CSC 384
Introduction to Artificial Intelligence

Alice Gao and Randy Hickey

Winter 2023

Instructors

Alice Gao

Randy Hickey

Alice Gao

- Grew up in Beijing, China.
- Moved to Vancouver in high school.
- Undergrad in Math and CS w/ Co-op @UBC.
- PhD in CS @Harvard.
  - Algorithmic Game Theory, Information Elicitation and Aggregation
- Microsoft Research Internships @UK and @NYC.
- Postdoc @UBC.
- Lecturer @UWaterloo.
- Assistant Professor, Teaching Stream @UofT.
- Current Research in CS Education
  - Procrastination.

Randy Hickey

- grew up in Toronto, Canada
- B.Sc. at York University in Computer Science
- M.Sc. at U of T in Computer Science
- still completing PhD at U of T
- my research is specialized in Artificial Intelligence!
- specifically, my research is about improving Propositional Satisfiability (SAT) solvers, which are a subset of Constraint Satisfaction Problems (CSP), which we study in Unit 3. SAT Solvers have many applications in Knowledge Representation (KR), which we study in Unit 5.

Q: Questions or comments for Alice and Randy?

Meet Your Peers!

- In the next 2 minutes, introduce yourself
- to someone you don’t know.
- Talk about courses, internship, dorms, extracurricular activities, graduation, jobs, etc.

Q: Share something interesting about your new/old friend!

OUR FAVOURITE AI MOMENTS

Q: Why did you decide to learn about AI?

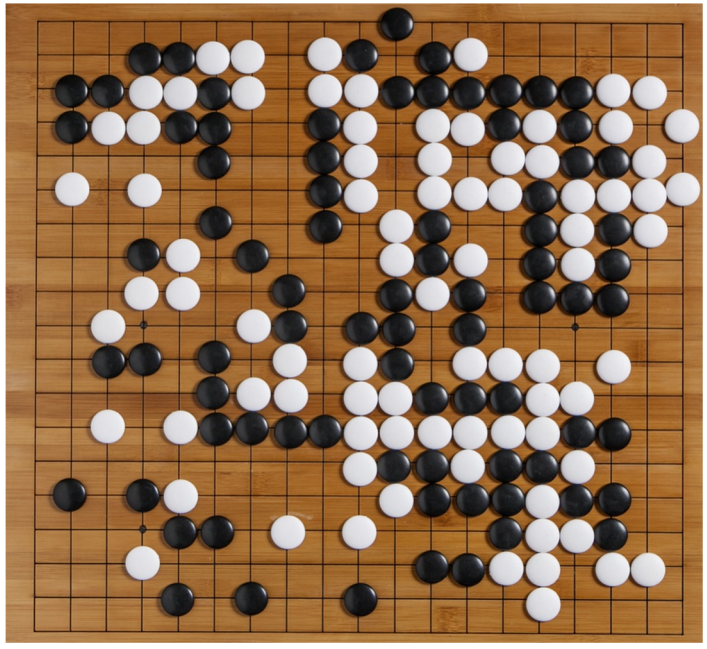
Share your thoughts!

DeepBlue vs. Kasparov (1997)

IBM’s Deep Blue beats chess grandmaster Garry Kasparov

AlphaGo vs. Lee Sedol (2016)

- AlphaGo defeated Lee Sedol (4-1).
- Tree search, deep convolutional neural networks, supervised learning, reinforcement learning.



AlphaGo and DeepMind

2016: AlphaGo beats 9-Dan pro Go player Lee Sedol

- Monte Carlo tree search + neural net trained on human games
- Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).

2017: AlphaGo Zero

- Neural net completely *self taught* (no human games data)
- Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017). https://doi.org/10.1038/nature24270

2017: AlphaZero

- Plays more games – chess, shogi
- Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362.6419 (2018): 1140-1144.

2019: MuZero

- Learns without knowing rules
- Schrittwieser, J., Antonoglou, I., Hubert, T. *et al.* Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* **588**, 604–609 (2020).

StarCraft II (2019)

Vinyals et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782), 350-354.

2-minute paper:
https://www.youtube.com/watch?v=jtlrWblOyP4&t

Watson v.s. Rutter & Jennings (2011)

- IBM's Watson trains a question-answering system on unstructured data (Wikipedia, IMDB, etc)
- Defeats Jeopardy champions Brad Rutter and Ken Jennings by a significant margin.

Autonomous Driving

- 2005: DARPA Grand Challenge – Stanford won!
- 2007: DARPA Urban Challenge – CMU won!
- The perception and decision-making systems.
- A survey paper on self-driving cars

Stanford's Stanley

CMU's Boss

…AND LOTS OF IT IS HAPPENING IN TORONTO!

theReview

How Canada became a hotspot for artificial intelligence research

Nvidia launches new metaverse efforts at SIGGRAPH

TRANSPORTATION | UBER | SELF-DRIVING CARS

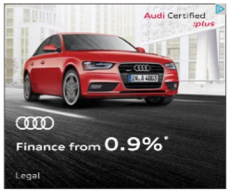
Uber hired noted AI researcher Raquel Urtasun to lead its self-driving expansion into Canada

Urtasun will continue to teach at the University of Toronto part-time and will be joined by eight of her students.

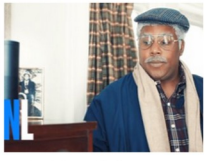
BY JOHANA BHUIYAN | @JMBOOYAH | MAY 8, 2017, 10:32AM EDT

TWEET SHARE LINKEDIN

Raquel Urtason | uber



TRENDING



Watch this 'SNL' skit about an Amazon Echo Silver for senior citizens

Vector Institute is just the latest in Canada's AI expansion

By Jessica Murphy
BBC News, Toronto

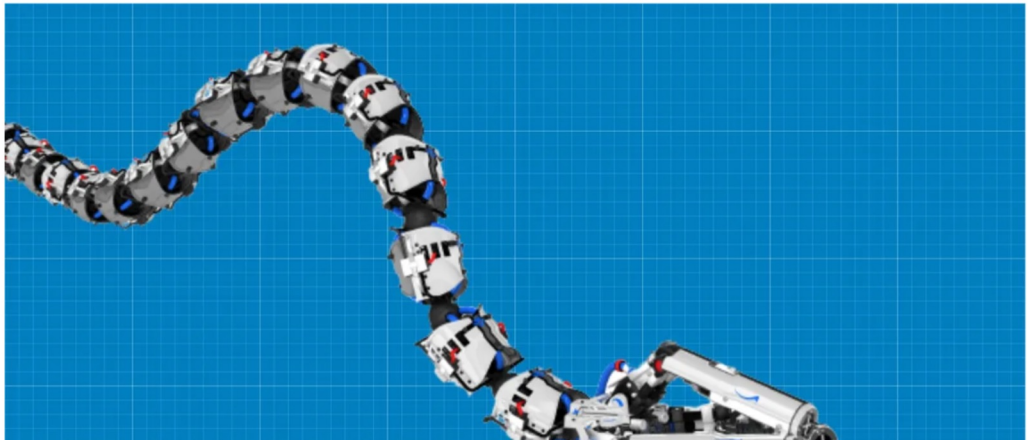
29 March 2017 Share

These snake-like robots could be used in surgery to save lives

The machines are not limited to one industry and have many useful applications.

Loukia Papadopoulos
Created: Aug 13, 2022 11:32 AM

HEALTH



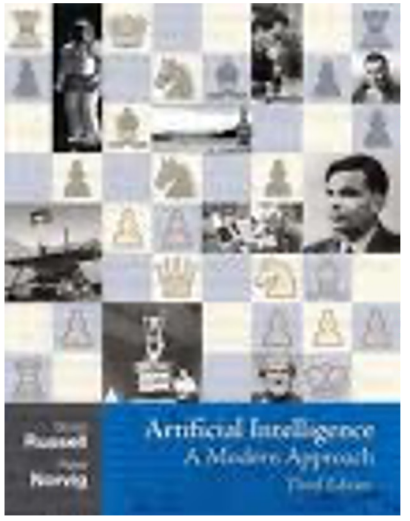
CSC384 TOPICS

Textbook

Artificial Intelligence: A Modern Approach

Stuart Russell and Peter Norvig.

3rd edition, 2010.

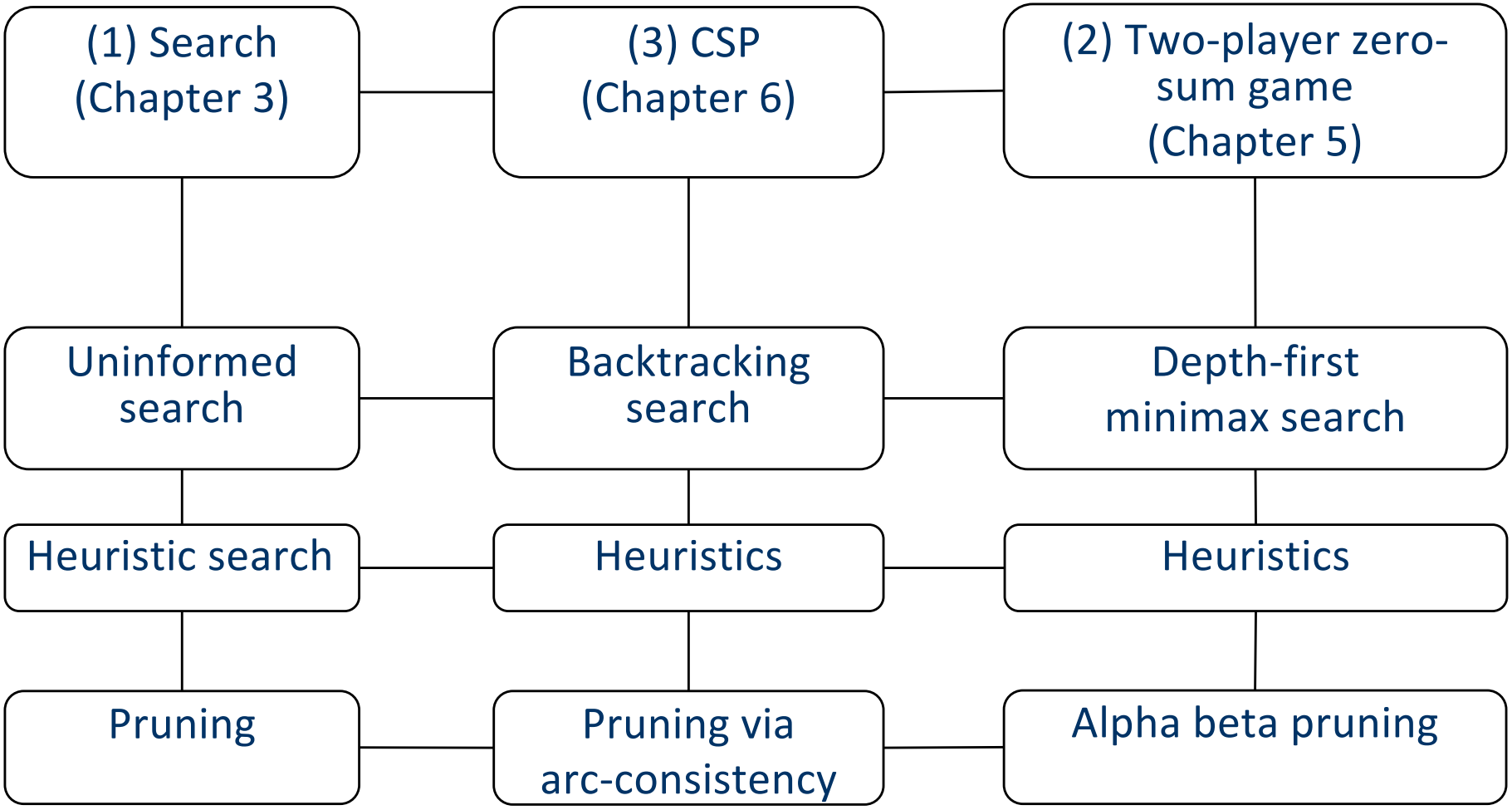


- Recommended but not required.
- Lecture notes will be available online before class.

CSC384 Topics (1/2)

- Search (R&N Ch3)
  - Formulating a search problem
  - Uninformed search
  - Heuristic search
  - Pruning
- Game Tree Search (R&N Ch5)
  - Properties of a game
  - Minimax search
  - Alpha-beta pruning
- Constraint Satisfaction Problems (R&N Ch6)
  - Formulating a CSP
  - Backtracking search
  - Pruning via Arc consistency

CSC 384 Topics (1/2)

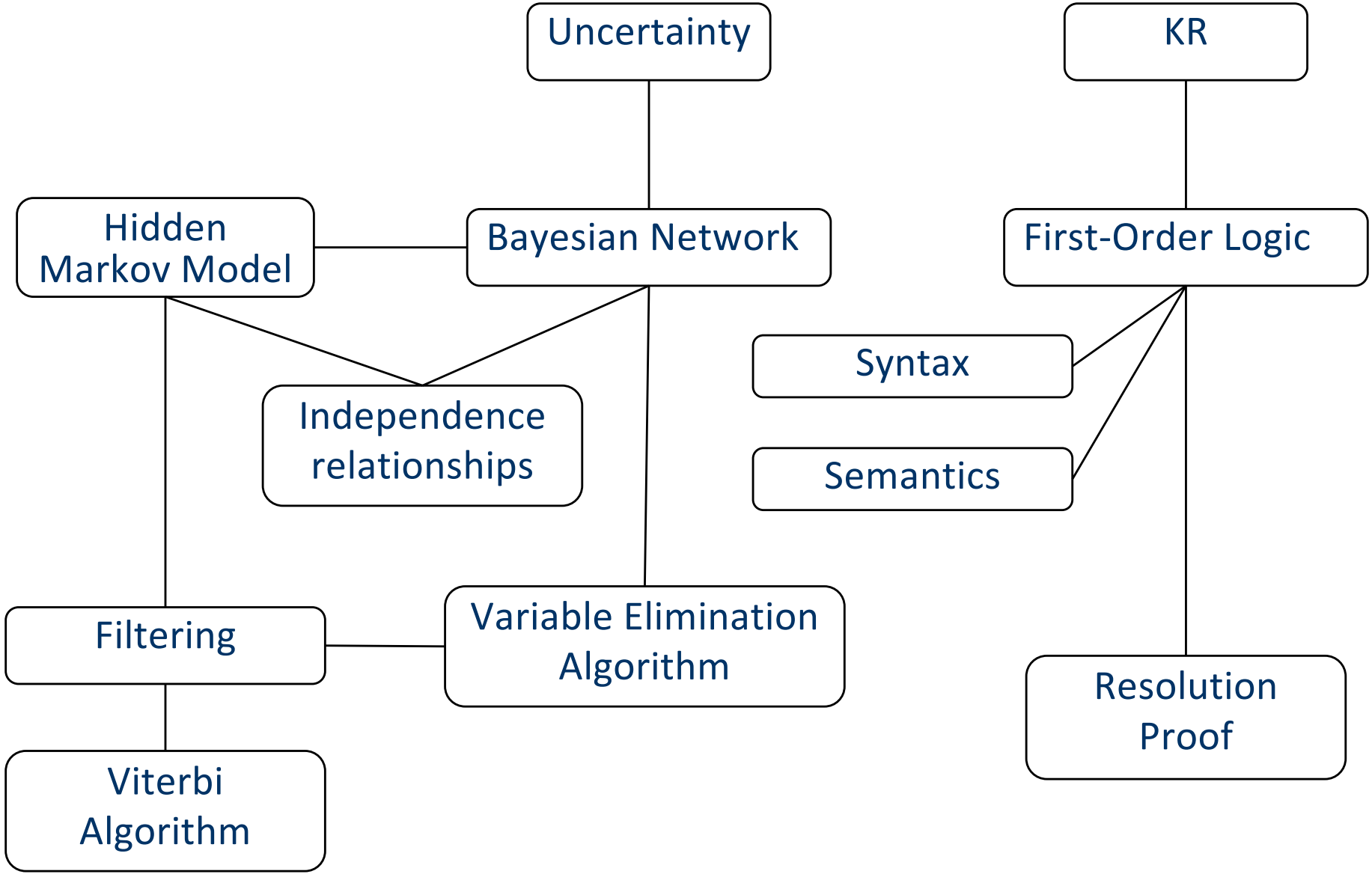


CSC384 Topics (2/2)

- Uncertainty (R&N Ch13-15)
  - Interpreting a Bayesian network.
  - Performing inference in a Bayesian network.

- Knowledge Representation (R&N Ch7-9)
  - Representing knowledge using first-order logic.
  - Constructing a resolution proof.

CSC384 Topics (2/2)



Prerequisites

- Basic data structures, graph, Big O notation and runtime complexity (to analyze and implement algorithms)
- Probability (to understand Bayesian networks)
- Python (for the programming assignments)

You will be responsible for relearning any needed background material.

Q: How are AI and ML related?

Further Courses in AI

- CSC311 Introduction to Machine Learning
- CSC320 Introduction to Visual Computing
- CSC401 Natural Language Computing
- CSC412 Probabilistic Learning and Reasoning
- CSC413 Neural Networks and Deep Learning
- CSC420 Introduction to Image Understanding
- CSC485 Computational Linguistics
- CSC486 Knowledge Representation and Reasoning

Getting Involved in AI @ UofT

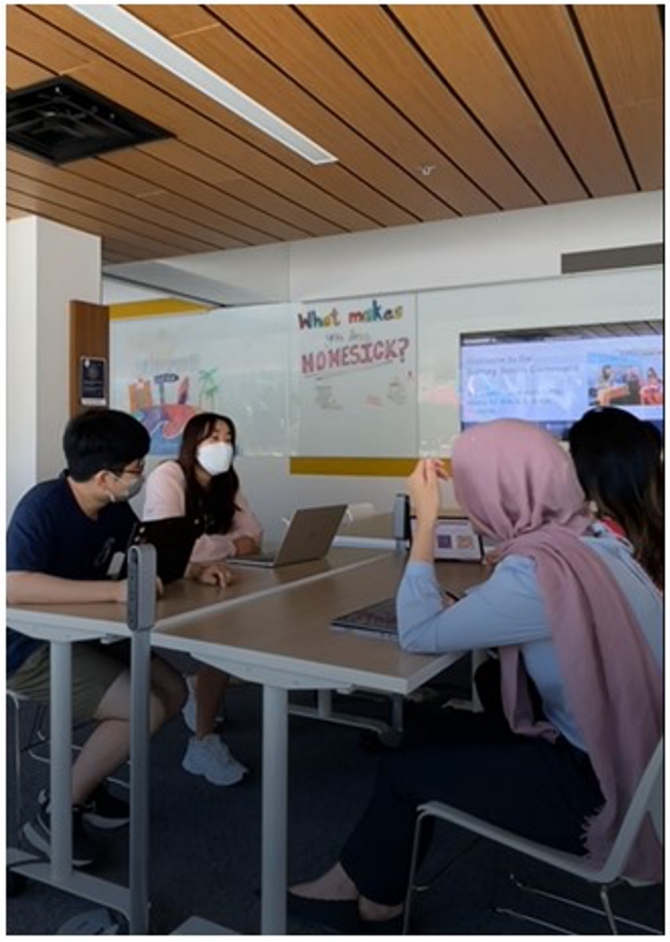
- Undergraduate AI Group (UAIG)
  - https://www.uoft.ai/
  - Project X research competition - https://www.uoft.ai/projectx
- Undergraduate Research
  - Undergraduate Summer Research Program (UGSRP) – Dep't CS
  - Undergraduate Research Fund (Faculty of Arts & Sciences)
  - CSC494/495 Project Courses
- UofT Self-driving Car Team
- "SCI-FI+" AI SciFi Reading Group
  - Run by Sonya Allin at UTM

...

