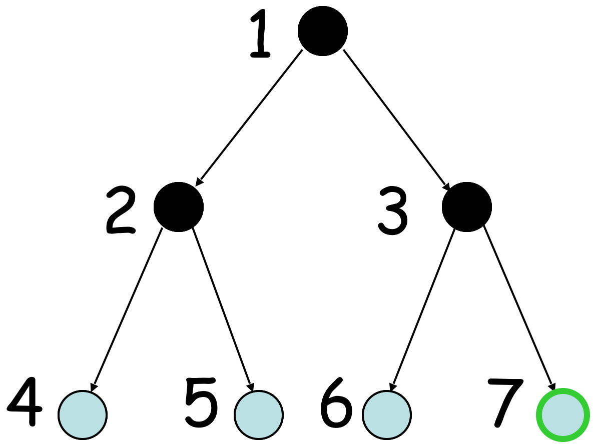
New nodes are inserted at the end of Open-List



Open-List = (3, 4, 5)

Breadth-First Strategy

New nodes are inserted at the end of Open-List



Open-List = (4, 5, 6, 7)

Important Parameters

1) Maximum number of successors of any state

   → branching factor b of the search tree

2) Minimal length (≠ cost) of a path between the initial and a goal state

   → depth d of the shallowest goal node in the search tree

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete? Not complete?
  - Optimal? Not optimal?

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:

  ???

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:

  $1 + b + b^2 + \ldots + b^d$ = ???

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated:

  $1 + b + b^2 + \ldots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$
- → Time and space complexity is $O(b^d)$

Big O Notation

$g(n) = O(f(n))$ if there exist two positive constants a and N such that:

for all $n > N$: $g(n) \leq a \times f(n)$

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/node

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^6$	1 sec	100 Mb
8	$\sim 10^8$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/node

Remark

If a problem has no solution, breadth-first may run for ever (if the state space is infinite or states can be revisited arbitrary many times)

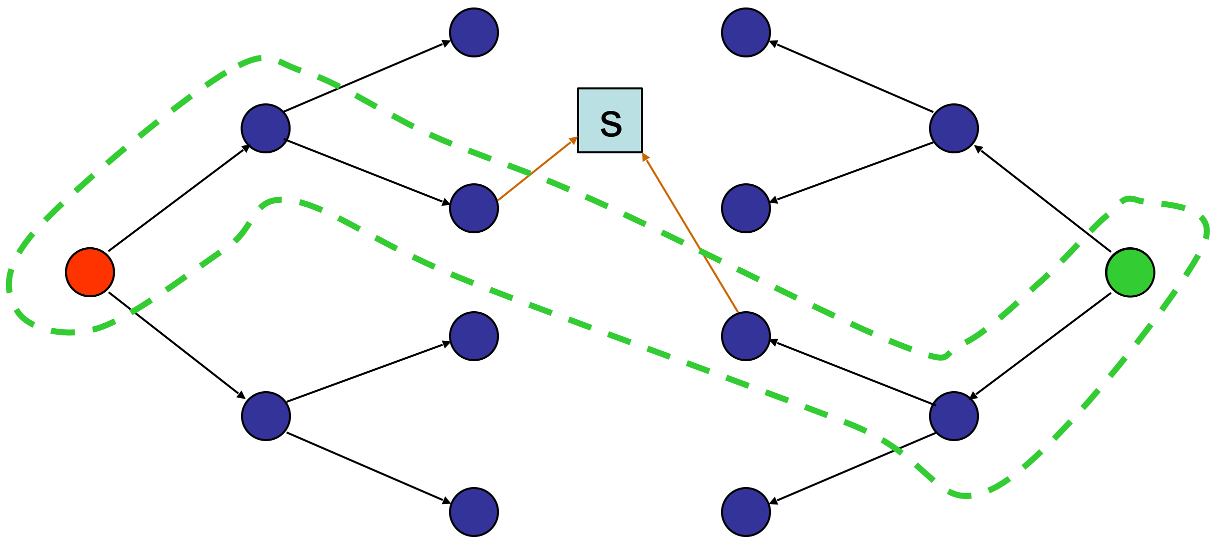
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

? →

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

Bidirectional Strategy

2 Open-List queues: Open-List1 and Open-List2

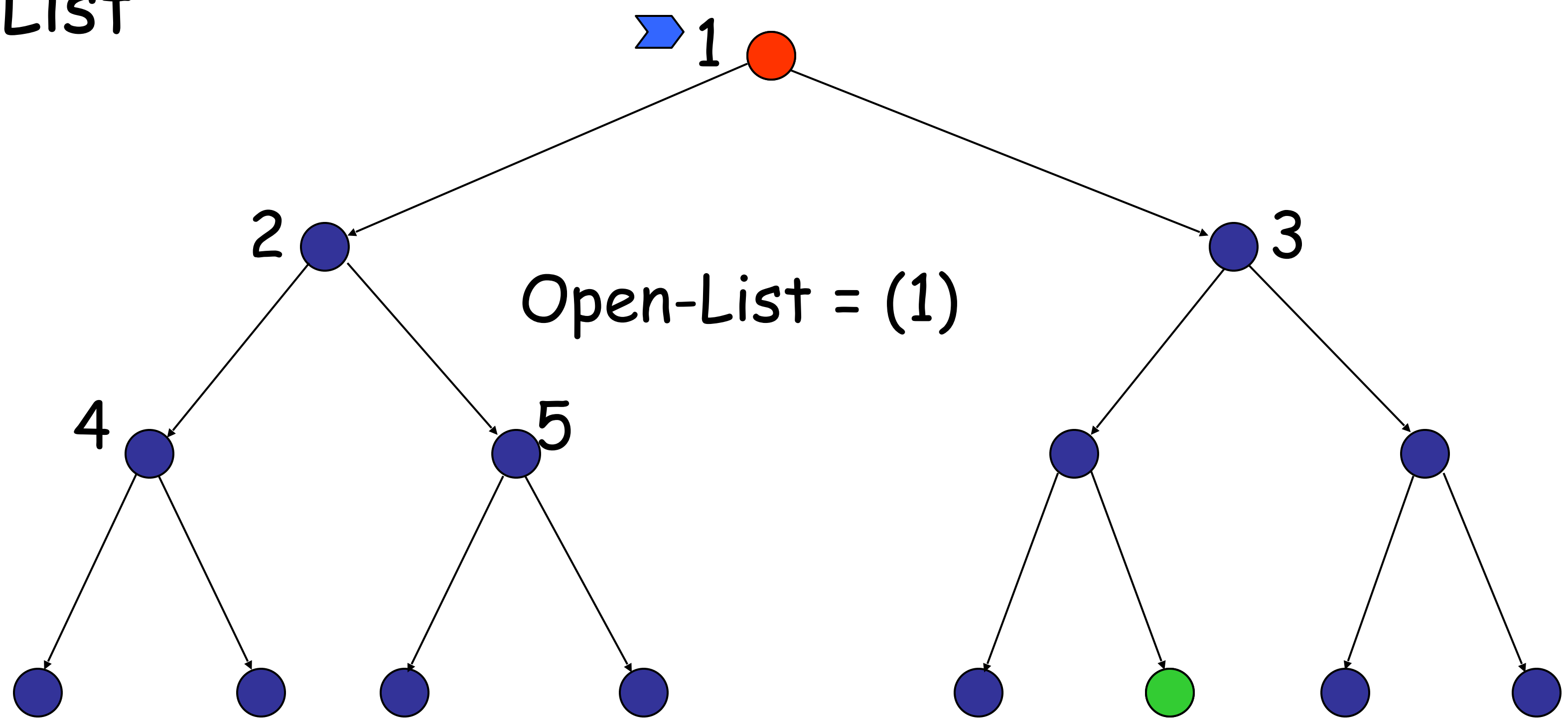


Time and space complexity is $O(b^{d/2}) << O(b^d)$ if both trees have the same branching factor b

Question: What happens if the branching factor is different in each direction?