Student Support Resources

- CSC Wellness Resources:
  - https://q.utoronto.ca/courses/221753/pages/wellness-resources
- Student Mental Health Resources
  - https://mentalhealth.utoronto.ca/my-student-support-program/
- Accessibility Resources
  - https://studentlife.utoronto.ca/service/accessibility-services-registration-and-documentation-requirements/
- Student Life
  - https://studentlife.utoronto.ca/
- Recognized Study Groups
  - https://sidneysmithcommons.artsci.utoronto.ca/recognized-study-groups/

UNIVERSITY OF TORONTO
FACULTY OF ARTS & SCIENCE



Join or Lead an RSG

- Meet weekly with up to 8 classmates
- Review and discuss course material
- Prepare for tests and exams
- Get student advice from upper year mentors

Last year, over 3,000 students joined an RSG where they met friends and reached their study goals.

Join an RSG at
UOFT.ME/RECOGNIZEDSTUDYGROUPS

SIDNEY SMITH COMMONS

@sidneysmithcommons

CCR CO-CURRICULAR RECORD

COURSE ADMINISTRATION

Marking Scheme

Course Component	Percentage of Final Grade
4 Assignments (10% each)	35%
4 Tests (5% each)	20%
Final Exam	45%

You must get $\geq 40\%$ on the final exam to pass the course.

Course Schedule

	Monday	Wednesday	Friday	Thursday
Jan 09 - Jan 13	Intro	Search 1 - 2		
Jan 16 - Jan 20	Search 3 - 5			<= A1 posted
Jan 23 - Jan 27	Search Review	Games 1 - 2		
Jan 30 - Feb 03	Test 1 Search	Games 3 - 4		<= A2 posted
Feb 06 - Feb 10	Games Review	CSP 1 - 2		<= A1 due
Feb 13 - Feb 17	Test 2 Games	CSP 3 - 4		<= A3 posted
Feb 20 - Feb 24	Reading Week			
Feb 27 - Mar 03	CSP 5 - 6, Review			<= A2 due
Mar 06 - Mar 10	Test 3 CSP	Uncertainty 1 - 2		<= A4 posted
Mar 13 - Mar 17	Uncertainty 3 - 5			<= A3 due
Mar 20 - Mar 24	Uncertainty 6 - 7, Review			
Mar 27 - Mar 31	Test 4 Uncertainty	KR 1 - 2		
Apr 03 - Apr 07	KR 3, Review			<= A4 due

Lecture Topics

1. Search	6 lectures
2. Game Tree Search	5 lectures
3. Constraint Satisfaction Problems	7 lectures
4. Uncertainty	8 lectures
5. Knowledge Representation	4 lectures

Tests

- Four tests for the first four topics.
- In-class on Mondays.
- Focuses on concepts and theory.

- Why do we have four tests instead of one midterm?
  - Encourage/force you to study regularly.
  - Retain knowledge better.
  - Feel less stressed by the end of the term.

Assignments

- Implement a substantial program in Python.
- Due at 11:59 pm on Thursdays

- Submitted and auto-tested on Markus.
- One-page write-up marked by TA manually.

Final Exam

- 3-hour final exam.
- Covers all the course materials.
- Must get $\geq 40\%$ on the final exam to pass the course.

Academic Integrity

Cheating only cheats yourself!

- Consult U of T Code of Behaviour on Academic Matters

What you should do for assignments:

- Ask questions during office hours.
- Discuss ideas and code examples with others.
- Write code on your own.
- Say no to sending code to others.

What you should do for tests and exams:

- Create practice questions.
- Test yourself/each other under time pressure.

COURSE RESOURCES

A few important resources

- Quercus:
  - https://q.utoronto.ca
- Course email:
  - csc384-2023-01@cs.toronto.edu
- Piazza:
  - link on Quercus.
- MarkUs:
  - https://markus.teach.cs.toronto.edu/2023-01/
- Crowdmark:

Course Resources (1/2)

Admin Matters

- Regular announcements → Piazza
- Admin questions → Course email
- Marks → Quercus

Lectures

- Syllabus, Lecture Slides → Quercus
- Questions about lectures → Piazza

Course Resources (2/2)

Assignments

- Instructions → Quercus
- Questions about assignments → Piazza
- Submission and Remark Requests → MarkUs

Tests

- Questions about tests → Piazza
- Marks → Crowdmark
- Solutions → Quercus
- Remark requests → Course email

Remark Requests

- Assignments: submit remark requests on MarkUs.
- Tests: send a form to the course email.

- When there's a marking error on test/assignment

- Submit within **one week** after marks are released

Special Consideration Requests

- Missing an assessment due to extraordinary circumstances?
  - Send a form and supporting documentation to the course email.

- Acceptable reasons:
  - Late course enrollment
  - Medical conditions (physical/mental health, hospitalizations, injury, accidents)
  - Non-medical conditions (i.e., family/personal emergency)
- Unacceptable reasons:
  - Heavy course loads, multiple assignments/tests during the same period, time management issues

- Accessibility students:
  - Accommodations are listed in Accessibility documentation

Instructor Office Hours

Alice Gao:

- In-person OH:
  - Wednesday, 10-11 am and 4-5 pm in BA 4240.
- Online OH:
  - Friday, 9-11 am, Zoom
  - https://utoronto.zoom.us/j/6768588609
  - Passcode: 829272

Randy Hickey:

- In-person OH: Tuesday, 4:45-5:45 pm, BA 3289.

Or email us to make appointments.

WELCOME TO CSC384!

CSC 384
Introduction to Artificial Intelligence

Definitions of AI and
Introduction to Search

Alice Gao and Randy Hickey

Winter 2023

Learning Goals

By the end of this lecture, you should be able to

- Describe each of the four definitions of AI.
- Compare and contrast the four definitions of AI.
- Give a few reasons why we chose the Rational Agent definition rather than the other three definitions.
- Formulate a real-world problem as a search problem.

Outline

- Definitions of AI
- Applications of Search
- Formulating a Search Problem

DEFINITIONS OF AI

Definitions of AI

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

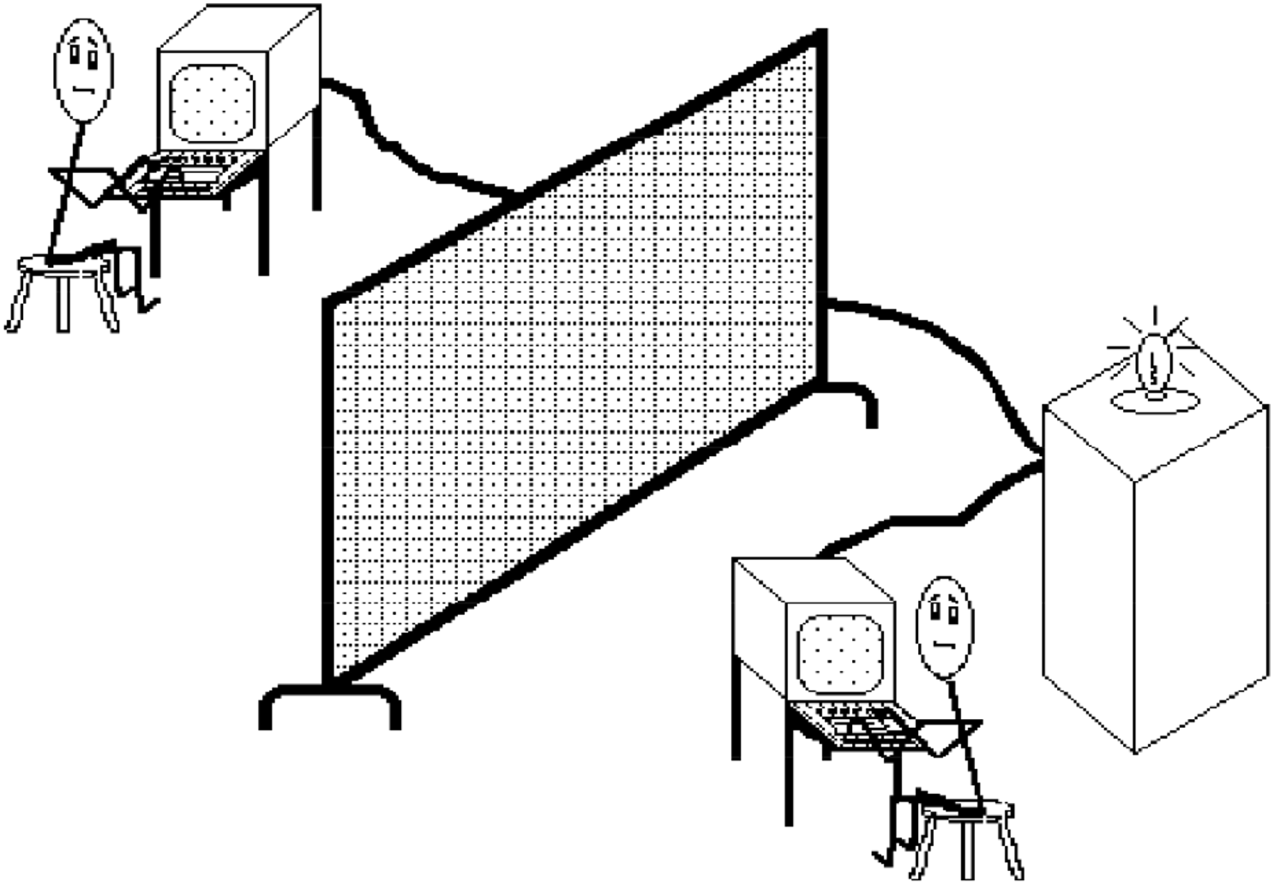
Cognitive Modeling (Thinking Humanly)

- Why humans?
  - Humans is one of a few examples of intelligence.

- How do humans think?
  - Introspection
  - Psychological experiments
  - Brain imaging (MRI)

- Cognitive science:
  - Goa is to develop precise and testable theories of the human mind using theoretical models and experiments.

Turing Test (Acting Humanly)



Turing Test (Acting Humanly)

- An operational definition:
  - If the interrogator cannot distinguish the entity from a human, the entity passes the test and is considered intelligent.

- Is the Turing Test useful?
  - Lots of debate.
  - Allows us to recognize intelligence, but does not provide a way to realize intelligence.
  - Gave rise to core areas of AI: natural language processing, knowledge representation, machine learning, computer vision, robotics, etc.

Rationality

- Rationality: an abstract “ideal” of intelligence, rather than “whatever humans do”.

- A system is rational if it does the “right thing” given what it knows.

Laws of Thought (Thinking Rationally)

- Greek philosopher Aristotle defined syllogisms:
  - formalize correct thinking.
  - gave rise to the field of logic.

- The logicist tradition:
  - goal is to express our knowledge using logic.
  - a system w/ such knowledge can solve any problem (in principle).

- Two obstacles for using this approach in practice.
  - difficult to translate natural language to logic.
  - brute force search through the large # of statements is too slow.

Rational Agent (Acting Rationally)

- Agent comes from Latin word “agere”, meaning “to do” or “to act”.

- A rational agent acts to achieve the best (expected) outcome.

- What behaviour is rational?
  - create and pursue goals,
  - operate autonomously,
  - perceive environment,
  - learn,
  - adapt to changes.

What is Artificial Intelligence?

Cognitive Modeling Systems that think like humans	**Laws of Thought** Systems that think rationally
Turing Test Systems that act like humans	**Rational Agent** Systems that act rationally

Choosing a Definition of AI

Which definition of AI would you choose and why?

Caring about Behaviour Rather than Thoughts

Why do we care about behaviour instead of thinking and reasoning?

- Acting rationally is more general than thinking rationally. Correct inference is only one way to achieve rationality.
- No logically correct thing to do, yet we still need to take an action.
- We have to act quickly without thinking.

Measure Success against Rationality Rather than against Humans

Why do we measure success against rationality instead of against humans?

- Humans often act in ways that we don't consider intelligent. "Predictably Irrational" by Dan Ariely.
- Rationality is mathematically well defined and completely general. It's easy to study it scientifically.
- Developing flying machines v.s. intelligent machines.
  - Develop structures w/ characteristics of birds.
  - Understand the principles of flying.

APPLICATIONS OF SEARCH

What is Search?

- One of the most fundamental techniques in AI
- Solves many problems that humans are not good at.
- Achieve super-human performance on other problems (Chess, Go)
- Useful general algorithmic technique for solving problems (both in AI and in other areas)

Applications of Search

Hua Rong Dao Puzzle

Alice solves Hua Rong Dao

FORMULATING A SEARCH PROBLEM

Why Search

We have a difficult problem.

- Do not know an algorithm to solve this problem.
- Have a description that helps us recognize a solution.

We must search for a solution!

Challenge of Problem Formulation

Real-World Problem → Search Problem

The formulation can have critical impacts on the search process.

Components of a Search Problem

1. A set of states.
2. The initial state.
3. A successor function.
4. The goal states or a goal test.
5. (Optional) a cost associated with each action.
6. (Optional) a heuristic function to guide the search.

A solution to this problem is a path from the initial state to any goal state (optionally with the smallest total cost).

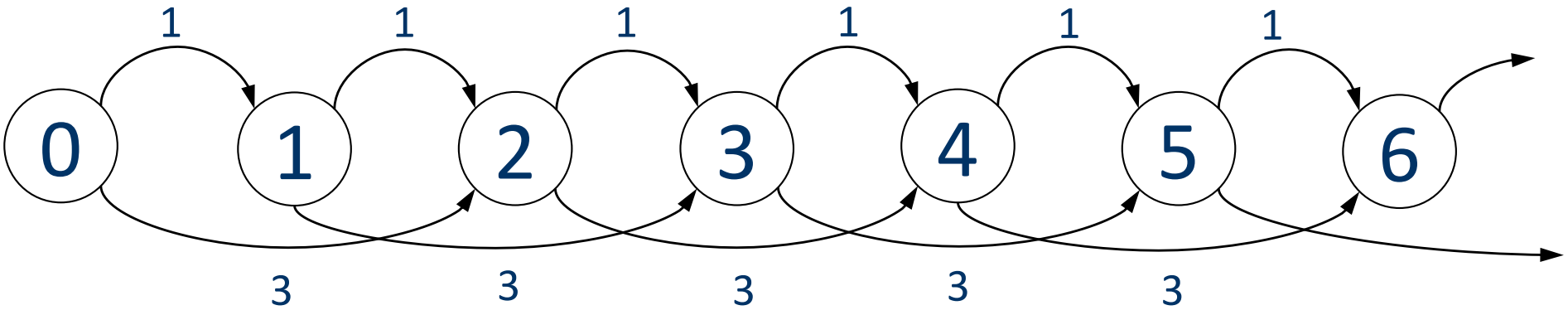
Example: Counting Integers

- Consider the non-negative integers.
- We start from 0.
- Our goal is to reach 5.
- At each step, we can add 1 or 2 to the current integer.
- The cost of adding 1 is 1. The cost of adding 2 is 3.

Counting Integers as a Search Problem

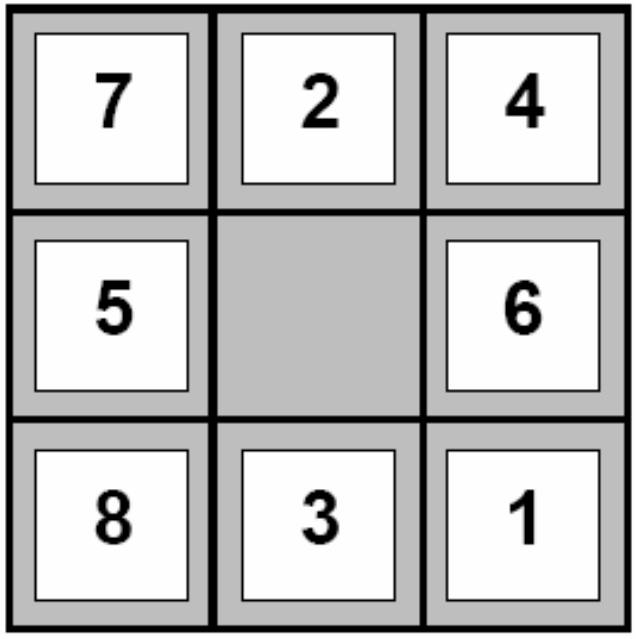
- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.

Partial Search Graph for Counting Integers

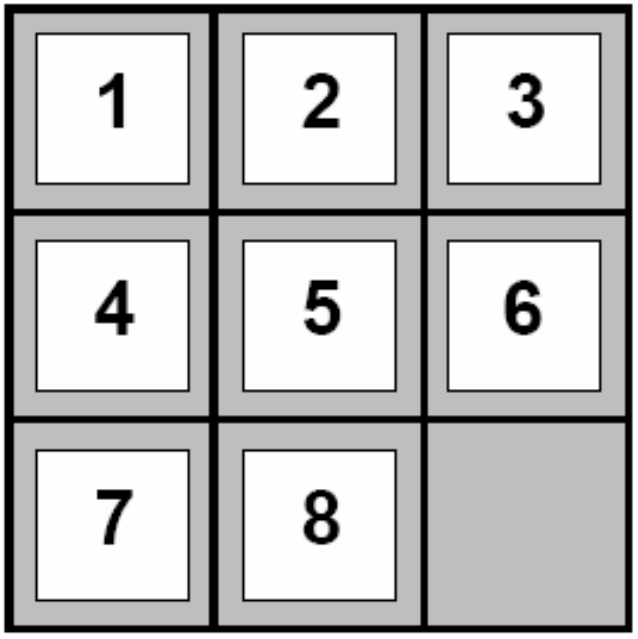


Example: 8-Puzzle

Rule: Can slide tile A into the blank square if A is adjacent to the blank square.



Start State



Goal State

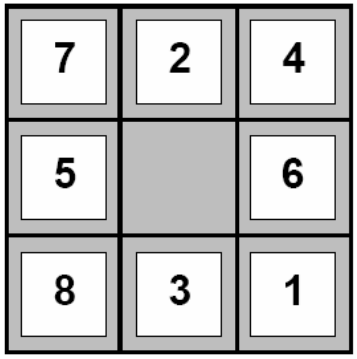
The 8-Puzzle as a Search Problem

State:

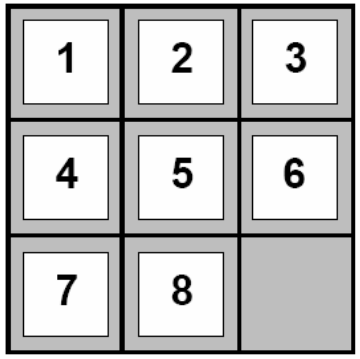
Initial state:

Goal states:

Successor function:



Start State



Goal State

The 8-Puzzle as a Search Problem

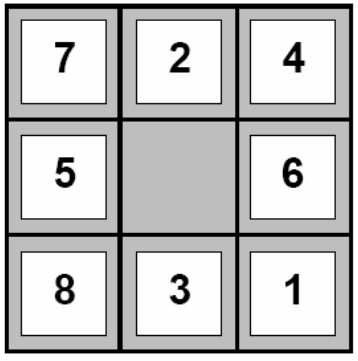
State: $x_{00}x_{01}x_{02}, x_{10}x_{11}x_{12}, x_{20}x_{21}x_{22}$ where x_{ij} is the number in row i and column j, $i, j \in \{0, 1, 2\}, x_{ij} \in \{0, \dots, 8\}$ and $x_{ij} = 0$ denotes the blank tile.

Initial state: 724,506,831

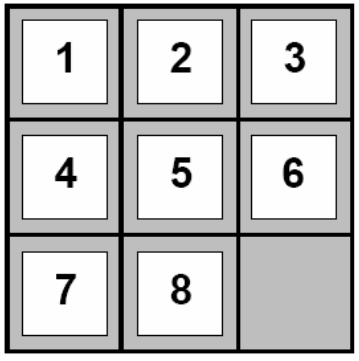
Goal states: 123,456,780

Successor function: State B is a successor of state A if and only if we can transform A to B by moving the blank up, down, left, or right.

Solution a sequence of moves of the blank that transform the initial state to a goal state.



Start State



Goal State

An alternative formulation for 8-Puzzle

State: A state consists of 8 coordinates. (x_i, y_i) denotes the (row, column) coordinates for tile i where $1 \leq i \leq 8$

Initial state: (2,2) (0,1) (2,1) (0,2) (1,0) (1,2) (0,0) (2,0)

Goal states: (0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2,0) (2,1)

Successor function: State B is a successor of state A iff we can transform A to B by moving the blank up, down, left, or right.

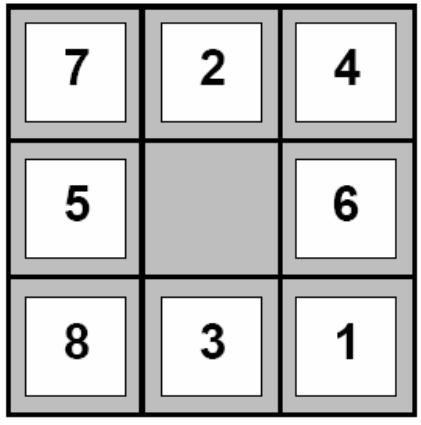
Which formulation do you prefer and why?

1

- Initial state: 724,506,831

2

- Initial state: (2,2) (0,1) (2,1) (0,2) (1,0) (1,2) (0,0) (2,0)



Start State

CSC 384
Introduction to Artificial Intelligence

Uninformed Search

Alice Gao and Randy Hickey

Winter 2023

Learning Goals

By the end of this lecture, you should be able to

- Trace the execution of and implement Breadth-first search, Depth-first search, Iterative-deepening search, and Uniform-cost search.
- Explain the properties of Breadth-first search, Depth-first search, Iterative-deepening search, and Uniform-cost search.
- Describe strategies for pruning the search space.

Outline

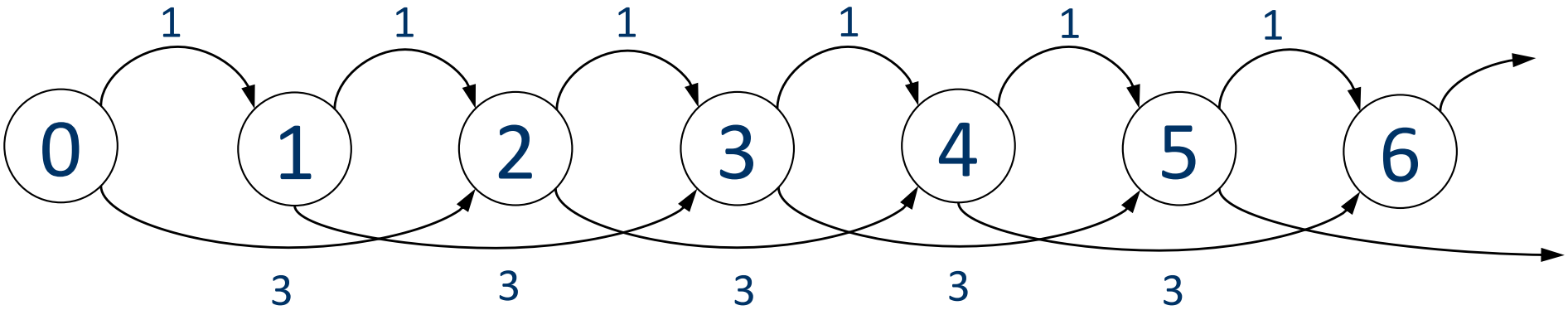
- Generic Search Algorithm
- Depth-First Search
- Breadth-First Search
- Iterative-Deepening Search
- Uniform-Cost Search
- Multiple-Path Pruning

GENERIC SEARCH ALGORITHM

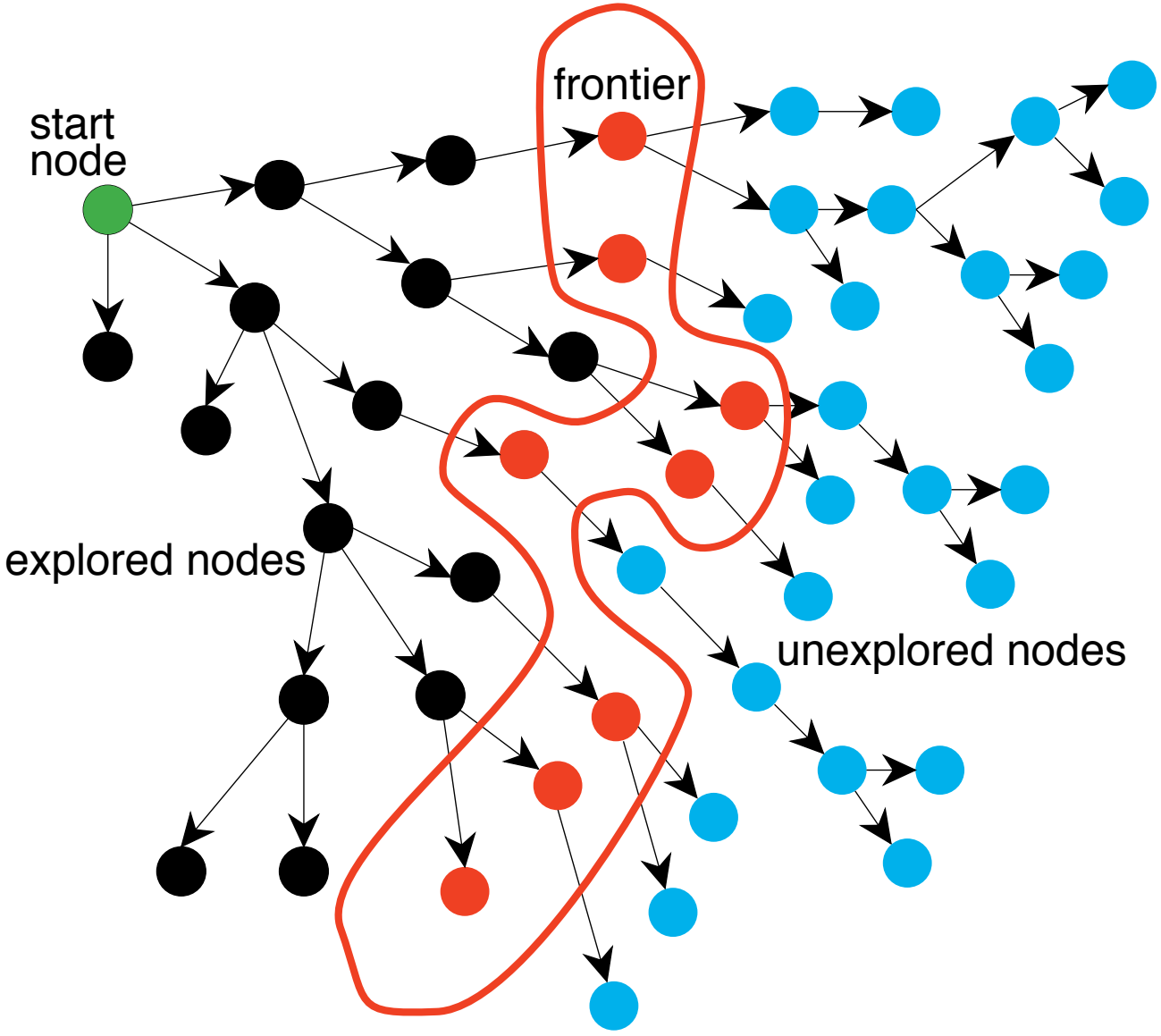
Integer Example

- States: the non-negative integers {0, 1, 2, …}.
- Initial state: 0.
- Goal state: 5.
- Successor function: S(n) = {n+1, n+2}.
- Cost function: C(n, n+1) = 1, C(n, n+2) = 3.

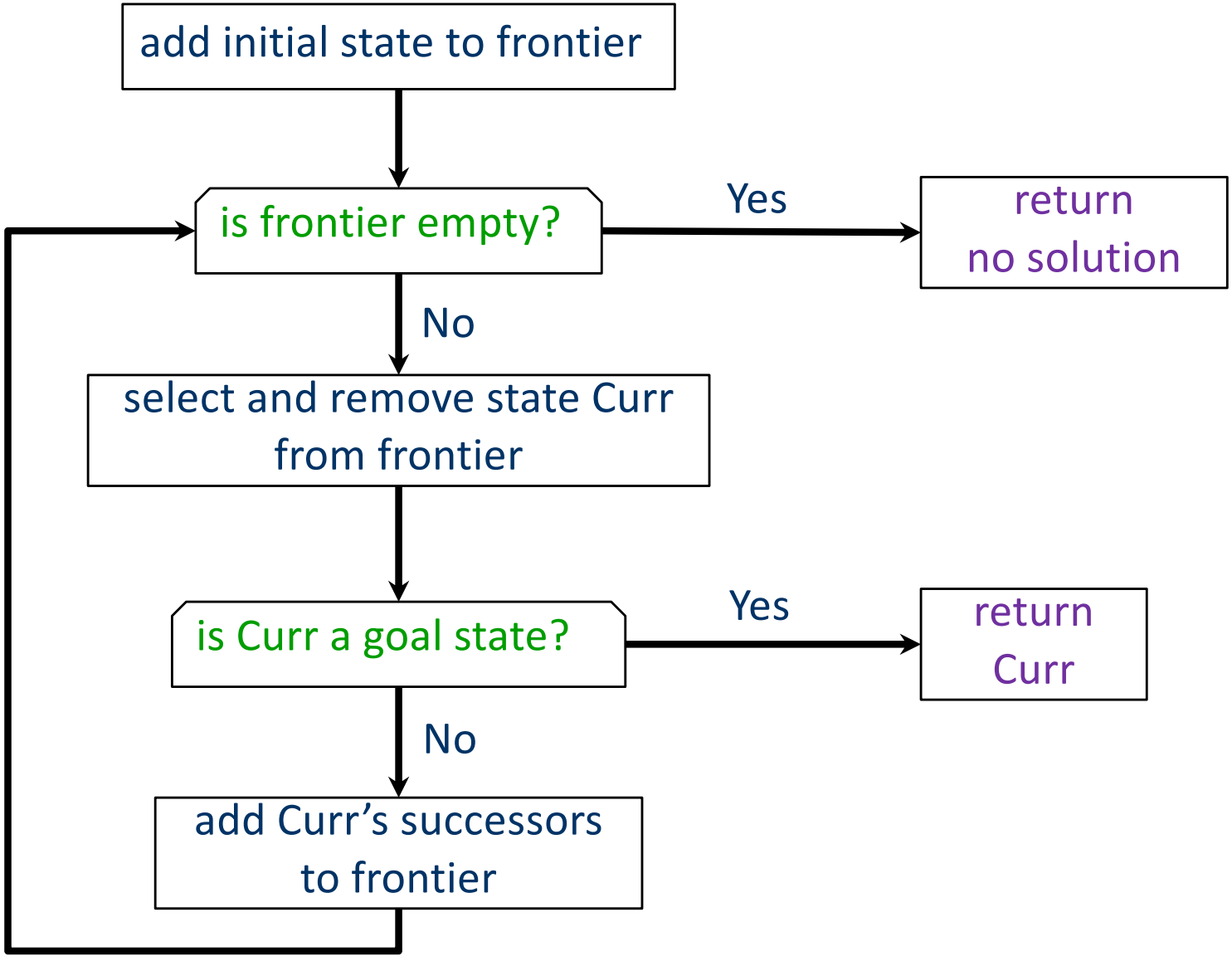
Search Graph for Integer Example



Generic Search Tree



Generic Search Algorithm as a Flowchart



Generic Search Algorithm

```
1 Search (Initial State, Successor Function, Goal Test)
2     Frontier = { Initial State }
3     While Frontier is not empty do
4         Select and remove state Curr from Frontier
5         If Curr is a goal state
6             return Curr
7         Add Curr's Successors to Frontier
8     Return no solution
```

Recovering the Path

How do we recover the path from the initial state if we return the goal node only?

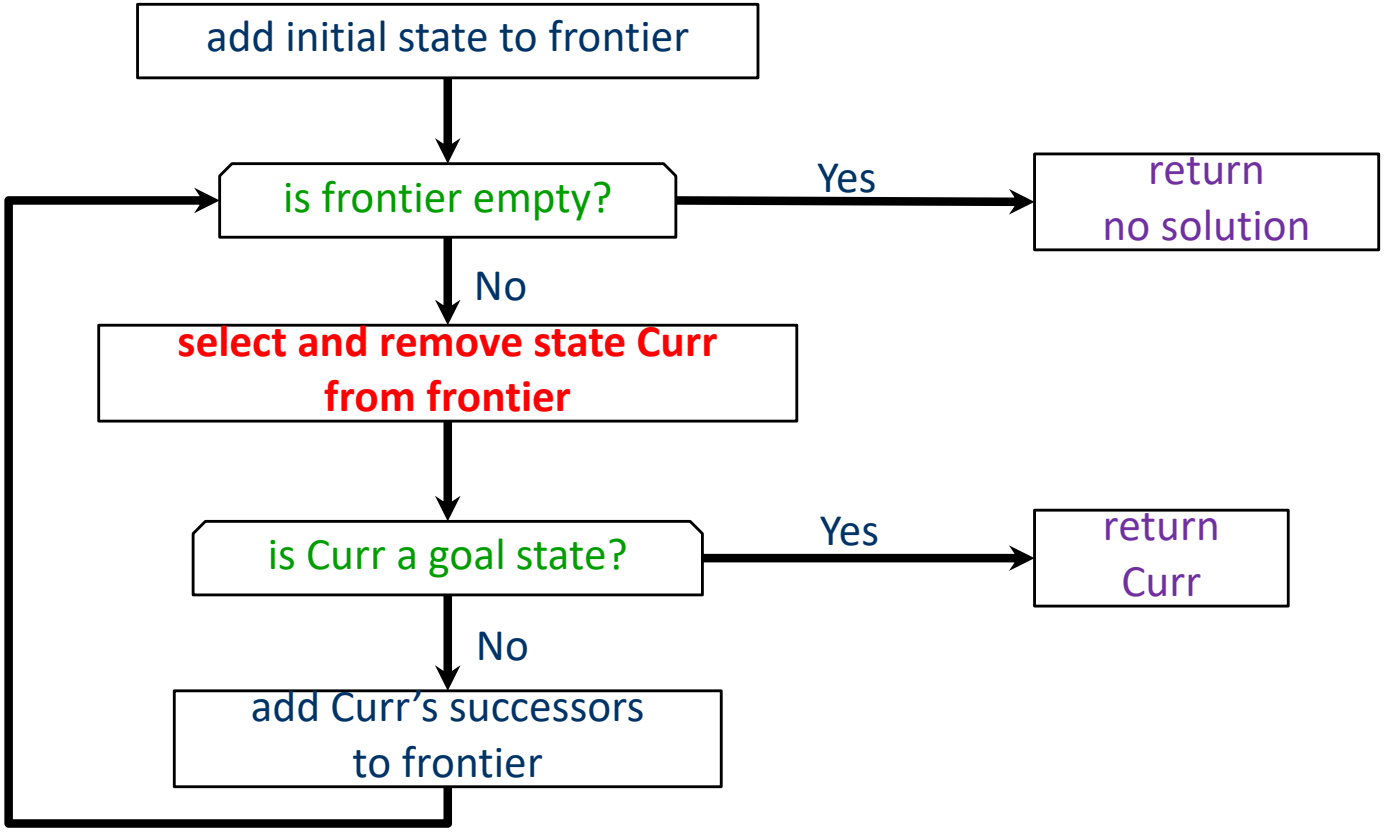
1. Back Tracking (Store every possible path in memory)
2. Store pointer to the prior state of the added frontiers

*1. let the frontier store the paths instead of the states

*2. let each node store a reference to its parent node

Uninformed Search Strategy

The order in which we remove nodes from the frontier determines our search strategy and its properties.



Uninformed Search Strategy

The order in which we remove nodes from the frontier determines our search strategy and its properties.

```
1 Search (Initial State, Successor Function, Goal Test)
2     Frontier = { Initial State }
3     While Frontier is not empty do
4         Select and remove state Curr from Frontier
5         If Curr is a goal state
6             return Curr
7         Add Curr's Successors to Frontier
8     Return no solution
```

Search Strategies

`Select and remove state Curr from Frontier`

Uninformed Search:

- DFS: remove newest state added
- BFS: remove oldest state added
- UCS: remove state with smallest total path cost

Heuristic Search:

- GBFS: remove state with smallest heuristic value
- A*: remove state with smallest (path cost + heuristic).

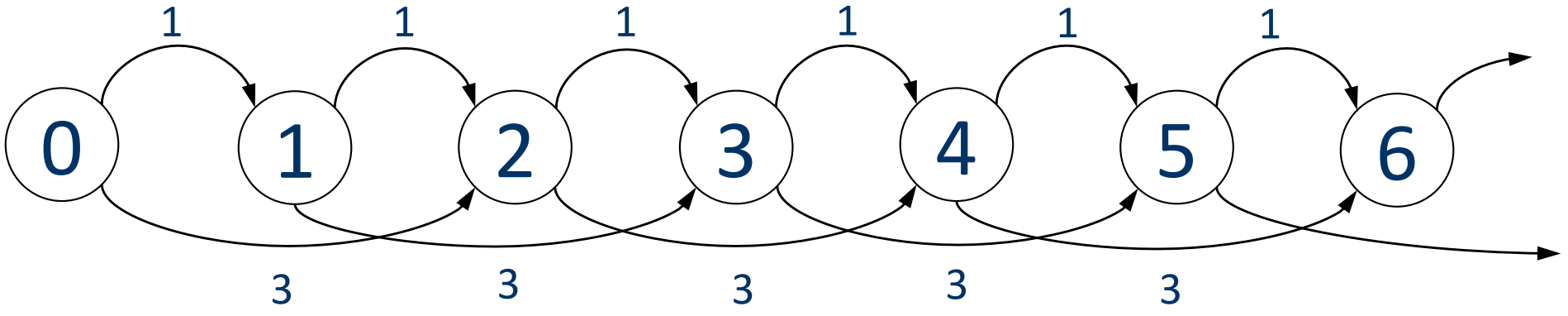
DEPTH-FIRST SEARCH

Depth-First Search

- The frontier is a stack (LIFO).
- Remove the most recent state added to the frontier.