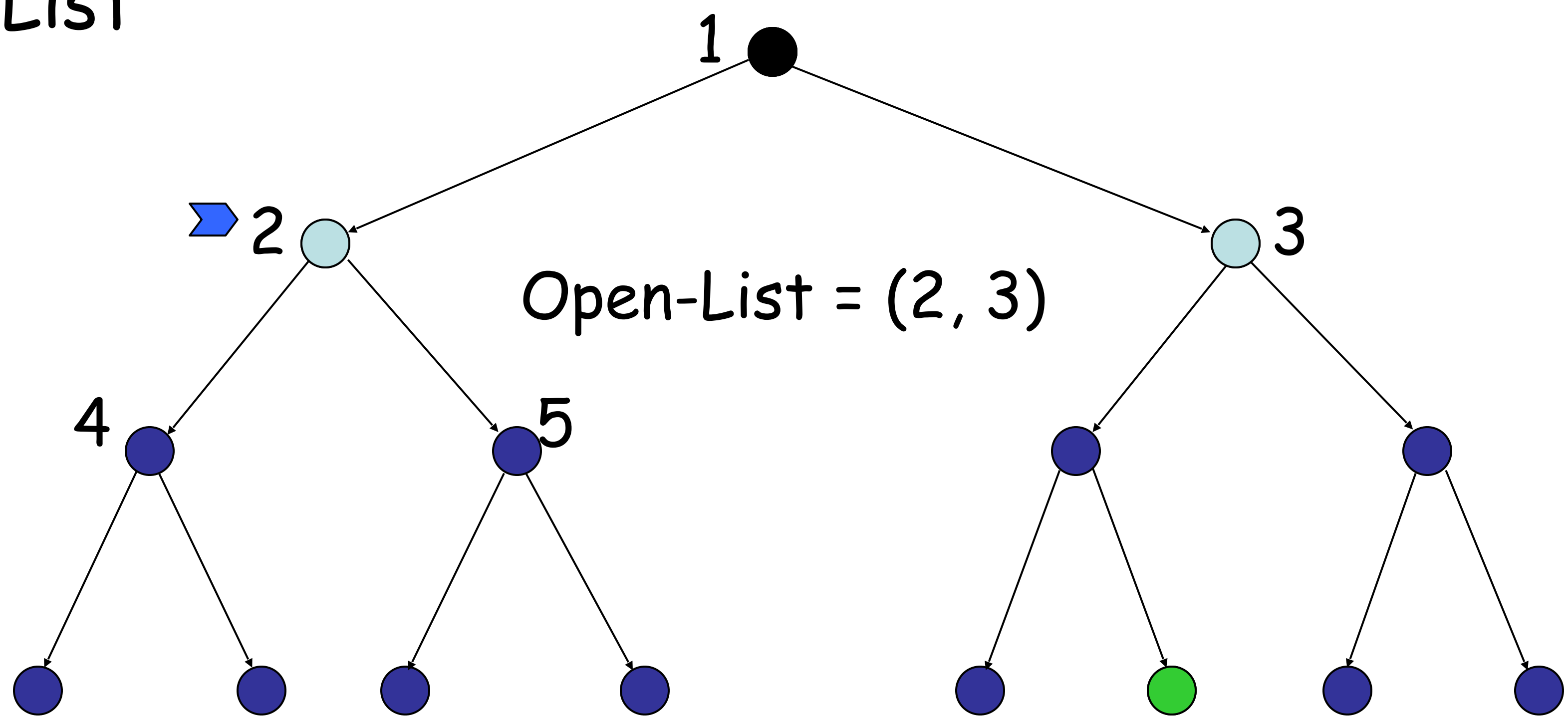
Depth-First Strategy

New nodes are inserted at the front of Open-List



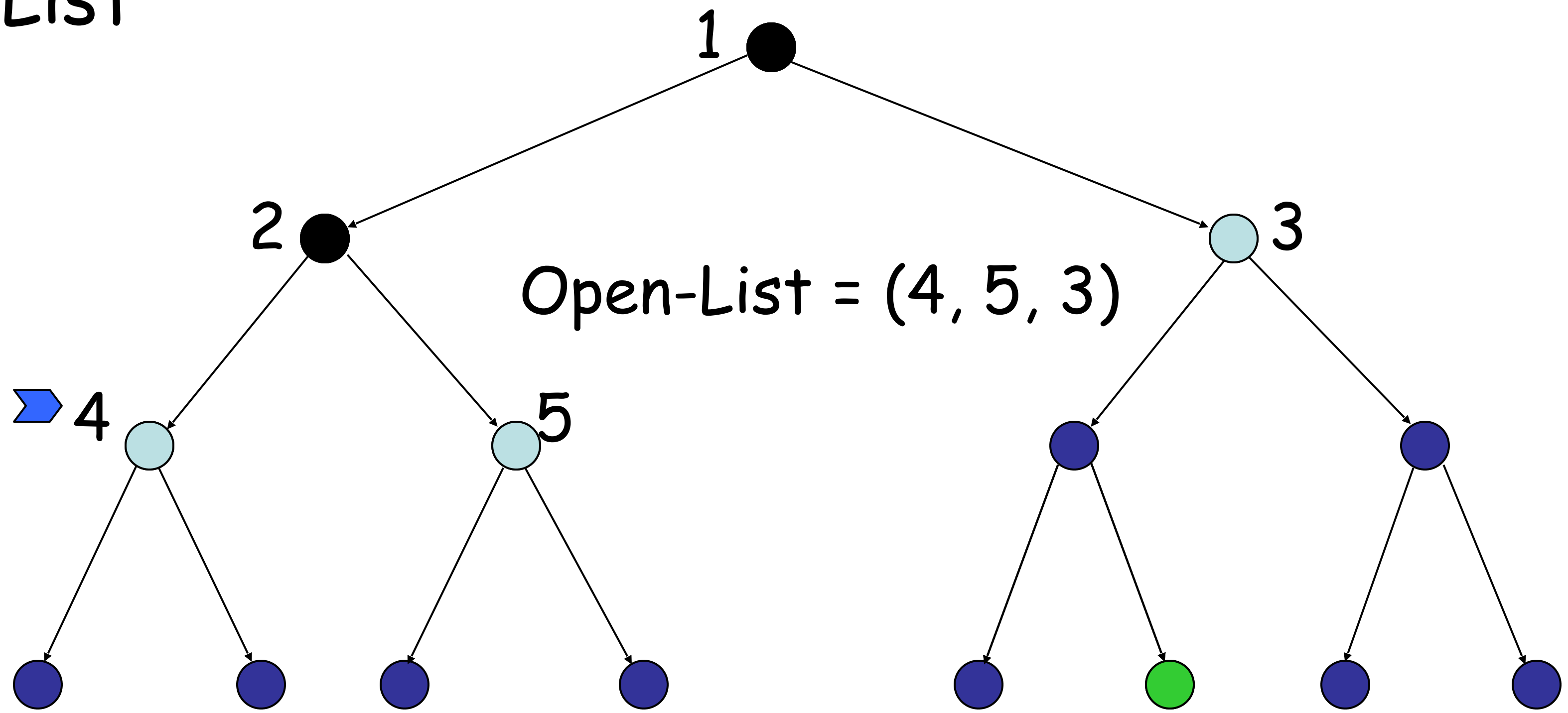
Depth-First Strategy

New nodes are inserted at the front of Open-List



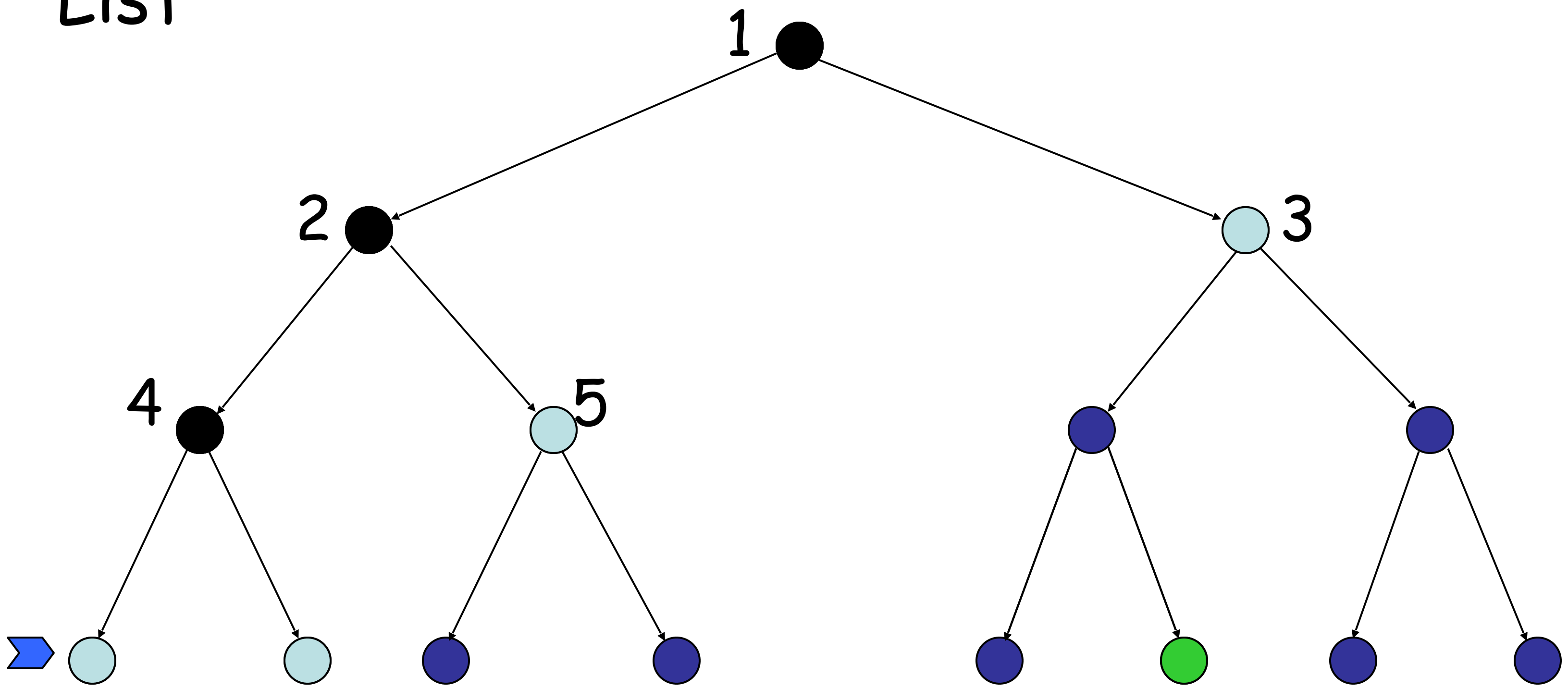
Depth-First Strategy

New nodes are inserted at the front of Open-List



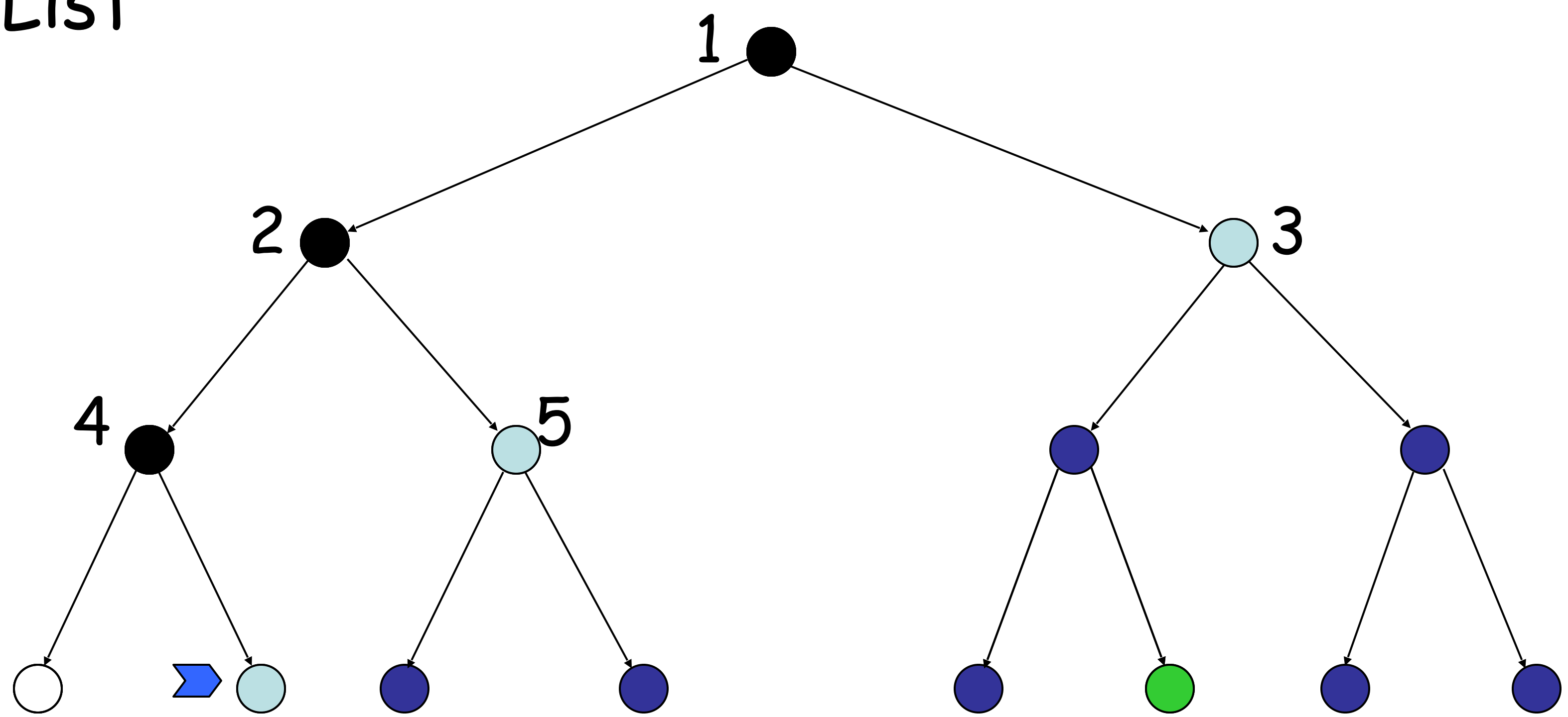
Depth-First Strategy