Integer Example

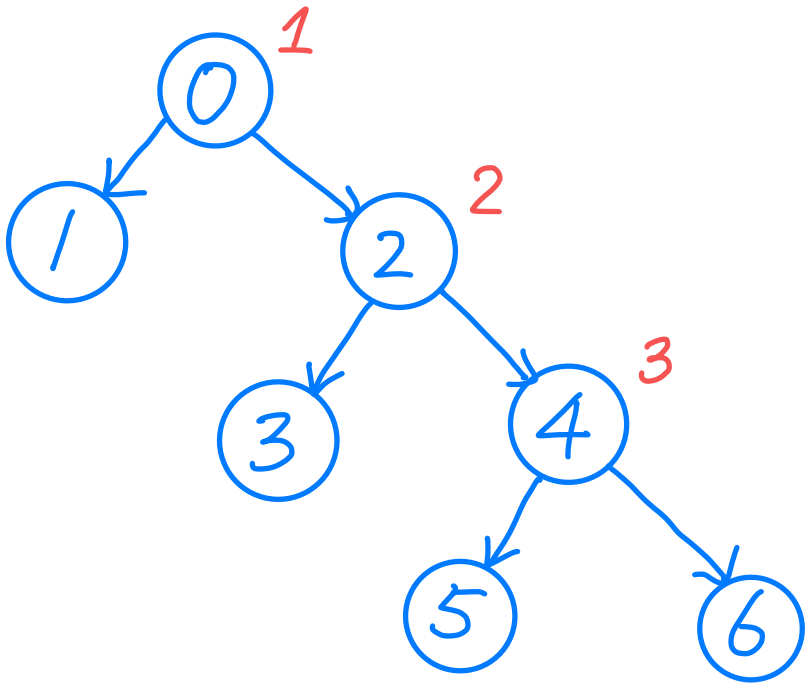
- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.



DFS on Integer Example

frontier: ~~0~~ (1), 01, ~~02~~ (2), 023, ~~024~~ (3),

search tree



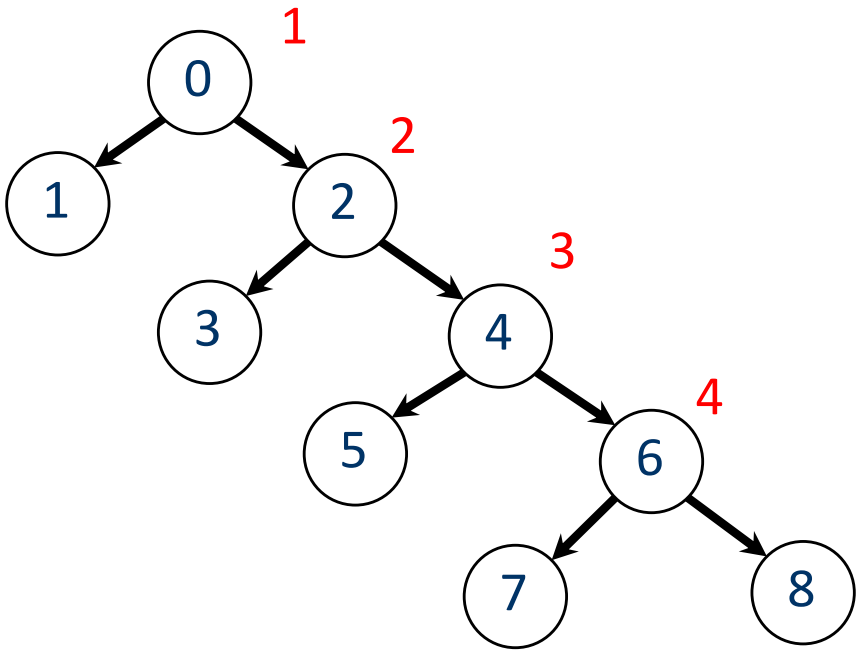
Solution: DFS on Integer Example

- add successors to frontier in order of n+1 and n+2.
- label nodes in tree in the order of expansion.

We "expand" a node whenever we remove it from the frontier.

frontier: ~~0~~, 1, ~~2~~, 3, ~~4~~, 5, ~~6~~, 7, 8

search tree



Outcome of executing DFS on Integer example

What is the outcome of

executing DFS on Integer example?

A. DFS finds a solution.

B. DFS finds the solution with the smallest total cost.

C. DFS does not terminate.

Properties of Search Algorithms

Completeness:

Will the search always find a solution if a solution exists?

Optimality:

Will the search always find the least-cost solution?

Time complexity:

What is the maximum number of nodes that we must expand?

Space complexity:

What is the maximum number of nodes that we must store in memory?

Useful Quantities

b

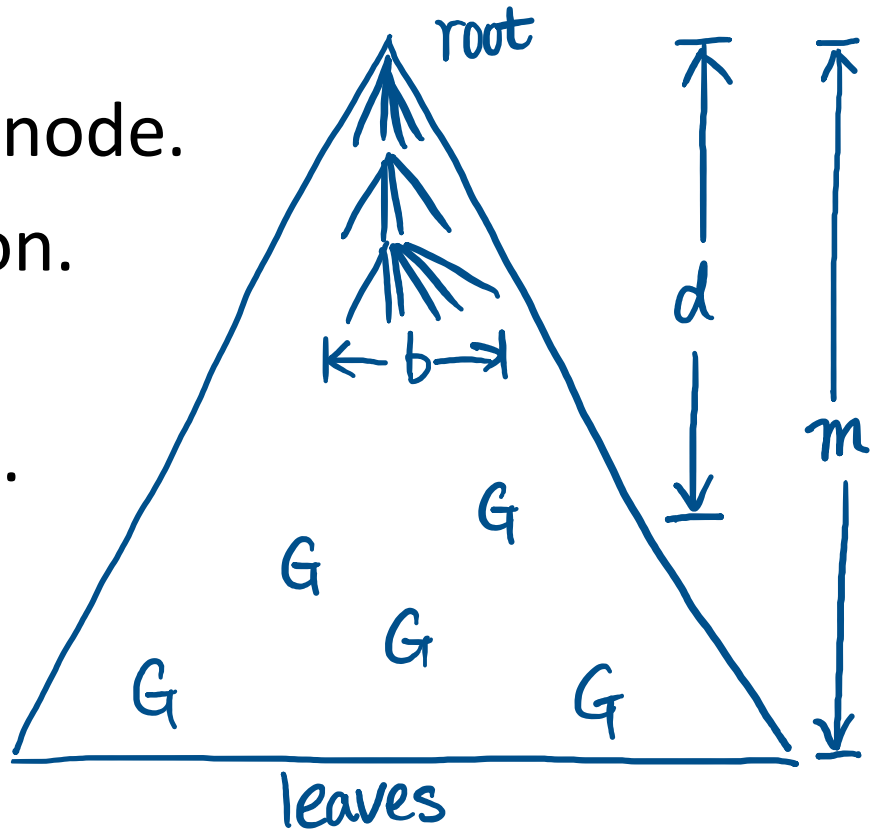
- Branching factor.
- Maximum number of successors of any state.

d

- Depth of the shallowest goal node.
- Length of the shortest solution.

m

- Max depth of the search tree.
- Length of the longest path.



DFS Properties – Completeness

Is DFS guaranteed to find a solution if a solution exists?

- No.
- Gets stuck in an infinite path.
  - E.g., integer example.
  - E.g., the search graph has a cycle.

DFS Properties – Optimality

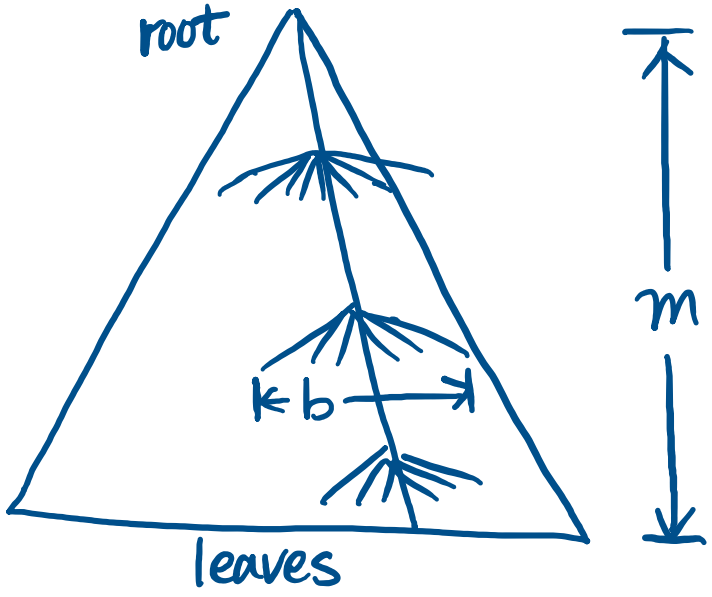
Is DFS guaranteed to find an optimal solution?

- No.
- Not optimizing costs.

DFS Properties – Space Complexity

$$O(bm)$$

- Linear in m.
- Explores a single path at a time.
- Remembers at most m nodes on the current path and at most b siblings for each node.

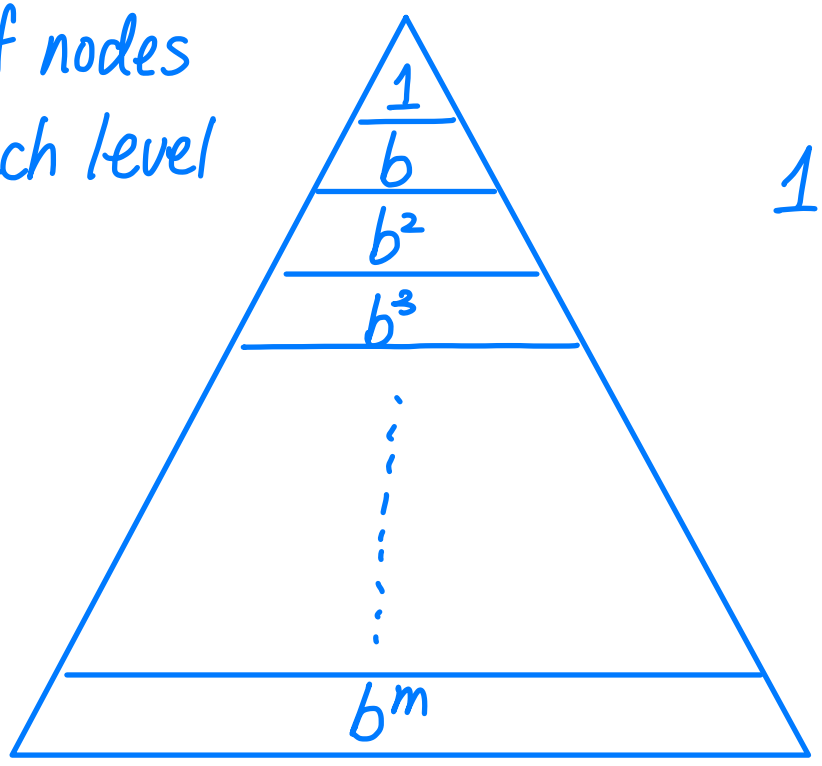


DFS Properties – Time Complexity

$$O(b^m)$$

- Exponential in m.
- Visits the entire search tree in the worst case.

# of nodes on each level



$1 + b + b^2 + b^3 + \ldots + b^m = O(b^m)$

Summary of Uninformed Search Algorithms

	DFS	BFS	IDS	UCS
Strategy	Remove newest state added			
Complete?	No			
Optimal?	No			
Space complexity	Linear in m $O(bm)$			
Time complexity	Exponential in m $O(b^m)$			

Question

What are some advantages of DFS?

A. It is complete.
B. It is optimal.
C. It has great space complexity.
D. It has great time complexity.

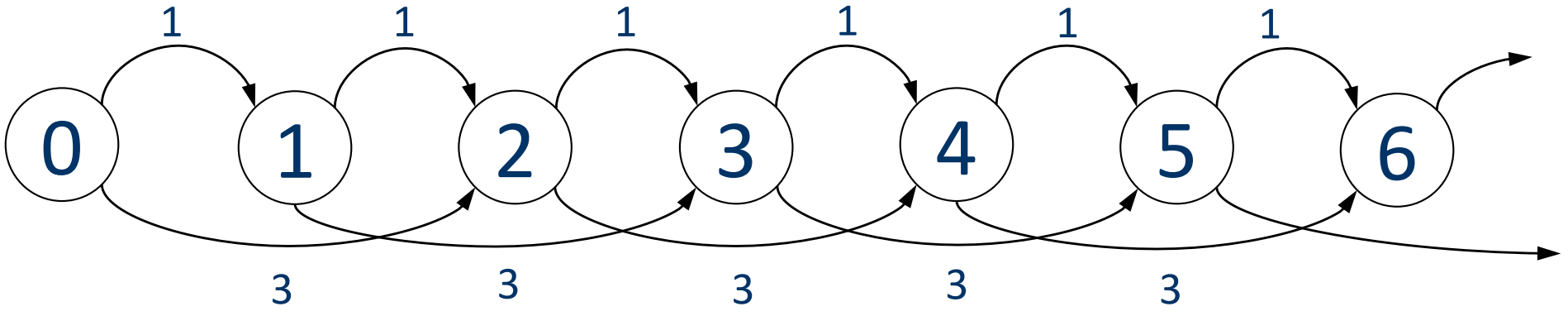
BREADTH-FIRST SEARCH

Breadth-First Search

- The frontier is queue (FIFO).
- Remove the oldest state added to the frontier.

Integer Example

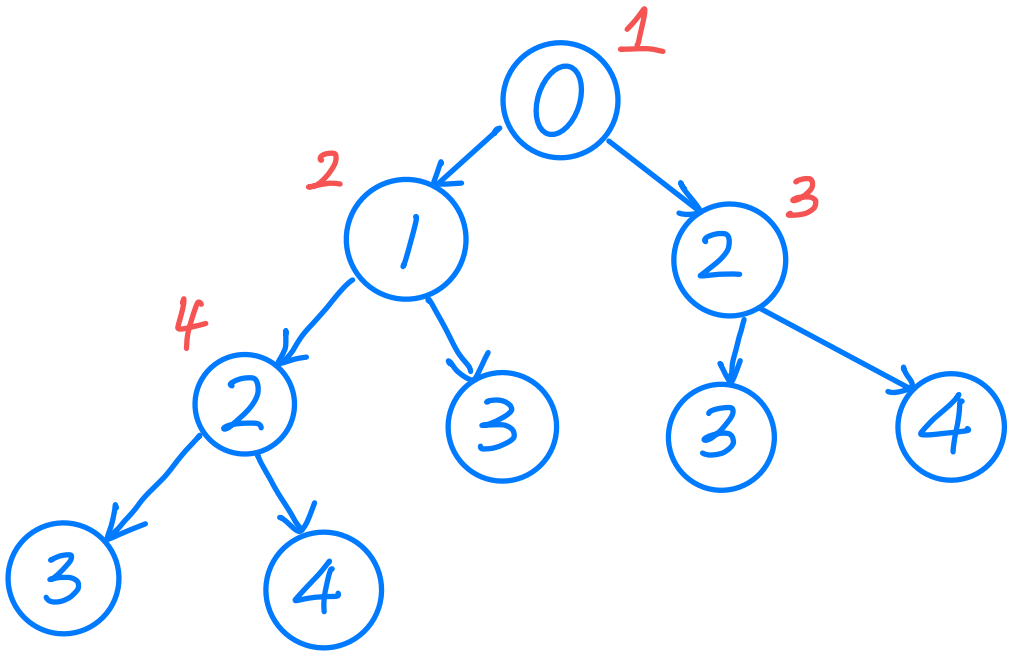
- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.



BFS on Integer Example

frontier: 0, 01, 02, 012, 013, 023, 024, 0123, 0124,

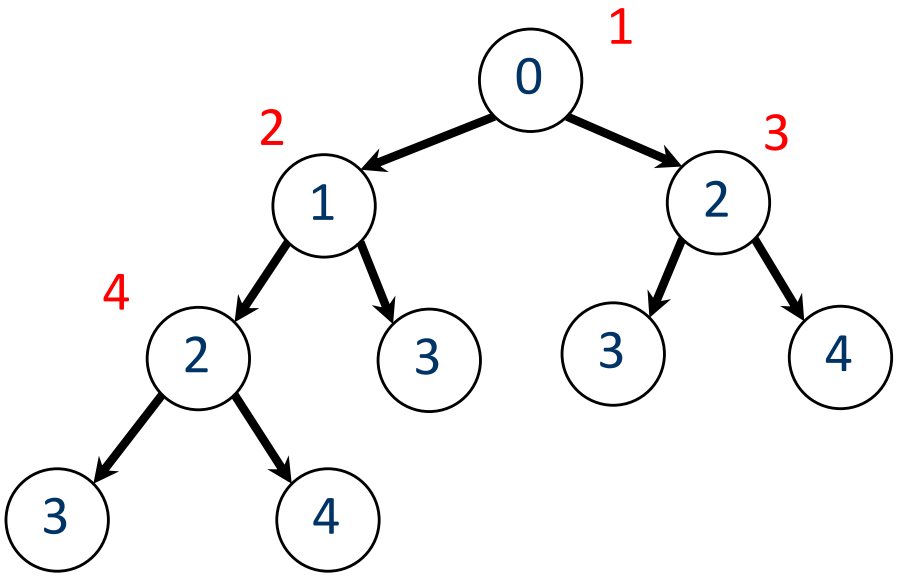
search tree



Solution: BFS on Integer Example

frontier: ~~0~~, ~~01~~, ~~02~~, ~~012~~, 013, 023, 024,

search tree



BFS Properties – Completeness

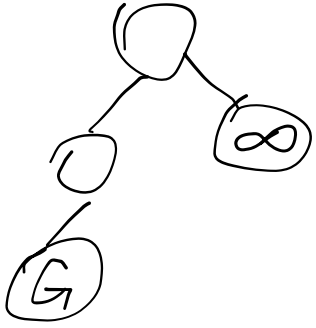
Is BFS guaranteed to find a solution if a solution exists?

- Yes.
- Guaranteed to find shallowest goal node at depth d.

BFS Properties – Optimality

Is BFS guaranteed to find an optimal solution?

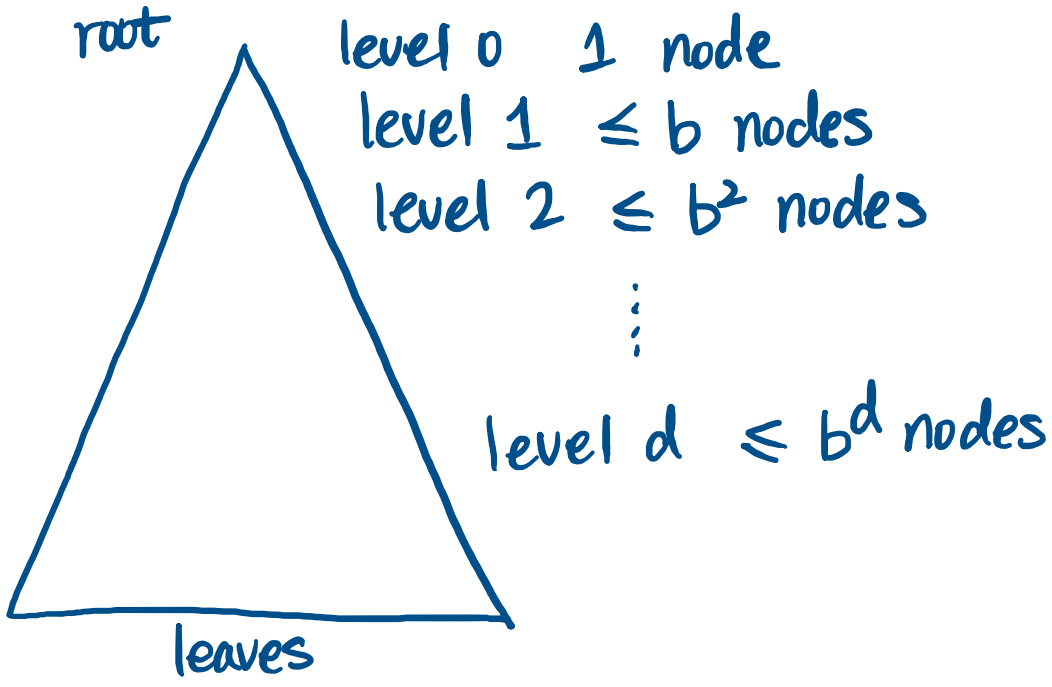
- No.
- Not optimizing costs.
- Guaranteed to find shallowest goal node at depth d.



BFS Properties – Time Complexity

$$O(b^d)$$

- Exponential in d.
- Must visit the top d levels.
- Frontier contains at most b^d nodes (on level d).



Total # of nodes explored:

$1 + b + b^2 + b^3 + \cdots + b^d$

$= O(b^d)$

BFS Properties – Space Complexity

$$O(b^d)$$

- Exponential in d.
- Visit the top d levels in the worst case.
- Total # of nodes dominated by # of nodes on level d.

Summary of Uninformed Search Algorithms

	DFS	BFS	IDS	UCS
Strategy	Remove newest state added	Remove oldest state added		
Complete?	No	Yes		
Optimal?	No	No		
Space complexity	Linear in m $O(bm)$	Exponential in d $O(b^d)$		
Time complexity	Exponential in m $O(b^m)$	Exponential in d $O(b^d)$		

Question 1: DFS v.s. BFS

Which search algorithm would you use
if the search graph has cycles?

A. I would use DFS but not BFS.
B. I would use BFS but not DFS.
C. Both DFS and BFS are appropriate.
D. Neither DFS nor BFS is appropriate.

A cycle in a search graph is an infinite path in a search tree. DFS will not terminate if there is an infinite path.

Question 2: DFS v.s. BFS

Which search algorithm would you use if the branching factor is infinite?

A. I would use DFS but not BFS.

B. I would use BFS but not DFS.

C. Both DFS and BFS are appropriate.

D. Neither DFS nor BFS is appropriate.

BFS will not terminate if the branching factor is infinite.

ITERATIVE-DEEPENING SEARCH

DFS versus BFS

Can we have a search algorithm that combines the best of DFS and BFS?

	Depth-First Search	**Breadth-First Search**
Space complexity	Linear in m $O(bm)$ 😄	Exponential in d $O(b^d)$
Completeness	May get stuck on an infinite path and never terminate.	Guaranteed to find a solution if it exists. 😄

Depth-Limited Search

Problem:

- DFS may follow an infinite path forever.

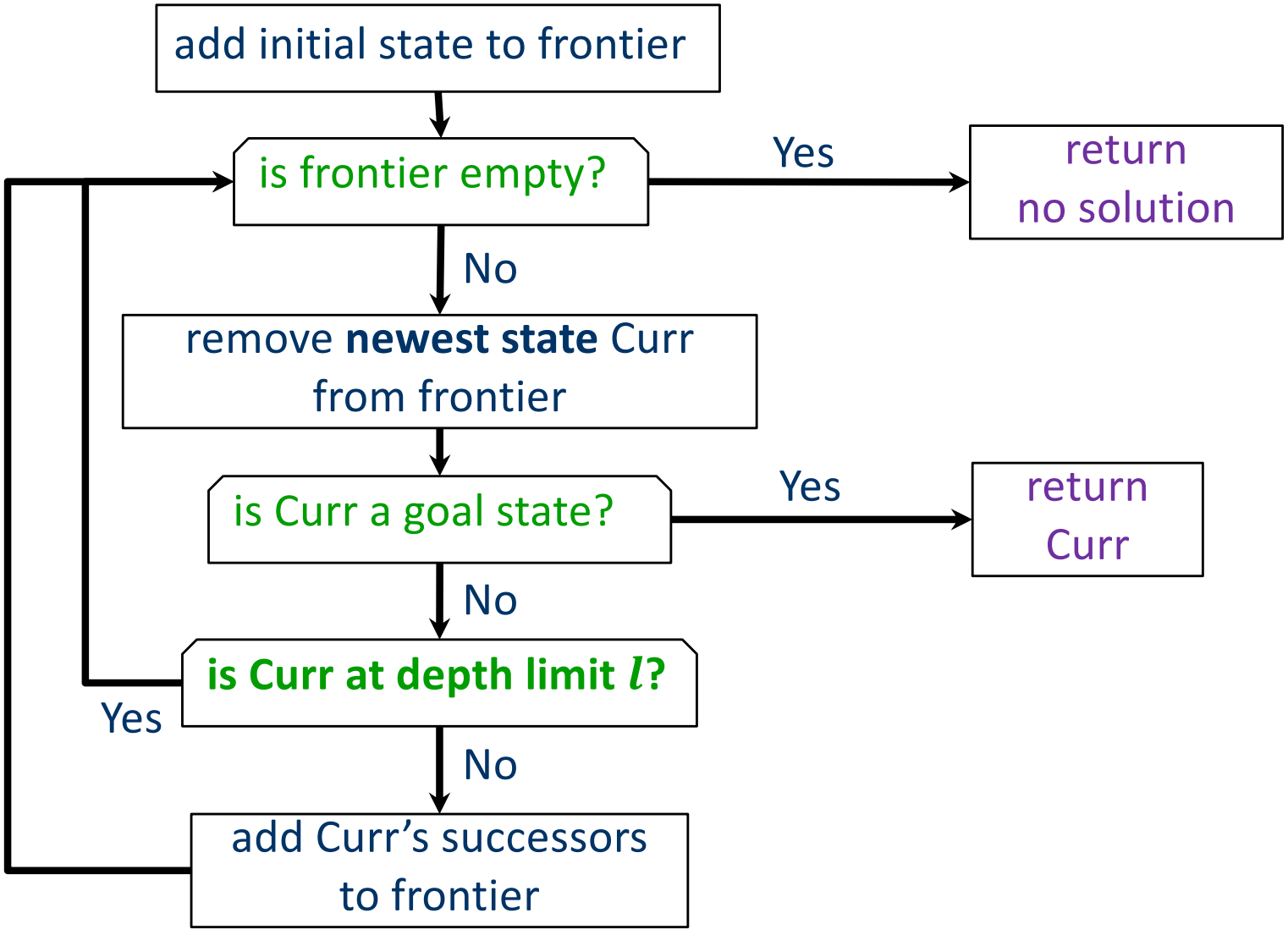
Idea:

- BFS is complete since it explores the tree level by level.

Solution:

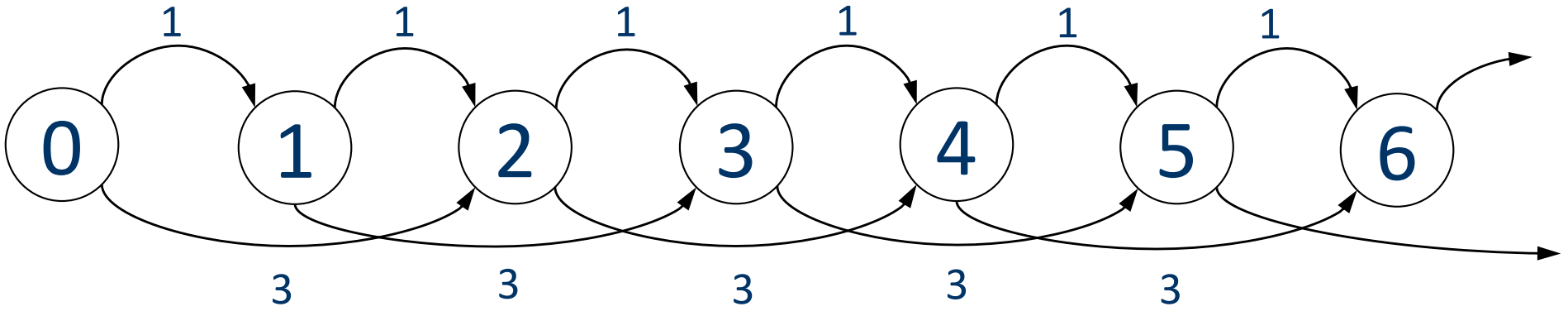
- Perform DFS up to a pre-specified depth limit l.

Depth-Limited Search as Flowchart



Integer Example

- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.

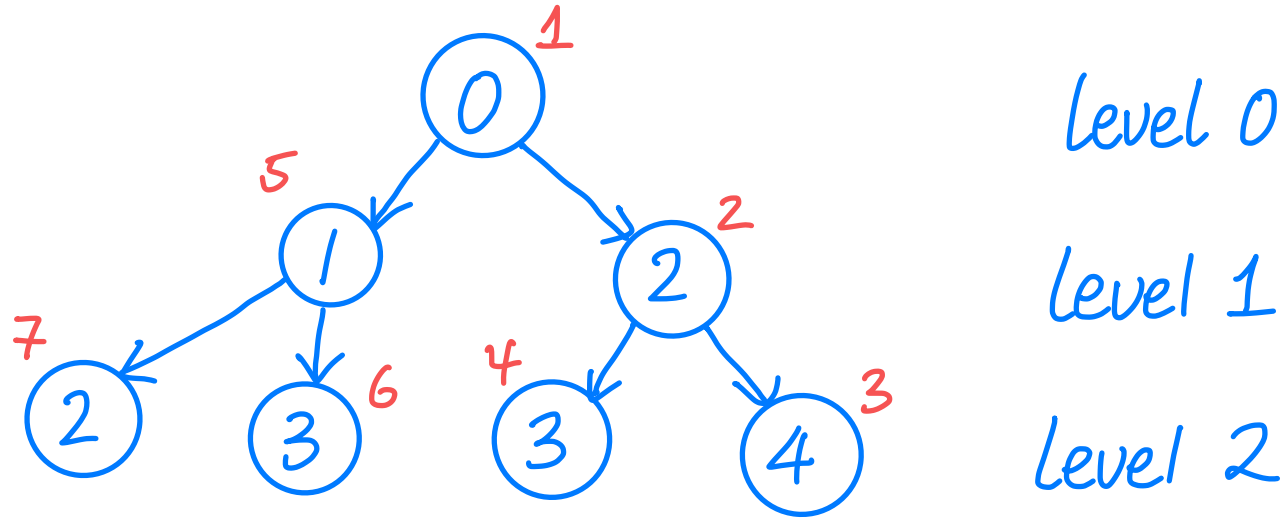


DLS on Integer Example

Limit = 2

frontier: 0, 01, 02, 023, 024, 012, 013,

search tree

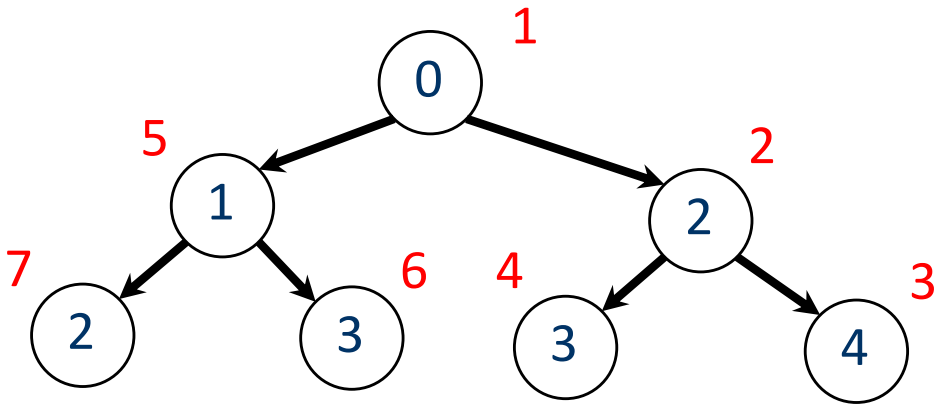


DLS on Integer Example

Limit = 2

frontier: ~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~2~~, ~~3~~

search tree



level 0

level 1

level 2

Depth-Limited Search

Desirable property:

- DLS no longer gets stuck on an infinite path.

Problem:

- If no goal node exists within the depth limit, DLS doesn't find a solution...

Solution: (If we do not succeed, try, try, try again...)

- If DLS doesn't find a solution w/ depth limit l, try again with depth limit $l + 1$.

Iterative-Deepening Search

- For depth limit 0 to ∞, perform depth-limited search.
- Depth-limited search:
  - When the node is goal, return the solution.
  - When depth reaches the limit, return without generating and adding successors.
  - Otherwise, add successors and continue the search.

IDS on Integer Example

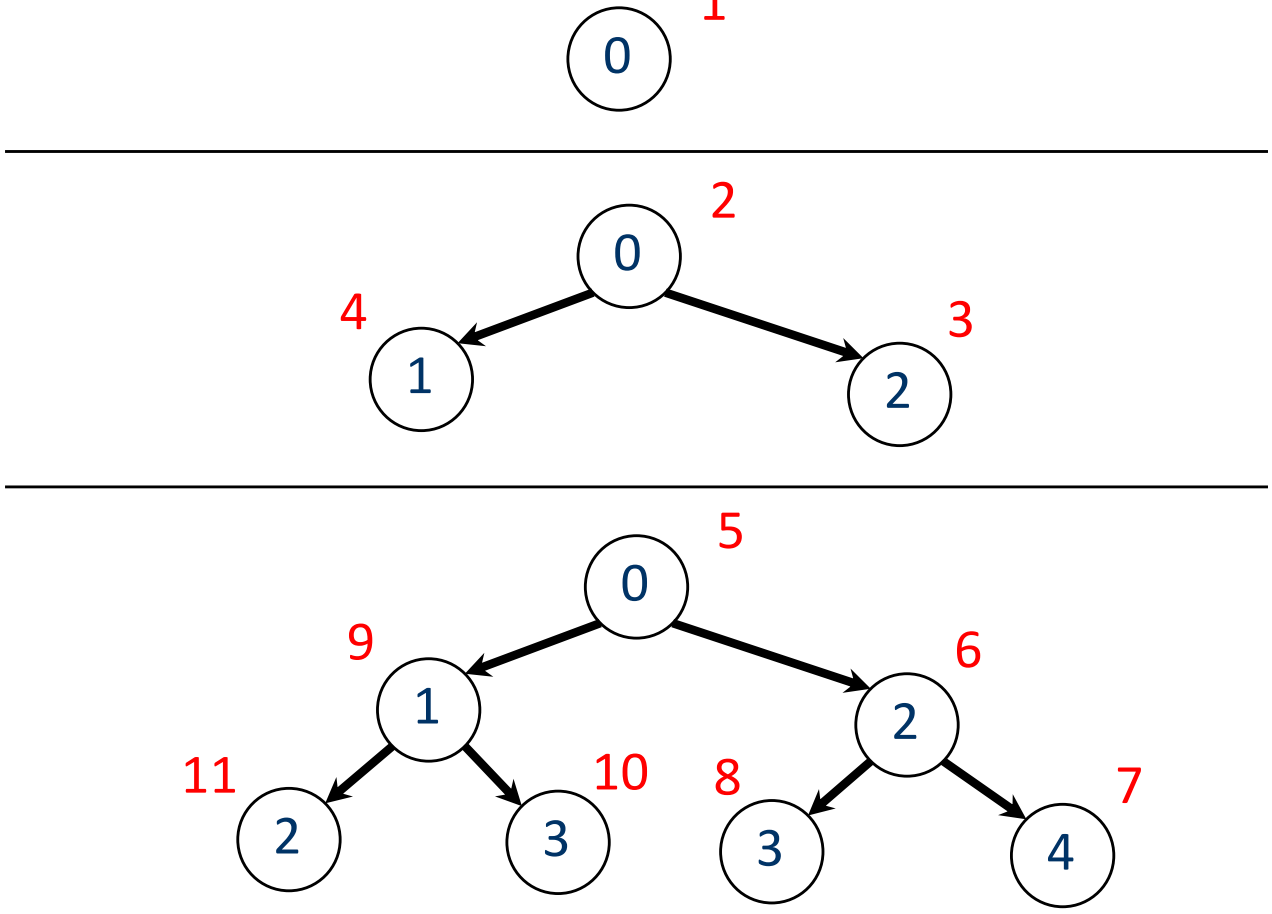
limit = ~~0~~, ~~1~~, ~~2~~

frontier: ~~0~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~2~~, ~~3~~

1 2 4 3 5 9 6 8 7 11 10

0, 0, 01, 02, 0, 01, 02, 023, 024, 012, 013

search tree(s)



Why restarting DLS from scratch?

Every time the depth limit l increases,

we start Depth-Limited Search from scratch ($l = 0$).

When performing DLS for depth limit l,

can we reuse the work done for depth limit $l - 1$?

No, DLS does not remember the states on the bottom level for the previous depth limit. If we use an additional data structure to remember these states, the space complexity becomes exponential instead of linear.

IDS Properties – Completeness

Is IDS guaranteed to find a solution if a solution exists?

A. Yes, IDS is complete.

B. No, IDS is not complete.

IDS Properties – Completeness

Is IDS guaranteed to find a solution if a solution exists?

- Yes.
- IDS is guaranteed to terminate at level d.
- Explores the tree level by level ($\approx$ BFS).

IDS Properties – Optimality

Is IDS guaranteed to find an optimal solution?

- No.
- Not optimizing costs.
- Guaranteed to find shallowest goal node (≈ BFS).

IDS Properties – Space Complexity

What is the space complexity for IDS?

≈ DFS ≈ BFS

A. Linear in d.
B. Linear in m.
C. Exponential in d.
D. Exponential in m.

IDS Properties – Space Complexity

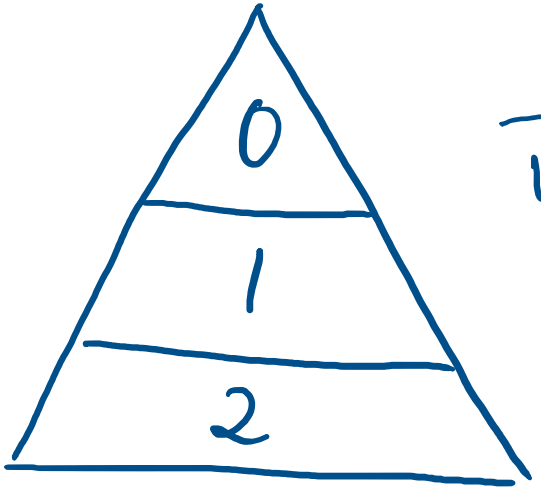
$$O(bd)$$

- Linear in d ($\approx$ DFS). 🤩
- Explores one path of length at most d.
- Remembers at most b siblings for each node.