New nodes are inserted at the front of Open-List



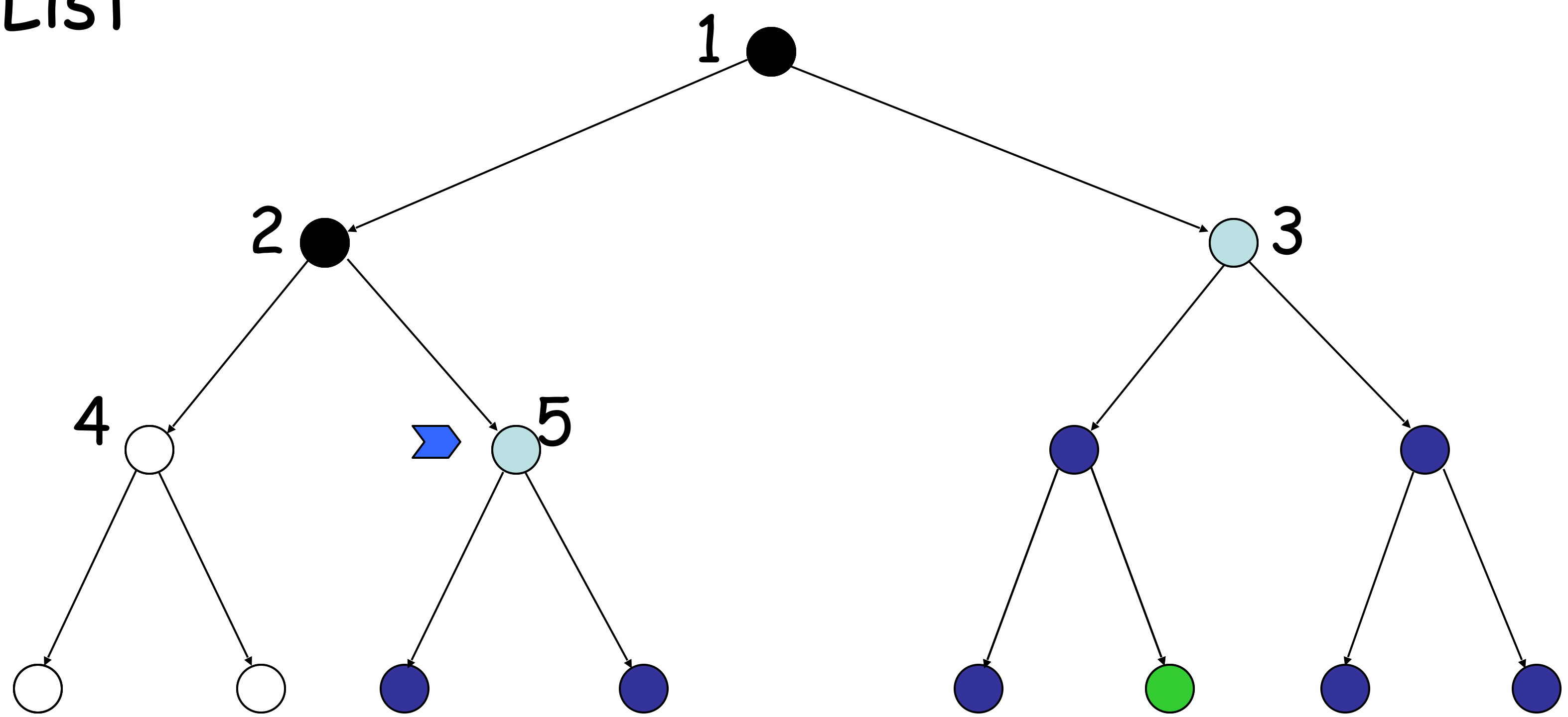
Depth-First Strategy

New nodes are inserted at the front of Open-List



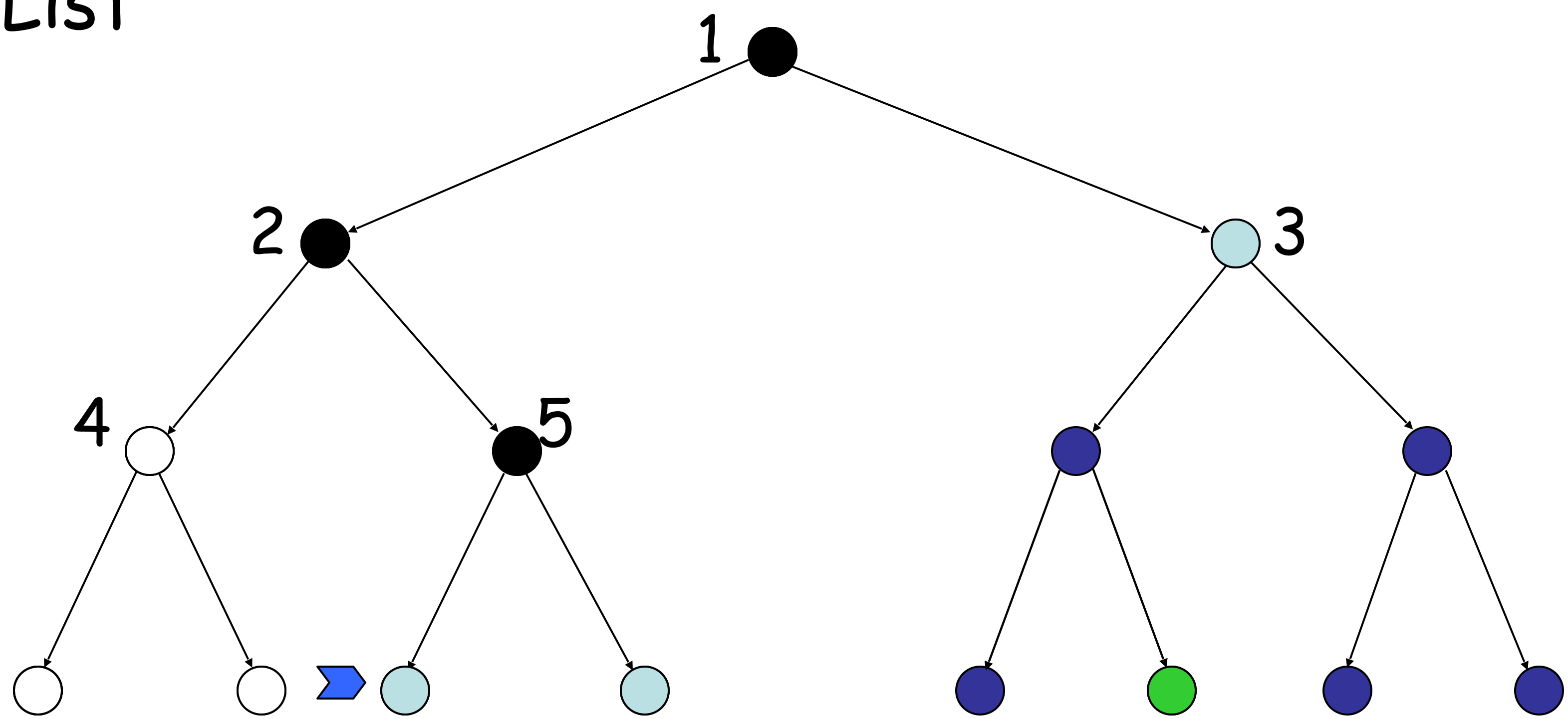
Depth-First Strategy

New nodes are inserted at the front of Open-List



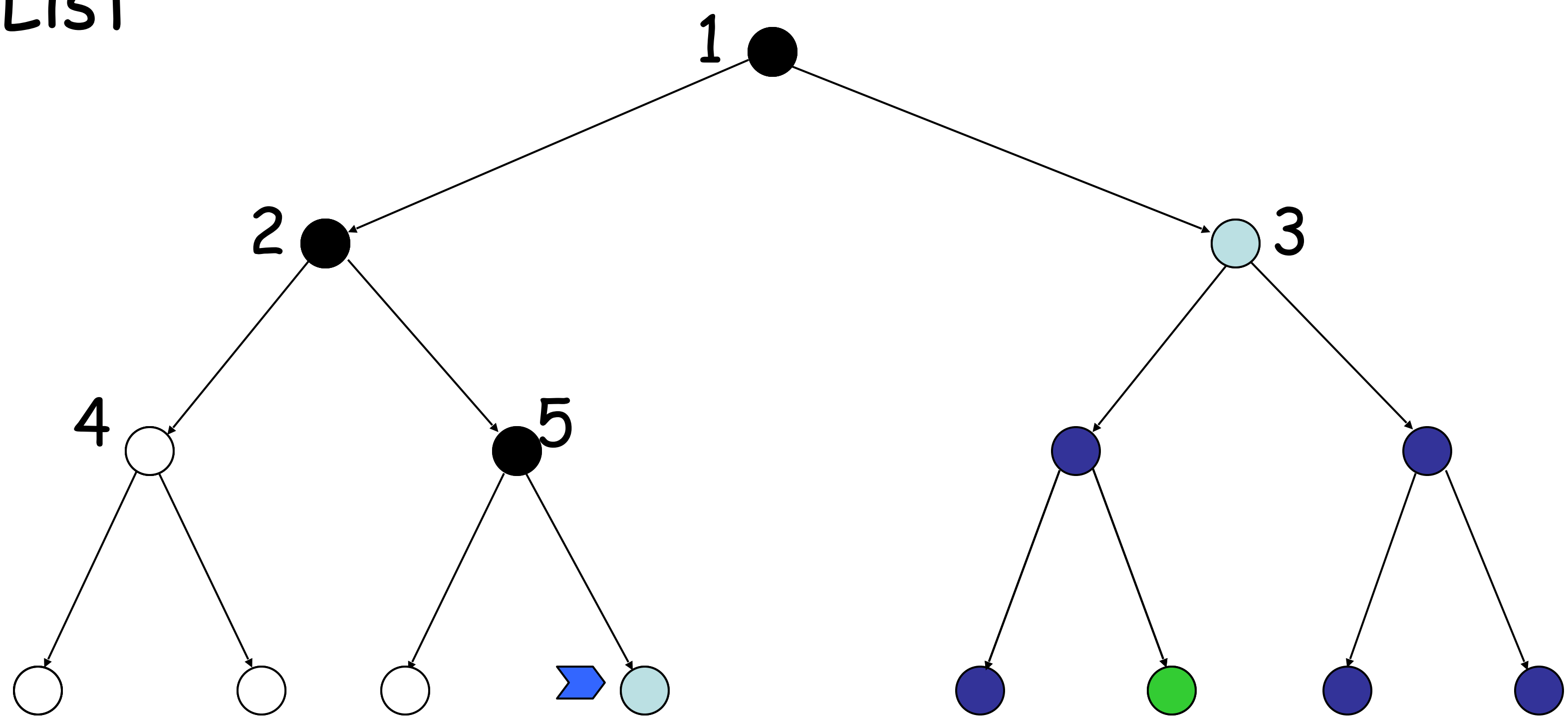
Depth-First Strategy

New nodes are inserted at the front of Open-List



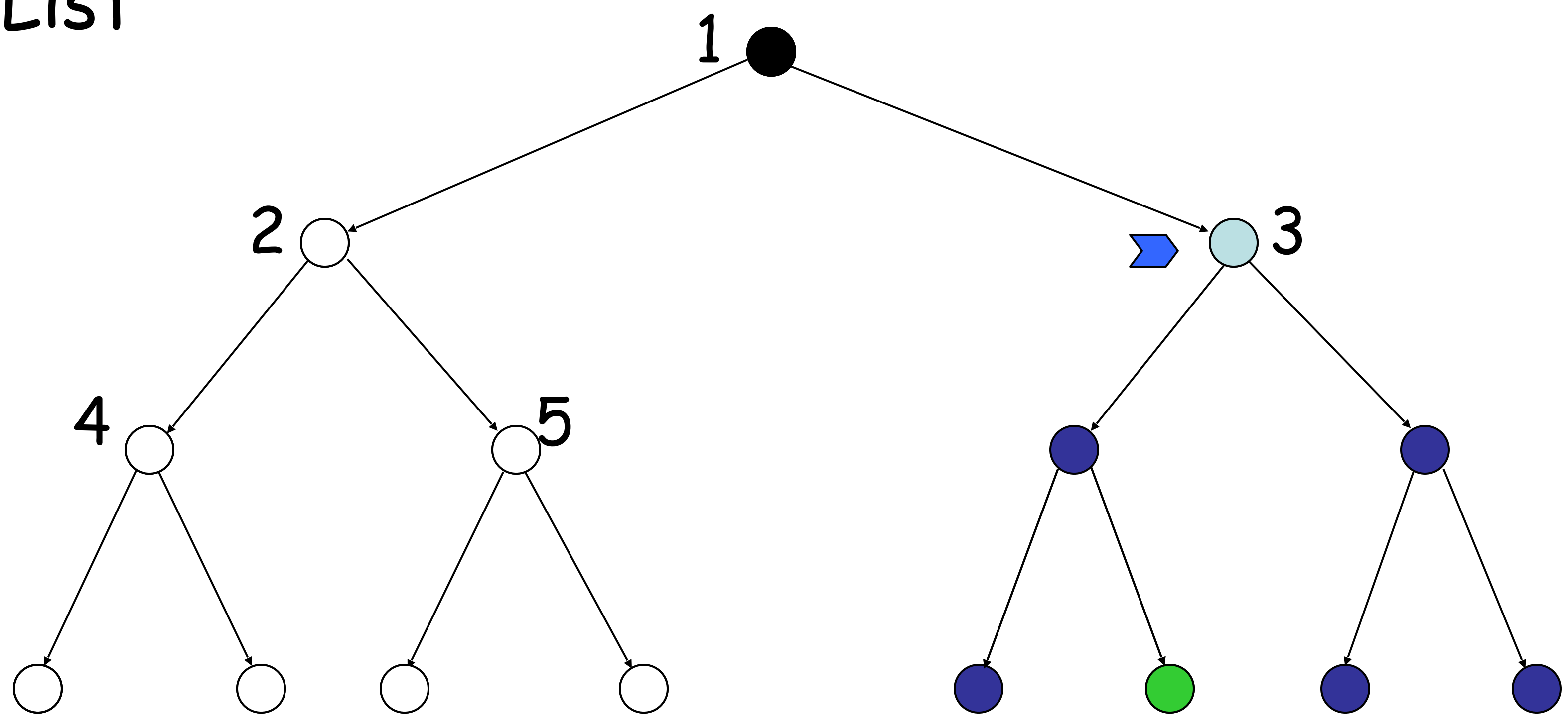