IDS Properties – Time Complexity

$$O(b^d)$$

- Exponential in d.
- What about the repeated computations?
  - States in upper levels are generated multiple times.
  - Wasteful? Most of the states are in the bottom level.



depth limit	0	1	2
visits level 0	✓	✓	✓
level 1		✓	✓
level 2			✓

Summary of Uninformed Search Algorithms

	DFS	BFS	IDS	UCS
Strategy	Remove newest state added	Remove oldest state added	Remove newest state added	
Complete?	No	Yes	Yes	
Optimal?	No	No	No	
Space complexity	Linear in m $O(bm)$	Exponential in d $O(b^d)$	Linear in d $O(bd)$	
Time complexity	Exponential in m $O(b^m)$	Exponential in d $O(b^d)$	Exponential in d $O(b^d)$	

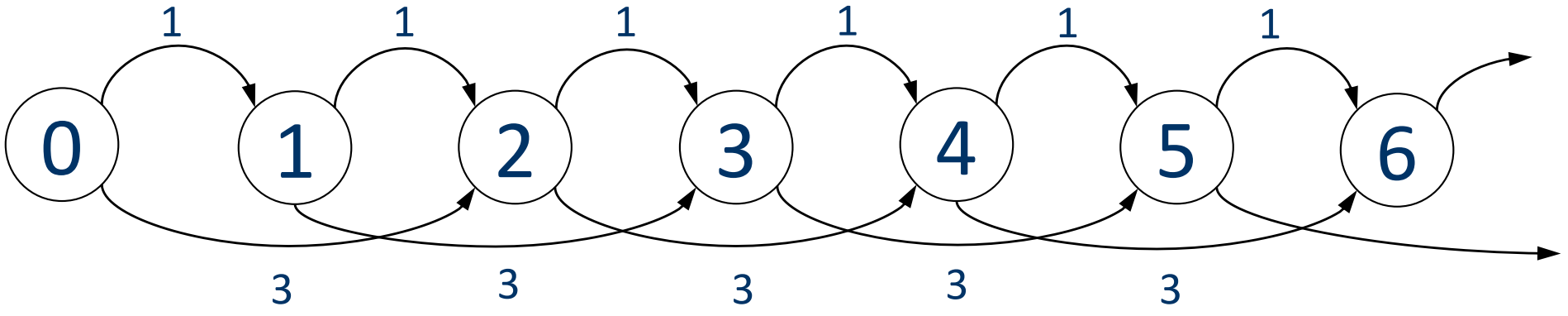
UNIFORM-COST SEARCH

Uniform-Cost Search

- The frontier is a priority queue ordered by path cost.
- Remove the least-cost path in the frontier.
- Also known as Dijkstra's shortest path algorithm.

Integer Example

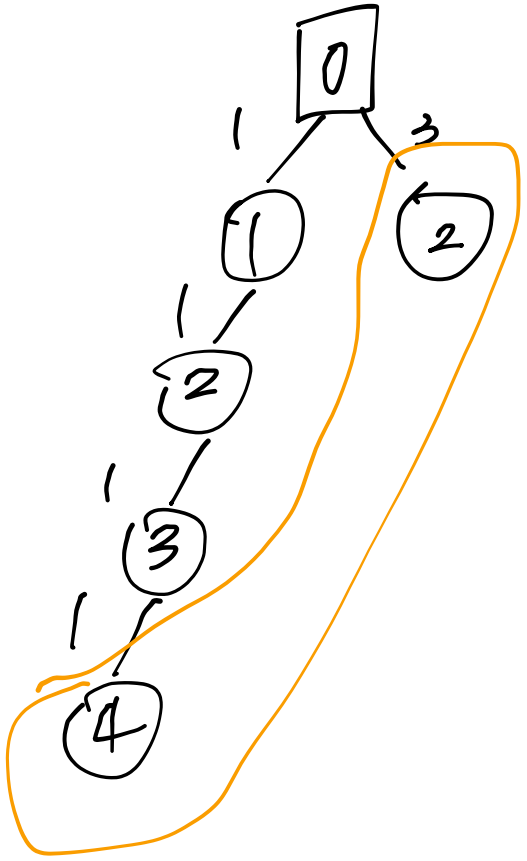
- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.



UCS on Integer Example

frontier:

search tree



UCS on Integer Example

tie-breaking rule: remove oldest state on the frontier.

frontier:

1 2 4 3 6 5 9 7 8

0, 01, 02, 012, 013, 0123, 0124, 023, 024, 01234,

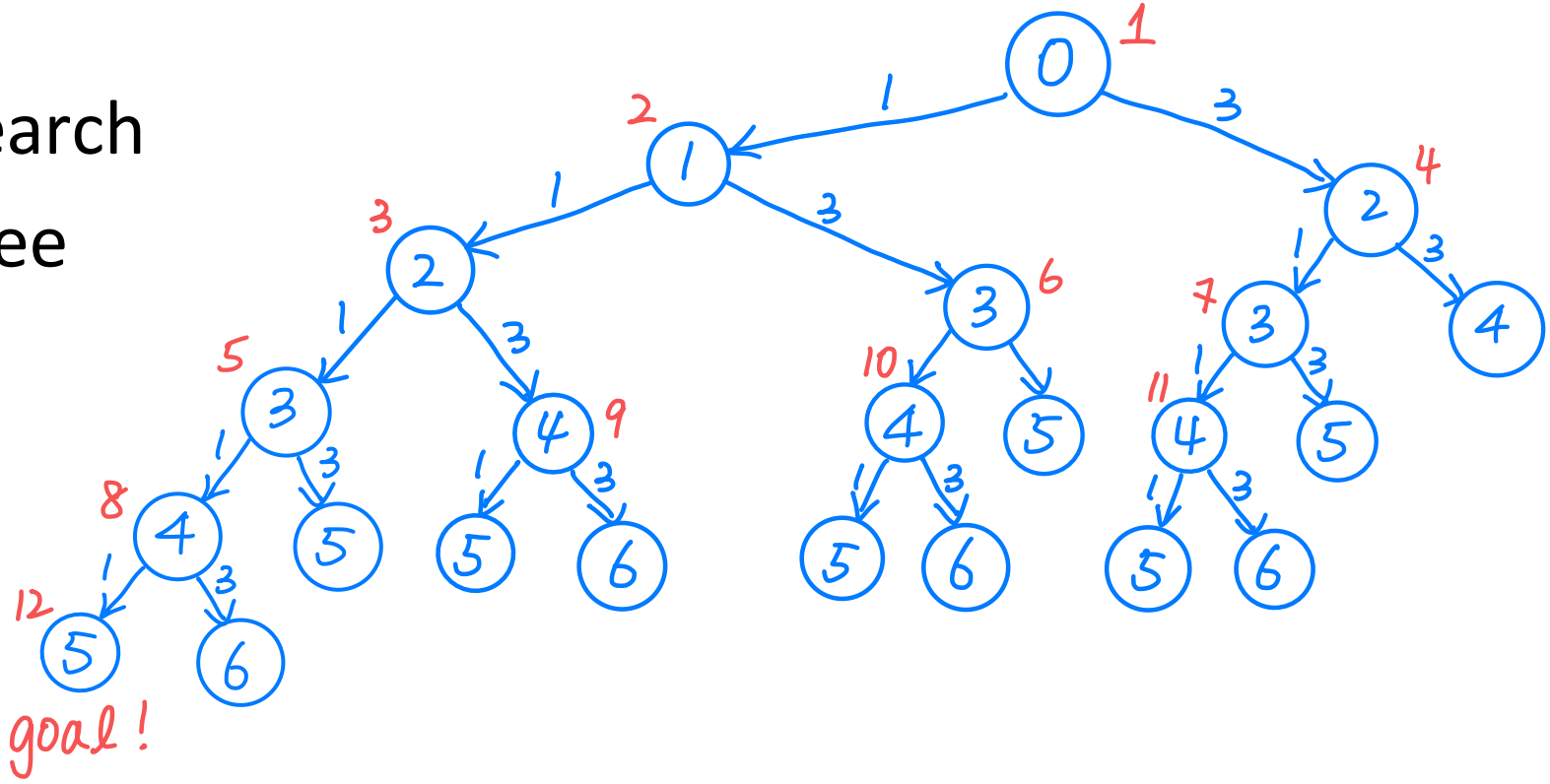
0 1 3 2 4 3 5 4 6 4

10 11 12

01235, 0134, 0135, 0234, 0235, 01245, 012346,

6 5 7 5 7 5 7

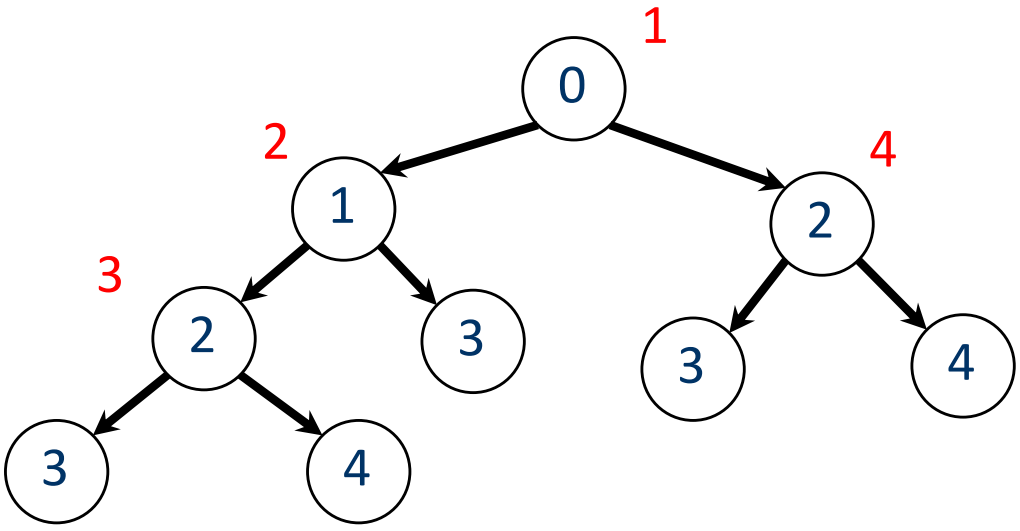
search tree



UCS on Integer Example

frontier: 0 (0), 01 (1), 02 (3), 012 (2), 013 (4), 0123 (3), 0124 (5), 023 (4), 024 (6)

search tree



UCS Properties – Completeness and Optimality

Is UCS guaranteed to find a solution if a solution exists?

Is UCS guaranteed to find an optimal solution?

- Yes, under mild conditions.

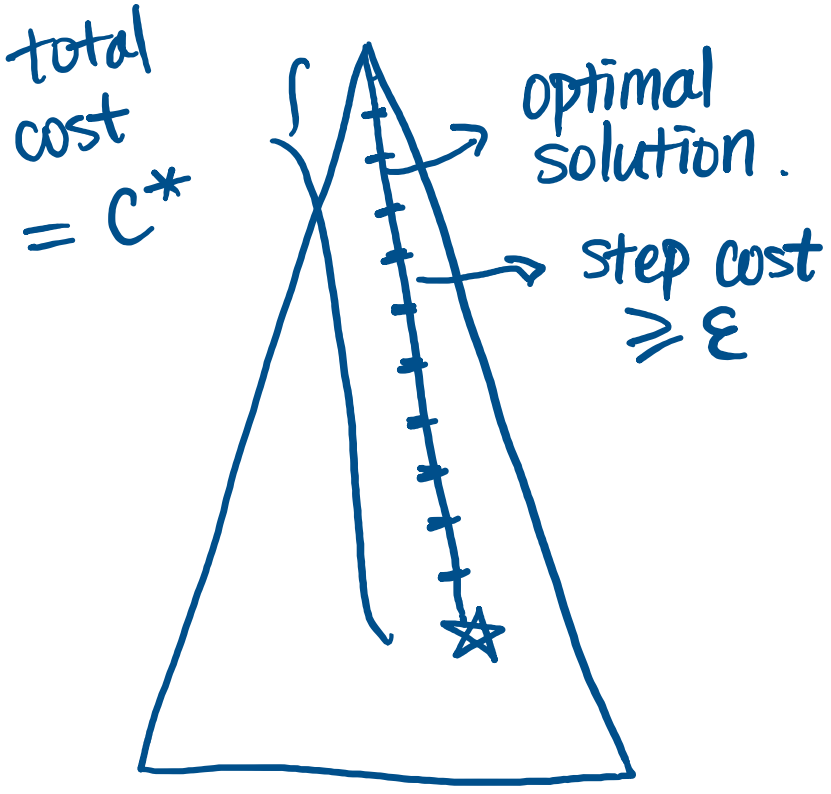
Mild conditions:

- The branching factor b is finite.
- The cost of every action is bounded below by a positive constant ($\varepsilon > 0$).

UCS Properties – Time and Space Complexity

$$O\left(b^{\frac{C^*}{\epsilon}}\right)$$

- C^* is the cost of the optimal solution.
- ε is the minimal cost of an action.



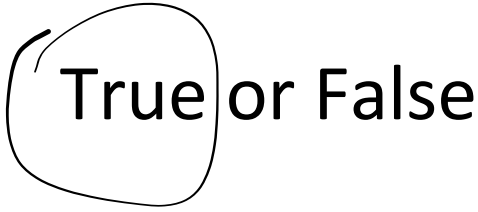
1. total cost
2. cost of each step
3. # of steps
4. level of optimal solution $\leq C^*/\varepsilon$.

Summary of Uninformed Search Algorithms

	DFS	BFS	IDS	UCS
Strategy	Remove newest state added	Remove oldest state added	Remove newest state added	Remove state with smallest path cost
Complete?	No	Yes	Yes	Yes
Optimal?	No	No	No	Yes
Space complexity	Linear in m $O(bm)$	Exponential in d $O(b^d)$	Linear in d $O(bd)$	$O(b^{\frac{C^*}{e}})$
Time complexity	Exponential in m $O(b^m)$	Exponential in d $O(b^d)$	Exponential in d $O(b^d)$	$O(b^{\frac{C^*}{e}})$

Question

If each edge has the same cost, UCS and BFS are identical.



MULTIPLE-PATH PRUNING

What is Multiple-Path Pruning?

Problem:

- There can be multiple paths to the same state
- We only need to find one (or the best) path to a state.

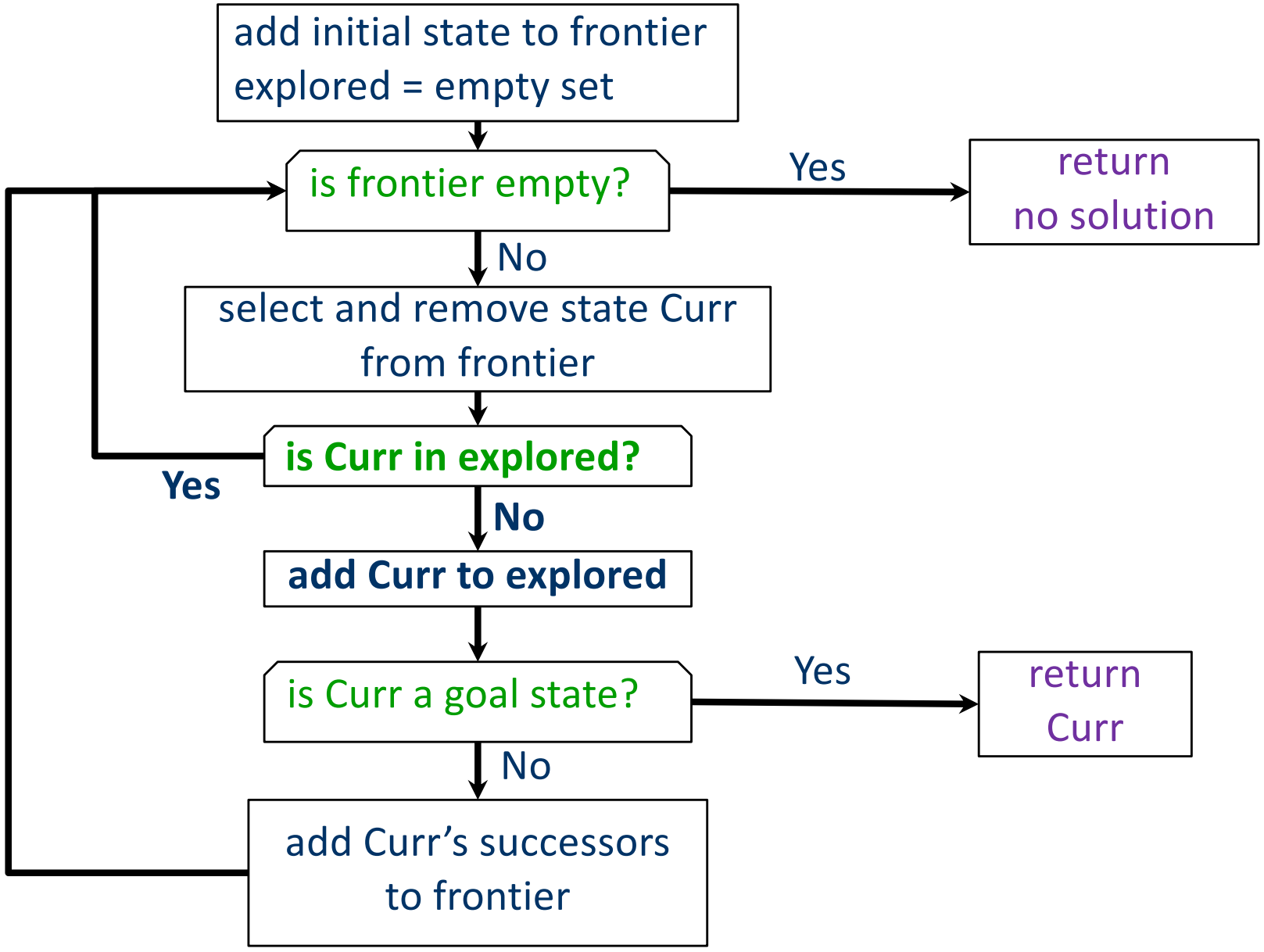
Solution:

- Keep only one path to each state.

Key ideas:

- Remembers all the expanded states in a set.
- Expand a node only if it's not in the set.

Generic Search Algorithm with Pruning



Generic Search Algorithm **with Pruning**

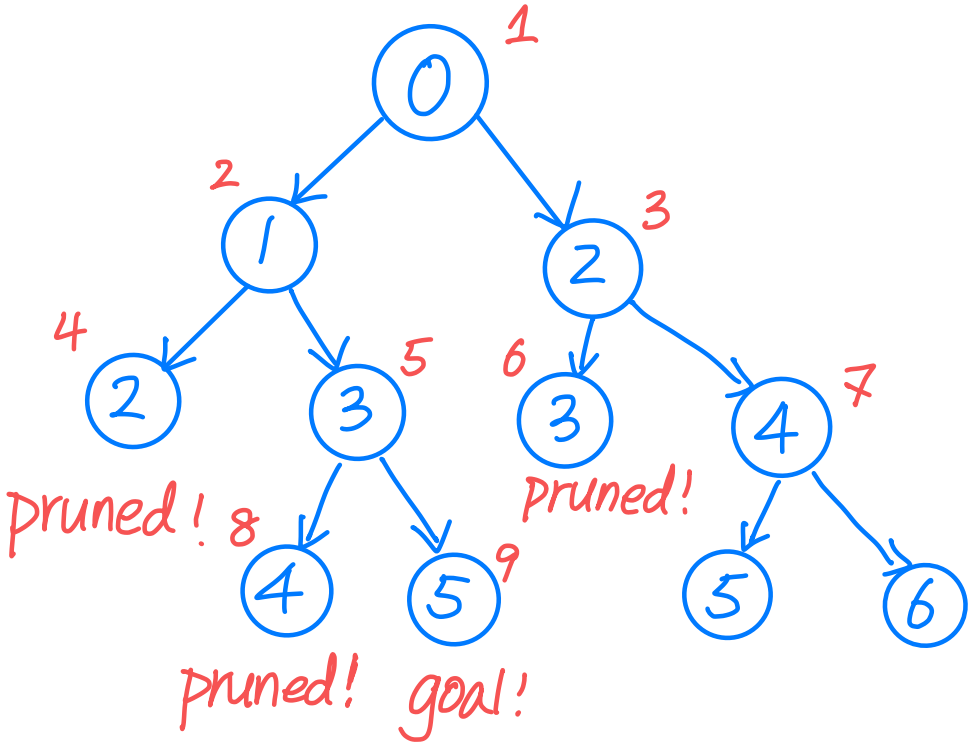
```
1  Search (Initial State, Successor Function, Goal Test)
2     Frontier = { Initial State }
3     Explored = { }
4     While Frontier is not empty do
5        Select and remove state Curr from Frontier
6        If Curr is NOT in Explored
7           Add Curr to Explored
8           If Curr is a goal state
9              return Curr
10          Add Curr's Successors to Frontier
11    Return no solution
```

BFS with Pruning on Integer Example

frontier: ~~0~~, ~~01~~, ~~02~~, ~~012~~, ~~013~~, ~~023~~, ~~024~~, ~~0134~~, ~~0135~~, 0245, 0246,

explored: 0, 1, 2, 3, 4, 5,

search tree

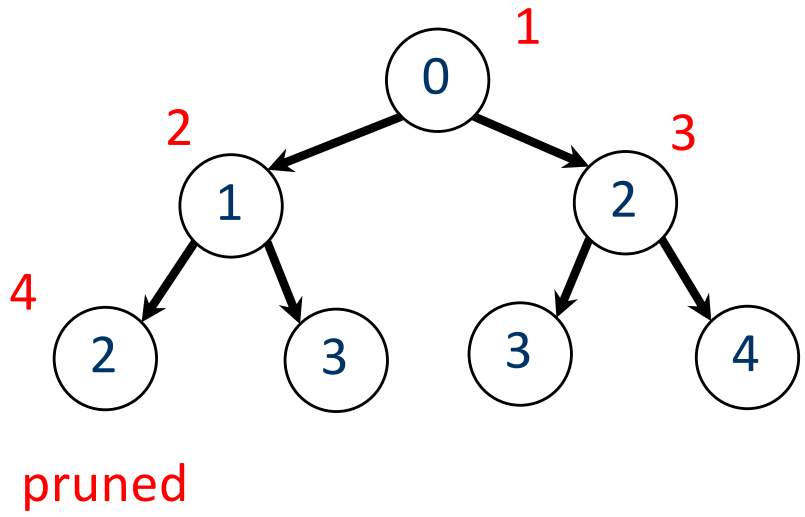


BFS with Pruning on Integer Example

frontier: 0, 01, 02, 012, 013, 023, 024

explored: 0, 1, 2

search tree



Question

How does Multi-Path Pruning

affect the space complexity of DFS?

A. No effect

B. Pruning changes the space complexity of DFS from linear to exponential.

C. Pruning changes the space complexity of DFS from exponential to linear.

Effects on Space Complexity

- Multiple-path pruning incurs high space complexity.

- Combining with BFS?
  - w/o pruning, BFS has exponential space complexity.
  - w/ pruning, BFS still has exponential space complexity.

- Combining with DFS?
  - w/o pruning, DFS has linear space complexity.
  - w/ pruning, DFS has exponential space complexity.

Effects on Optimality

Will UCS with multi-path pruning

find an optimal solution?

- Yes.
- The first path to each state found by UCS must be the lowest-cost path to the state.
- The pruned paths cannot have lower costs than the first path UCS found.

This no longer holds for some heuristic search algorithms.

CSC 384
Introduction to Artificial Intelligence

Heuristic Search

Alice Gao and Randy Hickey

Winter 2023

Learning Goals

By the end of this lecture, you should be able to

- Describe motivations for applying heuristic search algorithms.
- Trace the execution of and implement Greedy best-first search and A* search algorithm.
- Describe properties of Greedy best-first and A* search algorithms.
- Design an admissible heuristic function for a search problem.
- Describe strategies for choosing among multiple heuristic functions.

Outline

- Why Heuristic Search
- Greedy Best-First Search
- A* Search
- Constructing Heuristics

WHY HEURISTIC SEARCH

Why Heuristic Search?

How would _____ choose which state to expand?

- Humans
- An uninformed search algorithm

5	3	
8	7	6
2	4	1

1	2	3
4	5	
7	8	6

Uninformed v.s. Heuristic Search

Uninformed Search

- Has no problem-specific knowledge.
- Treats all the states in the same way.
- Does not know which state is closer to a goal.

Heuristic Search

- Has problem-specific knowledge, i.e., a heuristic.
- Considers some states to be more promising than others.
- Estimates which state is closer to a goal using the heuristic.

The Heuristic Function

Definition of a heuristic function:

A search heuristic $h(n)$ is an **estimate** of the cost of the **cheapest** path from node n to a goal node.

A good $h(n)$ has the properties below:

- Problem-specific.
- Non-negative.
- $h(n) = 0$ if n is a goal node.
- Can compute $h(n)$ easily without search.

Two Useful Functions

- Consider a state n.
- The cost function $g(n)$:
  - the cost of the path from the initial state to state n.
- The heuristic function $h(n)$:
  - the estimate of the cheapest path from state n to a goal state.

- UCS: remove the state w/ the lowest $g(n)$
- GBFS: remove the state w/ the lowest $h(n)$
- A*: remove the state w/ the lowest $f(n) = g(n) + h(n)$

GREEDY BEST-FIRST SEARCH

Greedy Best-First Search

- Frontier is a priority queue ordered by $h(n)$.
- Expands the node with the smallest $h(n)$.

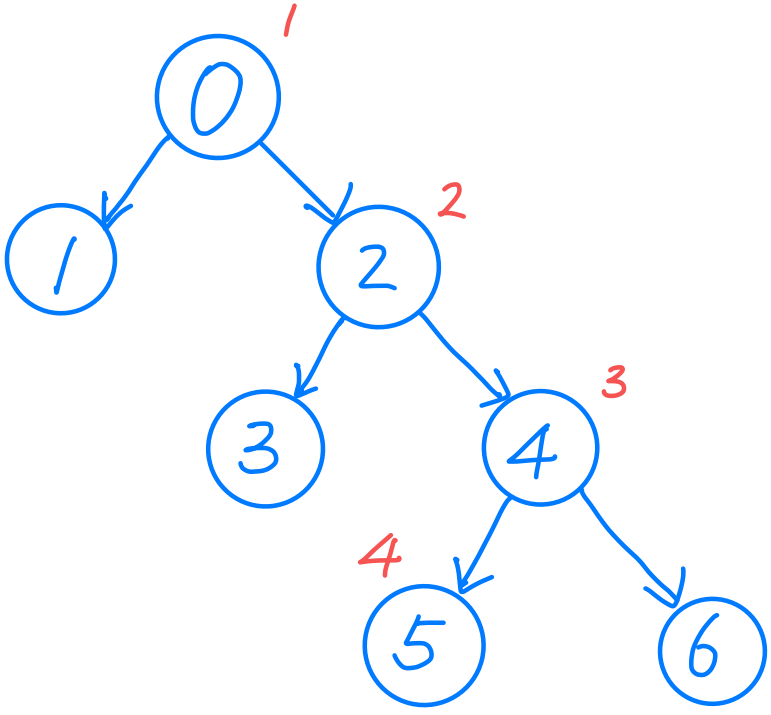
Integer Example

- States: the non-negative integers $\{0, 1, 2, \dots\}$.
- Initial state: 0.
- Goal state: 5.
- Successor function: $S(n) = \{n + 1, n + 2\}$.
- Cost function: $C(n, n + 1) = 1, C(n, n + 2) = 3$.
- Heuristic function:
  - If $n \leq 5$, $h(n) = 5 - n$.
  - Otherwise, $h(n) = \infty$.

GBFS on Integer Example

fronter: 0 (1), 01 (5), 02 (2), 023 (4), 024 (3), 0245 (0), 0246 (∞)

search tree



solution found 0245

cost = 3 + 3 + 1 = 7

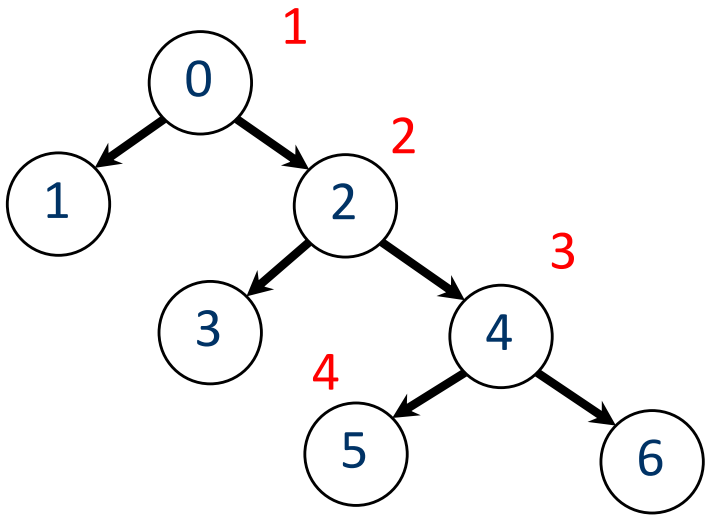
goal found!

GBFS on Integer Example

fronter: ~~0(5)~~, 01(4), ~~02(3)~~, 023(2), ~~024(1)~~, ~~0245(0)~~, 0246(∞)

Returns solution 0245 with total cost 7.

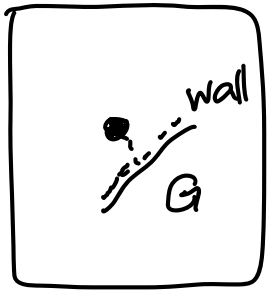
search tree

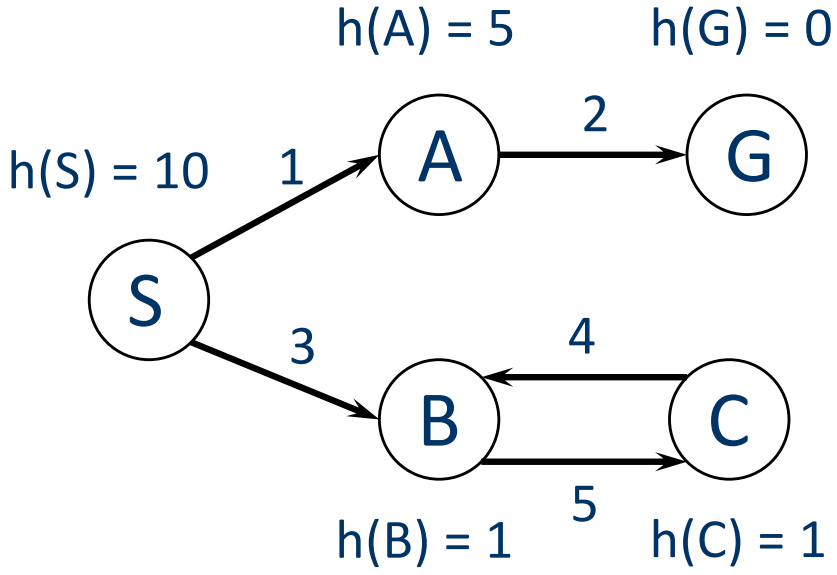


GBFS Properties - Completeness

Is GBFS guaranteed to find a solution if a solution exists?

- No.
- Could you construct a counterexample?

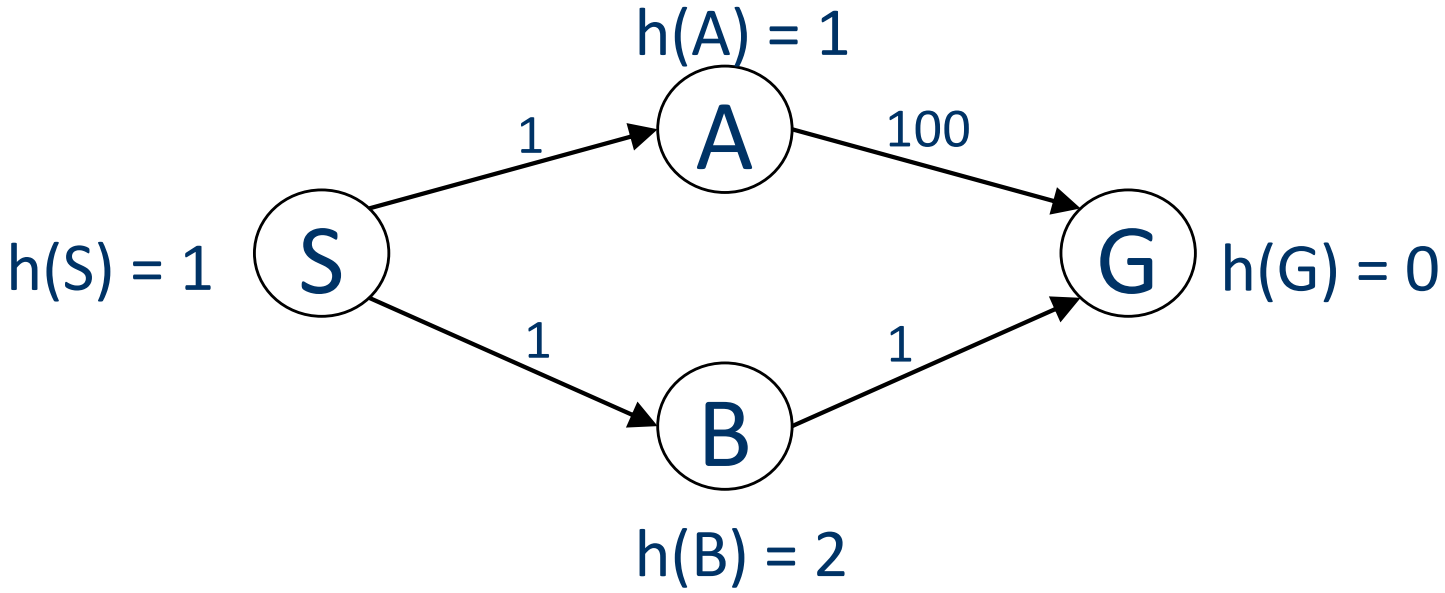




GBFS Properties - Optimality

Is GBFS guaranteed to find an optimal solution?

- No.
- Could you construct a counterexample?
  - GBFS returns SAG but the optimal solution is SBG.



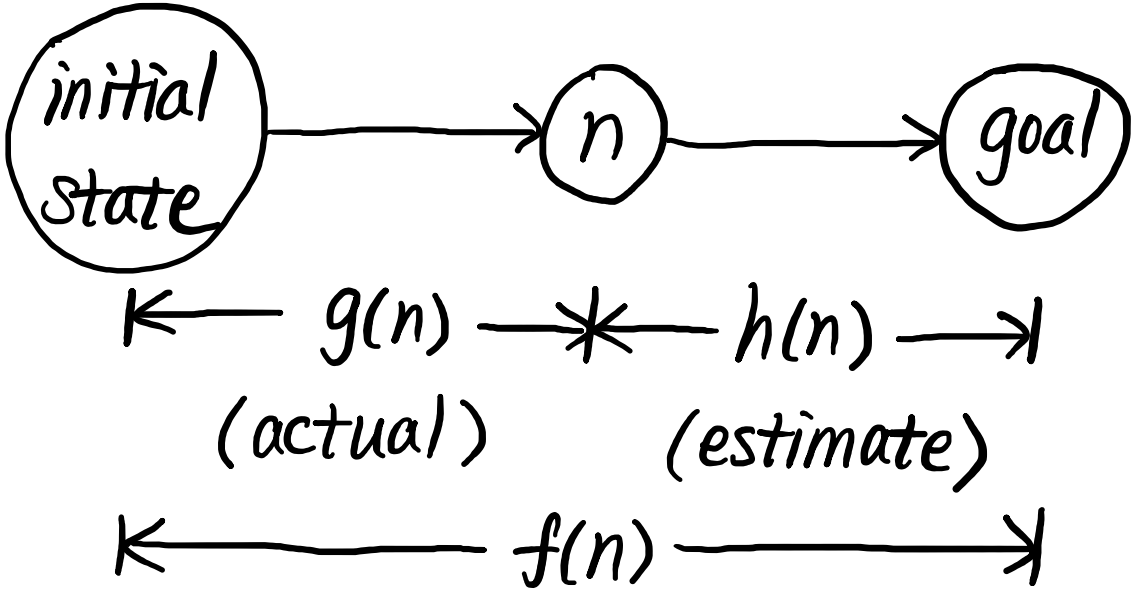
GBFS Properties – Space and Time Complexities

- Both are exponential.
- If $h(n) = 0$ for every state n, then GBFS becomes an uninformed search algorithm.

A* SEARCH

A* Search

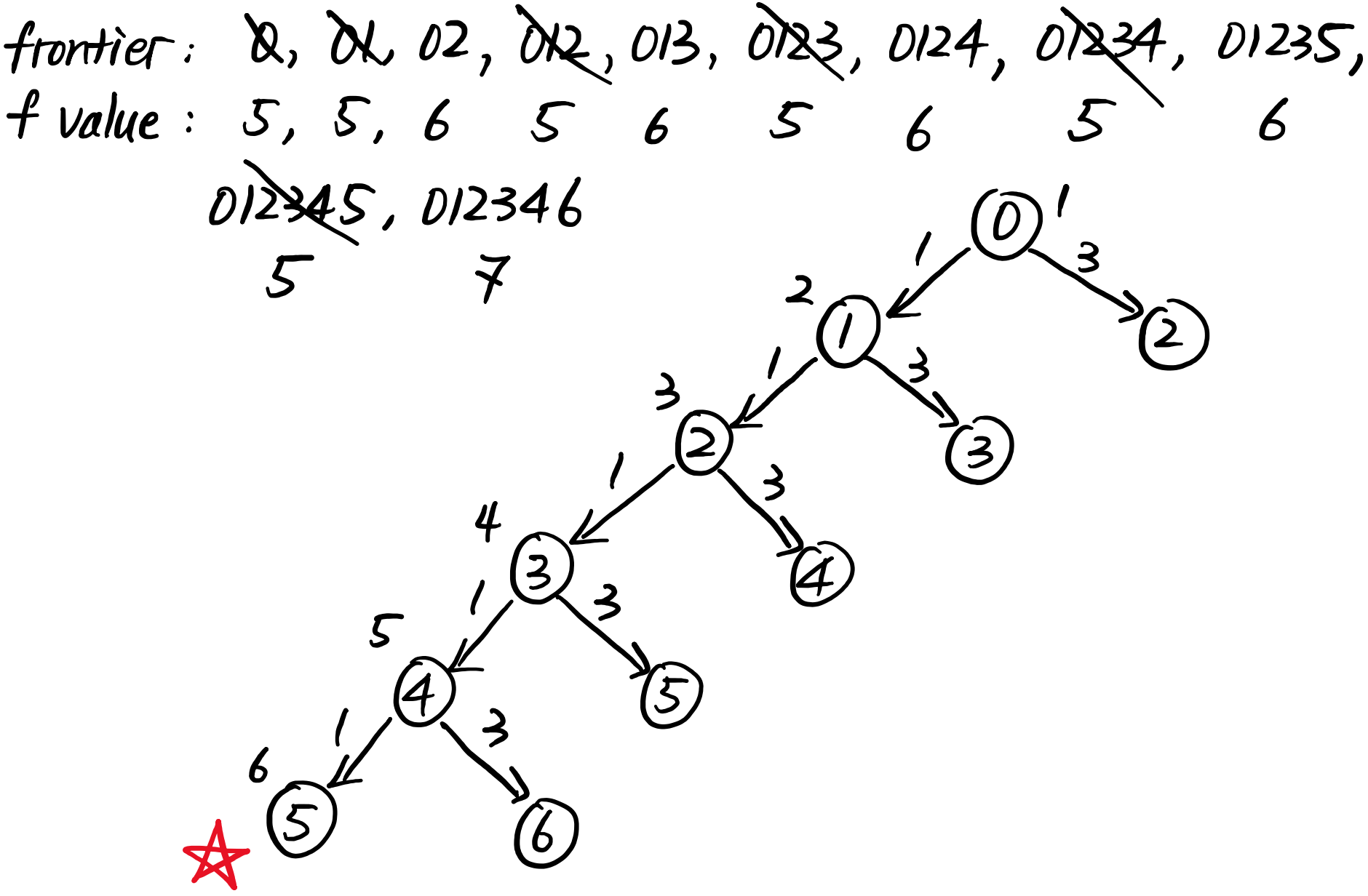
- Frontier is a priority queue ordered by
$$f(n) = g(n) + h(n).$$
- Expands the node with the smallest $f(n)$.