Depth-First Strategy

New nodes are inserted at the front of Open-List



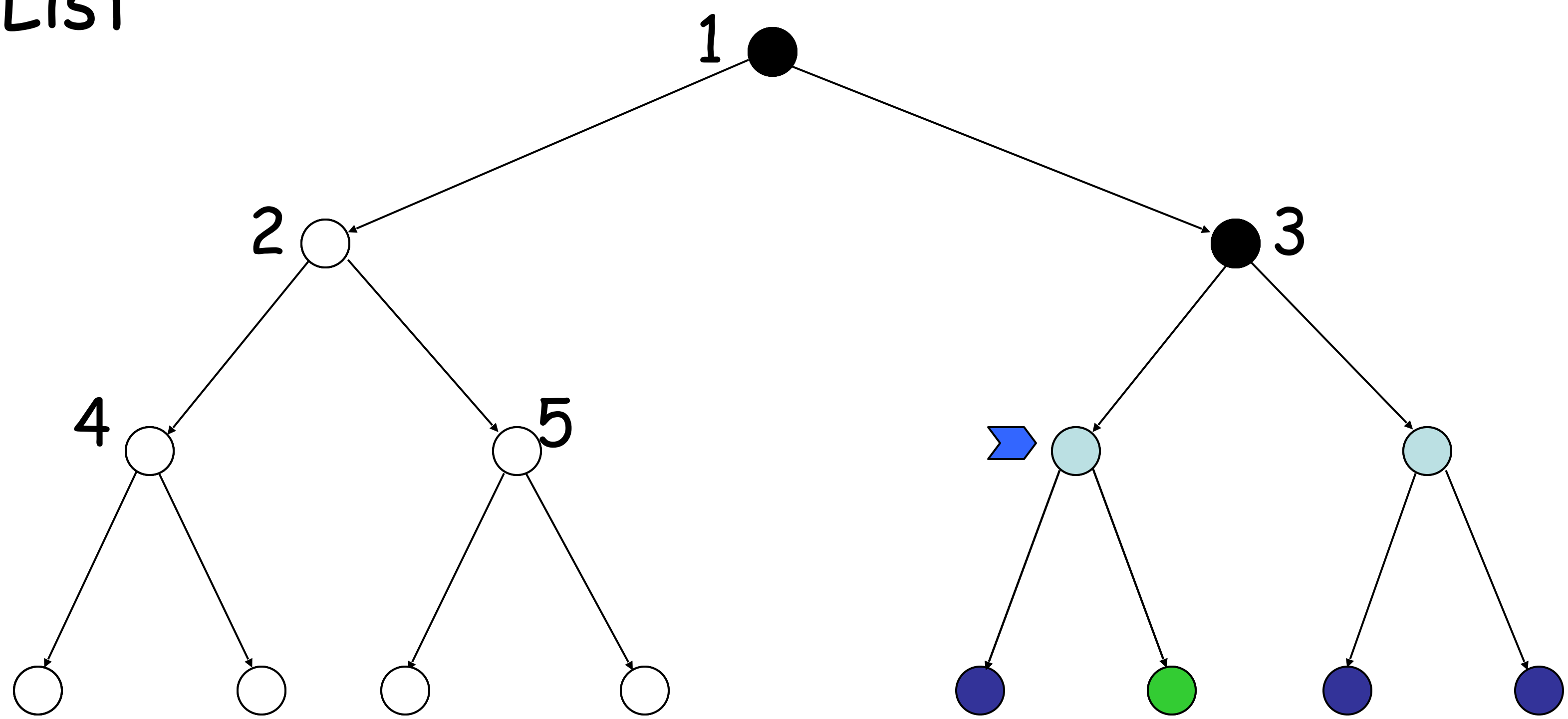
Depth-First Strategy

New nodes are inserted at the front of Open-List



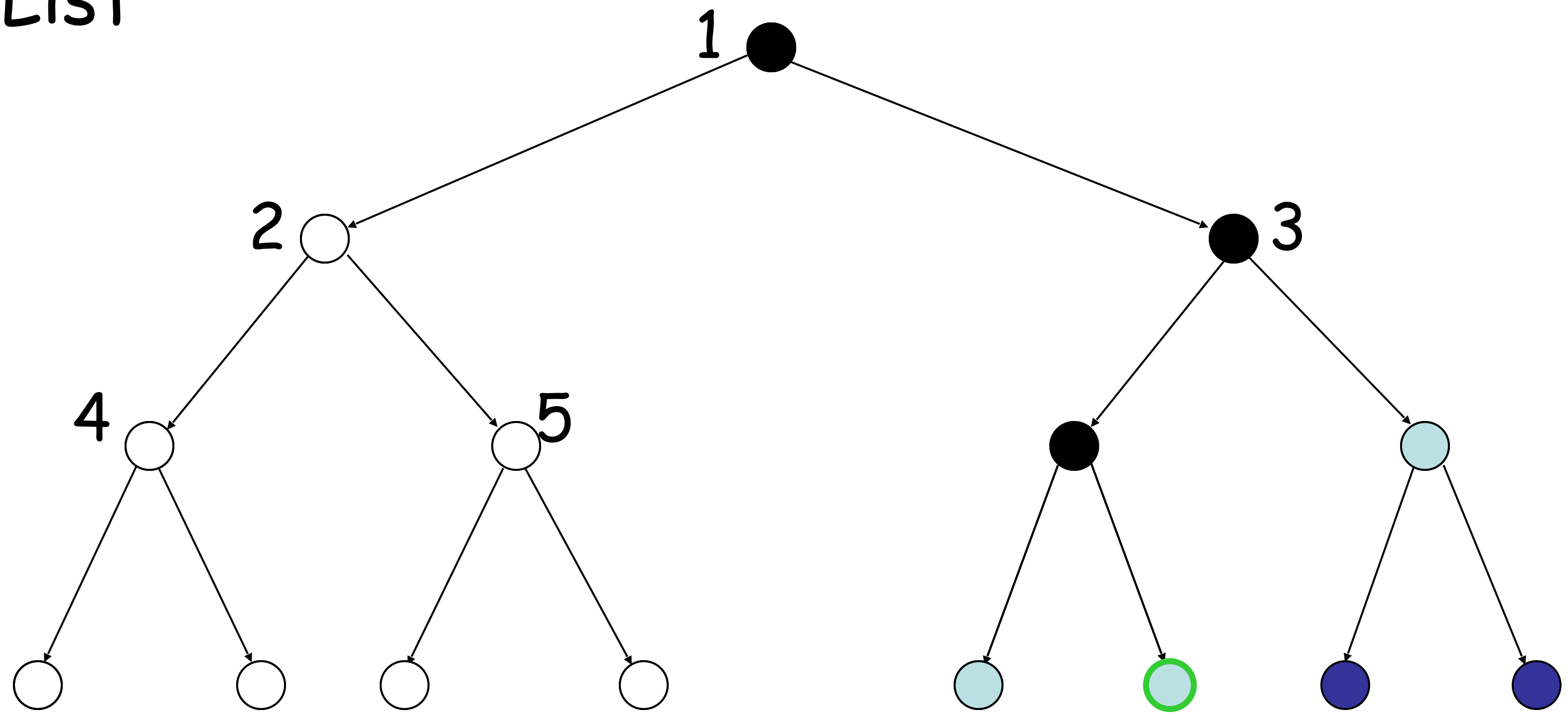
Depth-First Strategy

New nodes are inserted at the front of Open-List



Depth-First Strategy

New nodes are inserted at the front of Open-List



Evaluation

- b: branching factor
- d: depth of shallowest goal node
- m: maximal depth of a leaf node
- Depth-first search is:
  - Complete?
  - Optimal?

Evaluation

- b: branching factor
- d: depth of shallowest goal node
- m: maximal depth of a leaf node
- Depth-first search is:
  - Complete only for finite search tree
  - Not optimal
- Number of nodes generated (worst case):
  $1 + b + b^2 + \ldots + b^m = O(b^m)$
- Time complexity is $O(b^m)$
- Space complexity is $O(bm)$ [or $O(m)$]

[Reminder: Breadth-first requires $O(b^d)$ time and space]

Depth-Limited Search

- Depth-first with depth cutoff k (depth at which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Iterative Deepening Search

Provides the best of both breadth-first and depth-first search

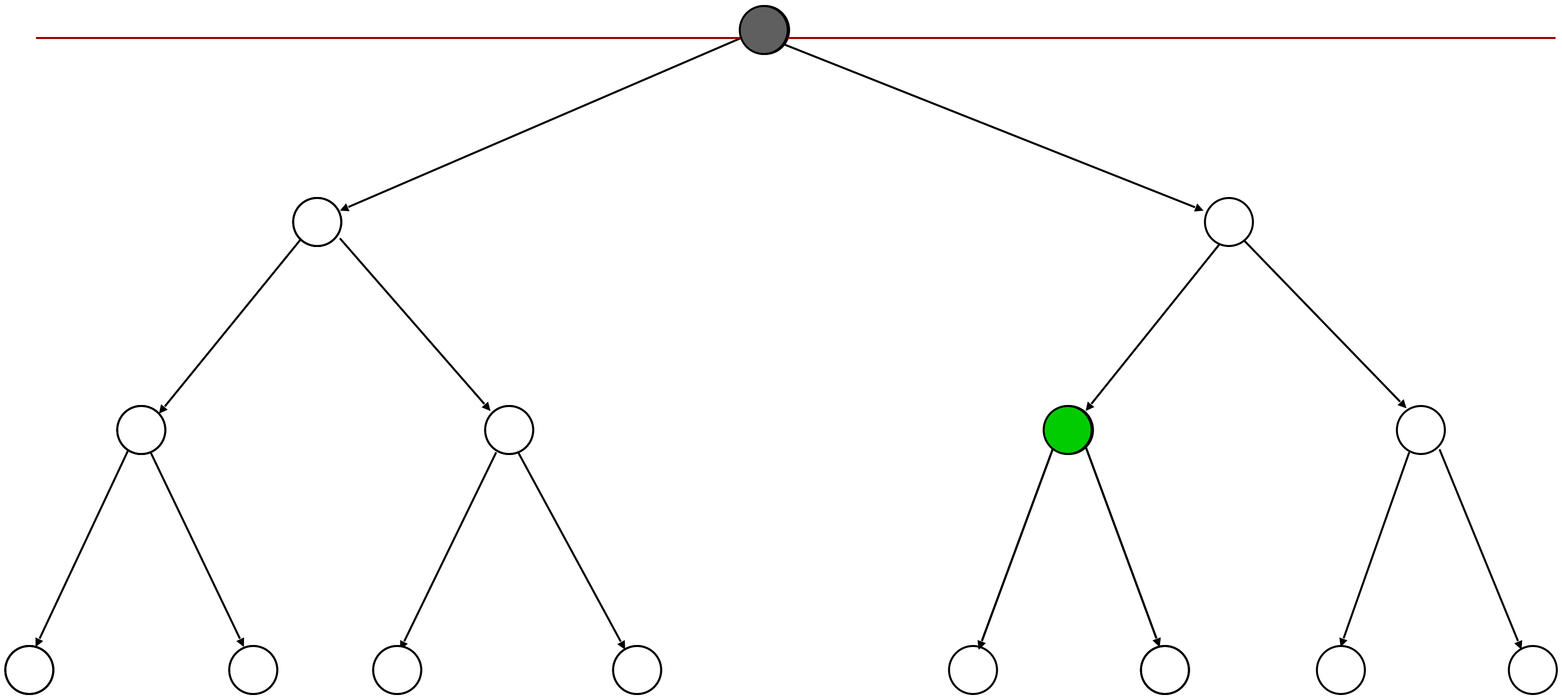
IDS:

For k = 0, 1, 2, … do:

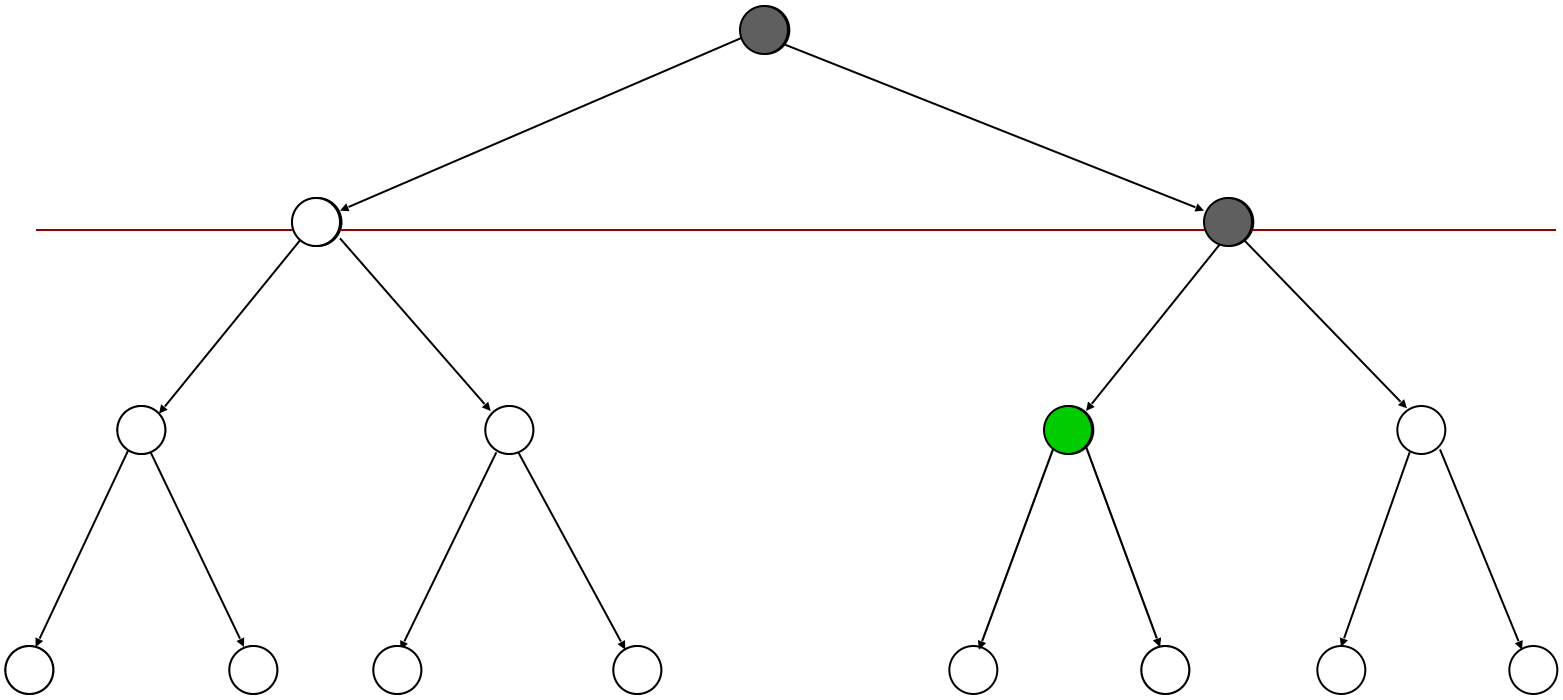
Perform depth-first search with depth cutoff k

(i.e., only generate nodes with depth $\leq$ k)

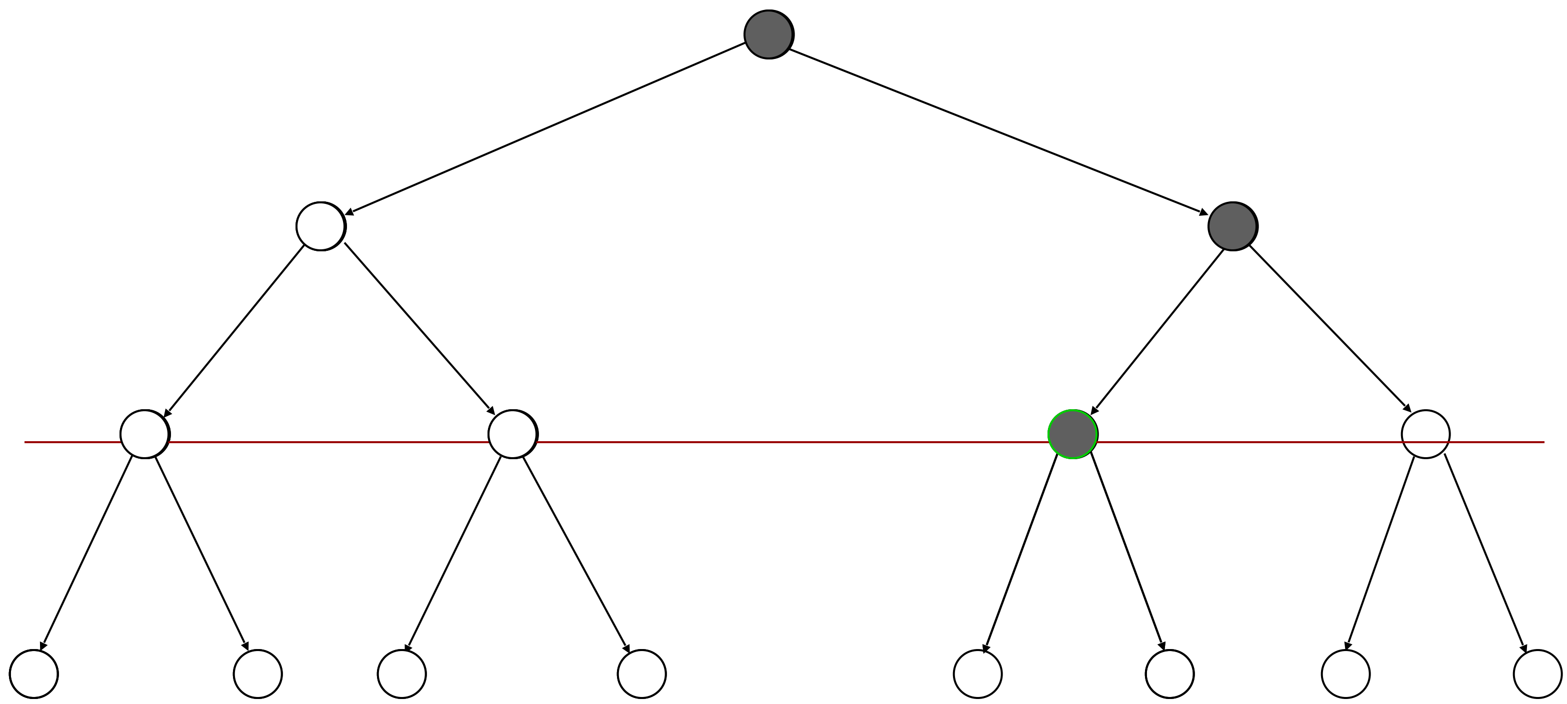
Iterative Deepening



Iterative Deepening



Iterative Deepening



Performance

- Iterative deepening search is:
  - Complete
  - Optimal if step cost =1
- Time complexity is:

  $(d+1)(1) + db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$
- Space complexity is: $O(bd)$ or $O(d)$

Calculation

$$db + (d-1)b^2 + \ldots + (1)\ b^d$$

$$= b^d + 2b^{d-1} + 3b^{d-2} + \ldots + db$$

$$= (1 + 2b^{-1} + 3b^{-2} + \ldots + db^{-d}) \times b^d$$

$$\leq \left(\Sigma_{i=1,\ldots,\infty}\ ib^{(1-i)}\right) \times b^d = b^d\,(b/(b-1))^2$$

Number of Generated Nodes
(Breadth-First & Iterative Deepening)

BF	ID
1	1 x 6 = 6
2	2 x 5 = 10
4	4 x 4 = 16
8	8 x 3 = 24
16	16 x 2 = 32
32	32 x 1 = 32
63	120

d = 5 and b = 2

120/63 ~ 2

Number of Generated Nodes
(Breadth-First & Iterative Deepening)

BF	ID
1	6
10	50
100	400
1,000	3,000
10,000	20,000
100,000	100,000
111,111	123,456

d = 5 and b = 10

123,456/111,111 ~ 1.111

Comparison of Strategies

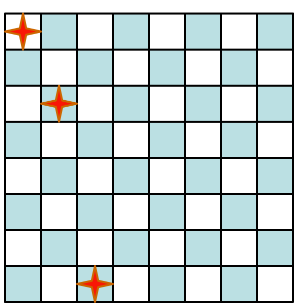
- Breadth-first is complete and optimal, but has high space complexity
- Depth-first is space efficient, but is neither complete, nor optimal
- Iterative deepening is complete and optimal, with the same space complexity as depth-first and almost the same time complexity as breadth-first

Revisited States

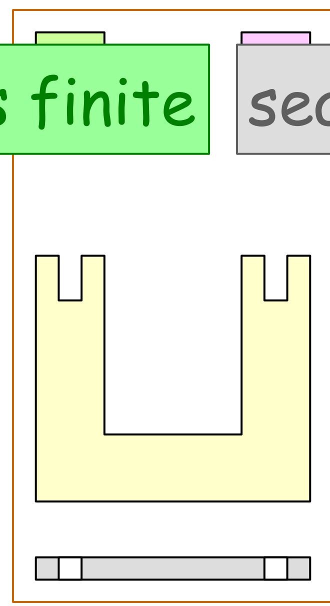
No | Few | Many

search tree is finite

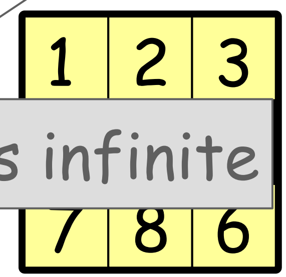
search tree is infinite

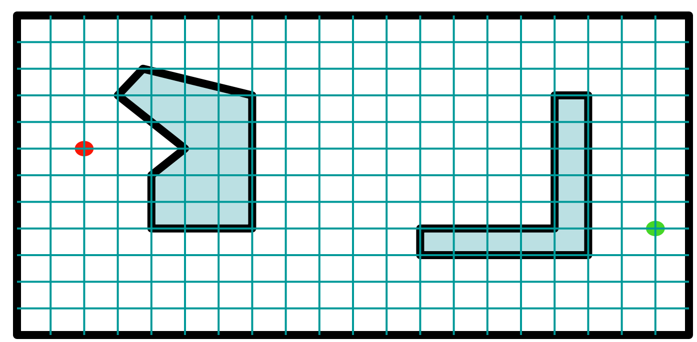


8-queens



assembly planning





8-puzzle and robot navigation

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with **generated** nodes in Closed-List
  - If the state of a new node is in Closed-List, then discard the node

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with **generated** nodes in Closed-List
  - If the state of a new node is in Closed-List, then discard the node

Avoiding Revisited States

- Depth-first search:

  Solution 1:

  - Store all states associated with nodes in current path in Closed-List
  - If the state of a new node is in Closed-List, then discard the node

Avoiding Revisited States

- Depth-first search:

  Solution 1:
  - Store all states associated with nodes in current path in Closed-List
  - If the state of a new node is in Closed-List, then discard the node

  Only avoids loops

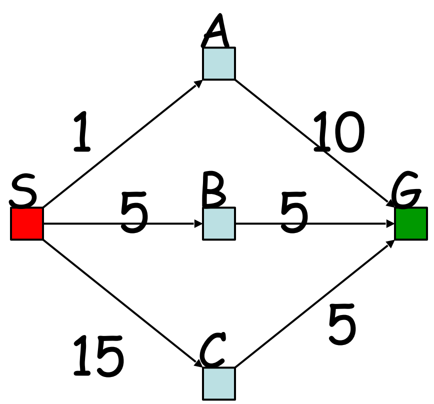
  Solution 2:
  - Store all generated states in Closed-List
  - If the state of a new node is in Closed-List, then discard the node

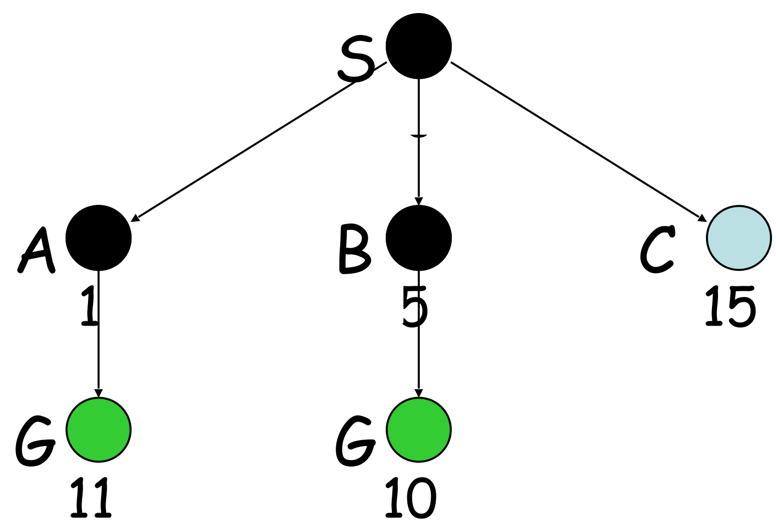
  Same space complexity as breadth-first !

Uniform-Cost Search

- Each arc has some cost $c \geq \varepsilon > 0$
- The cost of the path to each node N is

  $g(N) = \Sigma$ costs of arcs
- The goal is to generate a solution path of minimal cost
- The nodes N in the queue Open-List are sorted in increasing g(N)





- Need to modify search algorithm

Search Algorithm #1

SEARCH#1

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,Open-List)
3. Repeat:
    a. If empty(Open-List) then return failure
    b. N ← REMOVE(Open-List)
    c. s ← STATE(N)
    d. For every state s' in SUCCESSORS(s)
        i. Create a new node N' as a child of N
        ii. If GOAL?(s') then return path or goal state
        iii. INSERT(N',Open-List)

Search Algorithm #2

SEARCH#2

1. INSERT(initial-node,Open-List)
2. Repeat:
   a. If empty(Open-List) then return failure
   b. N ← REMOVE(Open-List)
   c. s ← STATE(N)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i. Create a node N' as a successor of N
      ii. INSERT(N',Open-List)

Avoiding Revisited States in Uniform-Cost Search

- For any state S, when the first node N such that STATE(N) = S is expanded, the path to N is the best path from the initial state to S
- So:
  - When a node is **expanded**, store its state into CLOSED
  - When a new node N is generated:
    - If STATE(N) is in CLOSED, discard N
    - If there exits a node N' in the Open-List such that STATE(N') = STATE(N), discard the node (N or N') with the highest-cost path

Homework

Permutation Inversions

- A tile j appears after a tile i if either j appears on the same row as i to the right of i, or on another row below the row of i.
- For every i = 1, 2, ..., 15, let n_i be the number of tiles j < i that appear after tile i (permutation inversions)
- $N = n_2 + n_3 + \ldots + n_{15}$ + row number of empty tile

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

$n_2 = 0$ $n_3 = 0$ $n_4 = 0$

$n_5 = 0$ $n_6 = 0$ $n_7 = 1$

$n_8 = 1$ $n_9 = 1$ $n_{10} = 4$

$n_{11} = 0$ $n_{12} = 0$ $n_{13} = 0$

$n_{14} = 0$ $n_{15} = 0$

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

→ N = 7 + 4

- Proposition: (N mod 2) is invariant under any legal move of the empty tile
- Proof:
  - Any horizontal move of the empty tile leaves N unchanged
  - A vertical move of the empty tile changes N by an even increment ($\pm 1 \pm 1 \pm 1 \pm 1$)

s =

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

s' =

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

$N(s') = N(s) + 3 + 1$

- Proposition: (N mod 2) is invariant under any legal move of the empty tile

- → For a goal state g to be reachable from a state s, a necessary condition is that N(g) and N(s) have the same parity

- It can be shown that this is also a sufficient condition