Integer Example

- States: the non-negative integers {0, 1, 2, …}.
- Initial state: 0.
- Goal state: 5.
- Successor function: S(n) = {n+1, n+2}.
- Cost function: C(n, n+1) = 1, C(n, n+2) = 3.
- Heuristic function:
  - If $n \leq 5$, $h(n) = 5 - n$.
  - Otherwise, $h(n) = 0$.

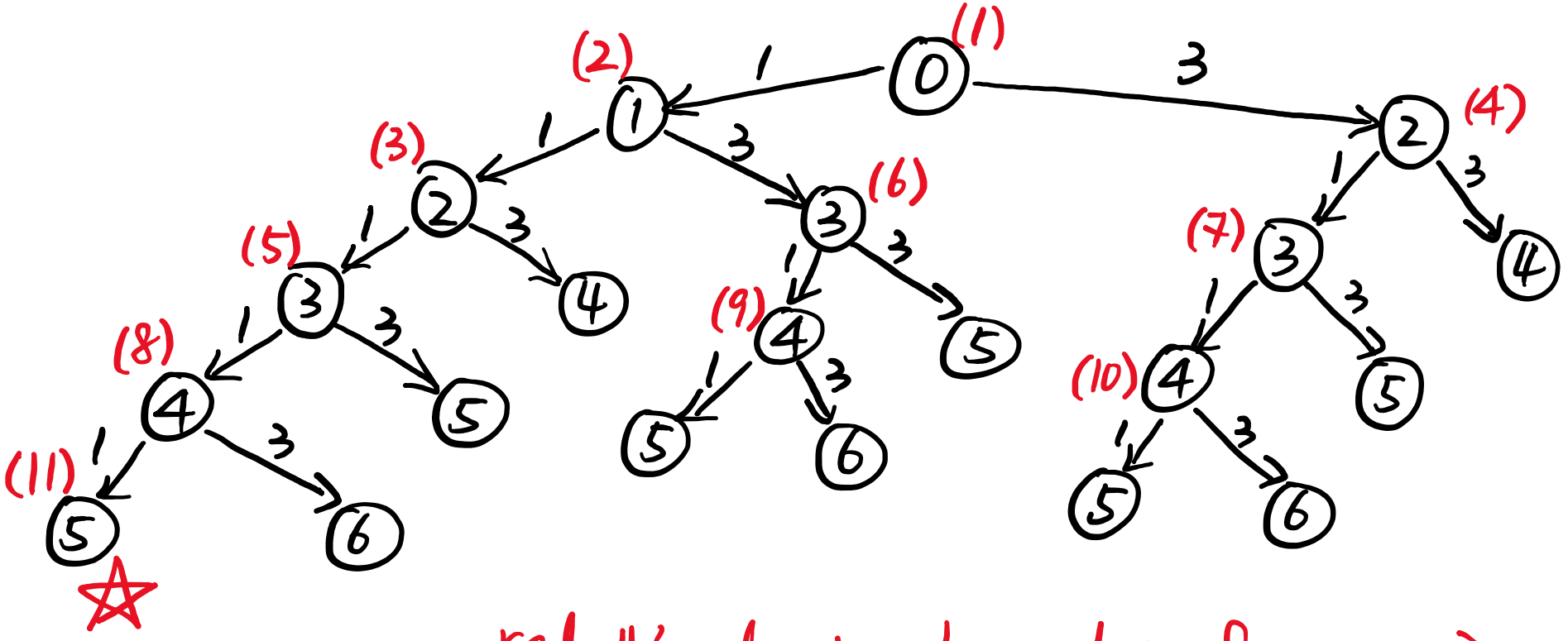
A* Search on Integer Example



UCS on Integer Example

tie-breaking rule: expand oldest node on the frontier.

frontier: 0(5), 01(1), 02(3), 012(2), 013(4), 0123(3), 0124(5)
023(4), 024(6), 01234(4), 01235(6), 0134(5), 0135(7),
0234(5), 0235(7), 012345(5), 012346(7), 01345(6),
01346(8), 02345(6), 02346(8)



red #'s denote the order of expansion.

A* Search Properties

- Complete?
  - Yes, under some mild conditions.
- Optimal?
  - Yes, if $h(n)$ satisfies a mild condition.
- Space Complexity
  - Exponential
- Time Complexity
  - Exponential

Admissible Heuristic Definition

A heuristic function $h(n)$ is admissible if and only if it never overestimates the cost of the cheapest path from state n to a goal state.

Let $h^*(n)$ denote the cost of the cheapest path from state n to a goal state. Then, an admissible $h(n)$ satisfies:

For every state n, $0 \leq h(n) \leq h^*(n)$.

Admissible Heuristic Intuition

What should an admissible $h(n)$ be if

- n is a goal state, or
- there is no path from state n to a goal state?

- A lower bound on the actual cost.
- Optimistic: it thinks the goal is closer than it really is.
- Ensures that A* doesn't miss any promising paths.
  - If both $g(n)$ and $h(n)$ are low, $f(n)$ will be low and A* will explore state n before considering more expensive paths.

A* is Optimal and Optimally Efficient.

Theorem (Optimality):

A* is optimal if and only if $h(n)$ is admissible.

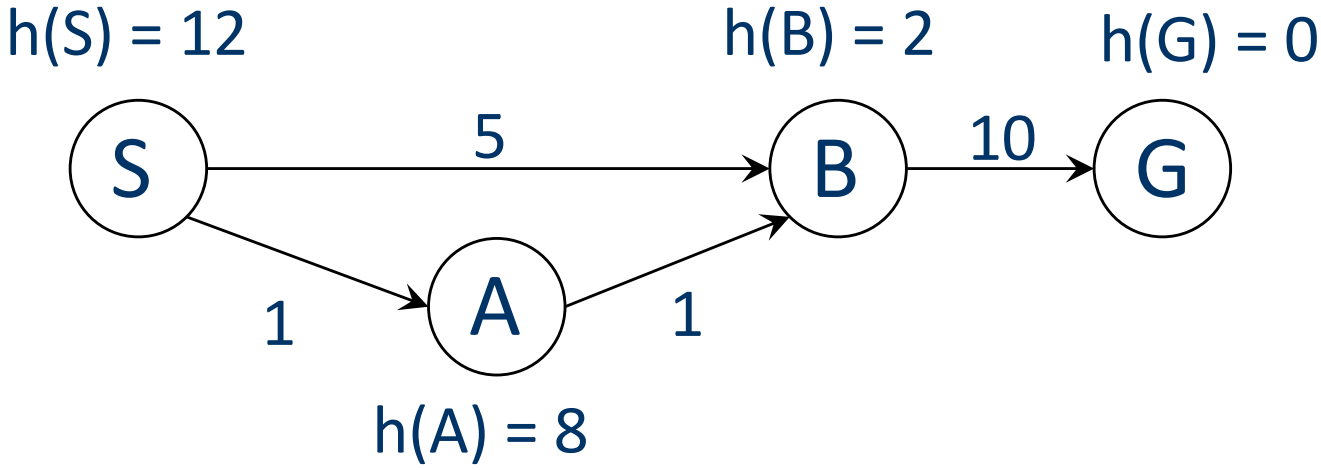
Theorem (Optimal Efficiency):

Among all optimal search algorithms that start from the **same initial state** and use the **same heuristic function**, A* expands the **fewest** states.

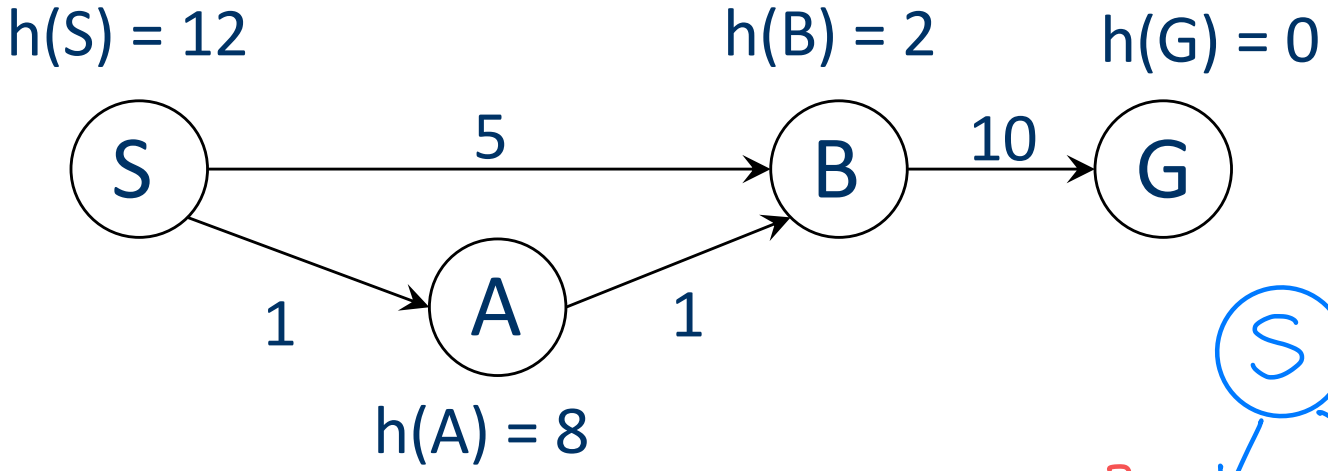
What about A* with Multi-Path Pruning?

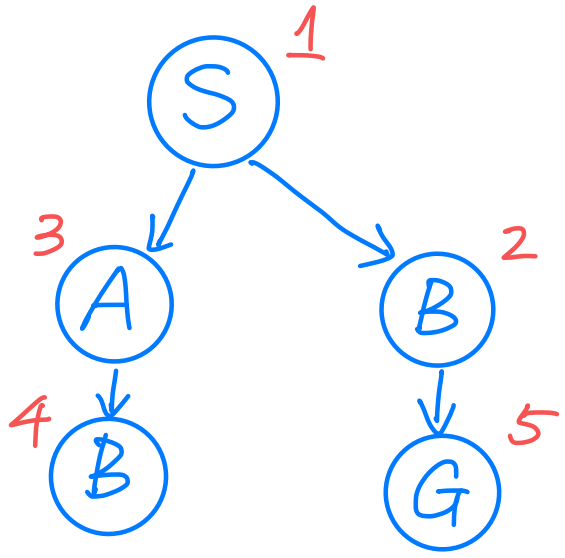
Is A* with multi-path pruning optimal?

- Let's try the example below.



A* with Multi-Path Pruning Example





frontier: ~~S~~, ~~SA~~, ~~SB~~, ~~SBG~~, ~~SAB~~ (12, 9, 7, 15, 4)

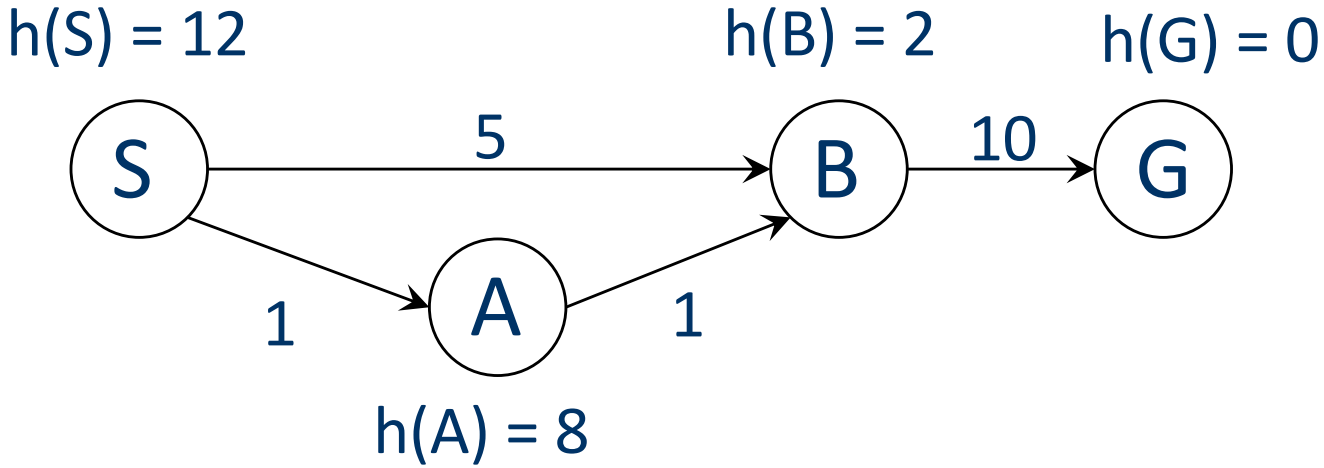
explored S, B, A, G

pruned!

solution found: SBG (cost = 15)

optimal solution: SABG (cost = 12)

A* with Multi-Path Pruning Example



frontier: ~~S(12)~~, ~~SB(7)~~, ~~SA(9)~~, ~~SBG(15)~~, ~~SAB(4)~~

explored: S, B, A

solution found: SBG (total cost = 15)

optimal solution: SABG (total cost = 12)

Consistent (Monotone) Heuristic

A heuristic function $h(n)$ is consistent if and only if

1. $h(n)$ is admissible, and
2. For every two neighboring states n_1 and n_2,
$$h(n_1) \leq g(n_1, n_2) + h(n_2).$$

Notes:

- If there are more than one edge from n_1 to n_2, the inequality must hold for all the edges.
- A consistent heuristic is admissible.

A* with Multi-Path Pruning

Theorem:

If $h(n)$ is consistent, A* w/ multi-path pruning is optimal.

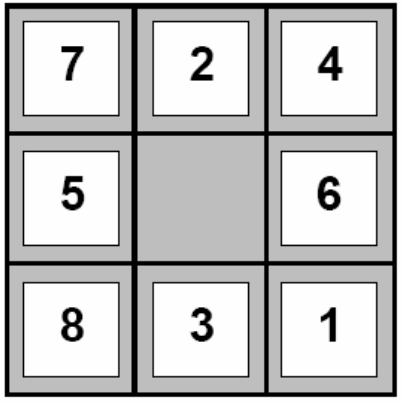
How do we ensure that $h(n)$ is consistent?

- Most admissible heuristic functions are consistent.
- Verify the definition of consistency.

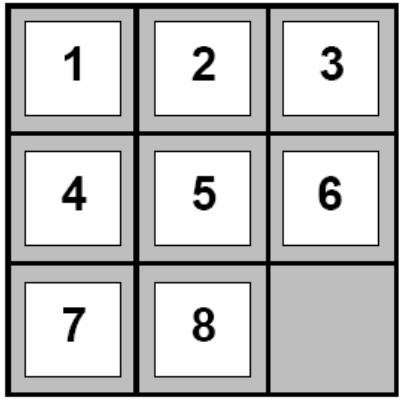
CONSTRUCTING HEURISTICS

Two Admissible H(n) for 8-Puzzle

- Manhattan distance heuristic:
  - The sum of the Manhattan distances of the tiles to their goal positions
- Misplaced tile heuristic:
  - The number of tiles that are NOT in their goal positions



Start State



Goal State

Constructing an Admissible Heuristic

1. Define a relaxed problem by simplifying or removing constraints on the original problem.

2. Solve the relaxed problem without search.

3. The cost of the optimal solution to the relaxed problem is an admissible heuristic for the original problem.

Constructing Admissible Heuristics for 8-Puzzle

8-puzzle rule: A tile can move from square A to B if

- A and B are adjacent, and
- B is empty.

Which heuristic functions can we derive from relaxed versions of this problem?

Removing One Constraint

Which heuristics can we derive from the following relaxed 8-puzzle?

A tile can move from square A to square B if A and B are adjacent ~~and B is empty~~.

A. The Manhattan distance heuristic
B. The Misplaced tile heuristic
C. Another heuristic not described above

Removing Both Constraints

Which heuristics can we derive from the following relaxed 8-puzzle?

A tile can move from square A to square B

~~if A and B are adjacent and B is empty.~~

A. The Manhattan distance heuristic
B. The Misplaced tile heuristic
C. Another heuristic not described above

Dominating Heuristics

Definition (Dominating Heuristic):

$h_1(n)$ dominates $h_2(n)$ if and only if

- $h_1(n) \geq h_2(n)$, for every state n.
- $h_1(n) > h_2(n)$, for at least one state n.

Theorem:

If $h_1(n)$ dominates $h_2(n)$, A* with $h_1(n)$ never expands more states than A* with $h_2(n)$ for any problem.

Which 8-Puzzle Heuristic is Better?

Which of the two 8-puzzle heuristics is better?

A. The Manhattan distance heuristic dominates the Misplaced tile heuristic.
B. The Misplaced tile heuristic dominates the Manhattan distance heuristic.
C. Neither heuristic dominates the other one.

CSC 384
Introduction to Artificial Intelligence

Game Tree Search

Alice Gao and Randy Hickey

Winter 2023

Outline

1. Types of Games
2. Minimax Search
3. Alpha-Beta Pruning
4. Extensions to Alpha-Beta

TYPES OF GAMES

Learning Outcomes

By the end of this section, you should be able to

- Determine whether a game has a property.
- Explain how each property has implications on the complexity of analyzing a game.

Games

Games are the oldest, most well-studied domain in AI.

- Fun
- Easy to represent. Clear rules.
- State spaces can be very large
  - Search tree for chess has $\sim 10^{154}$ nodes.
- Decisions must be made in real-time.
- Easy to determine when a program is doing well.

Properties of Games

- Single-Player v.s. Multiple-Player
- Zero-Sum v.s. Non-Zero-Sum
- Deterministic v.s. Stochastic
- Perfect Information v.s. Imperfect Information

Single v.s. Multiple Player

Single Player

- Your opponent is the game...

Multiple Player

- Other players may be
  - adversarial, or
  - cooperative

Zero-Sum or not

Zero (Constant) – Sum

- Total payoff to all players is constant.
- You win what the other players lose.

Non–Zero (Constant) Sum

Deterministic v.s. Stochastic

Deterministic

- Change in state is fully controlled by the players.
- No random elements.

Stochastic

- Change in state is partially determined by chance.

Perfect v.s. Imperfect Information

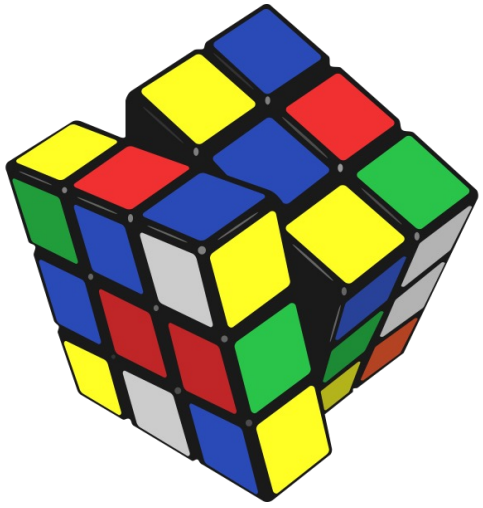
Perfect information

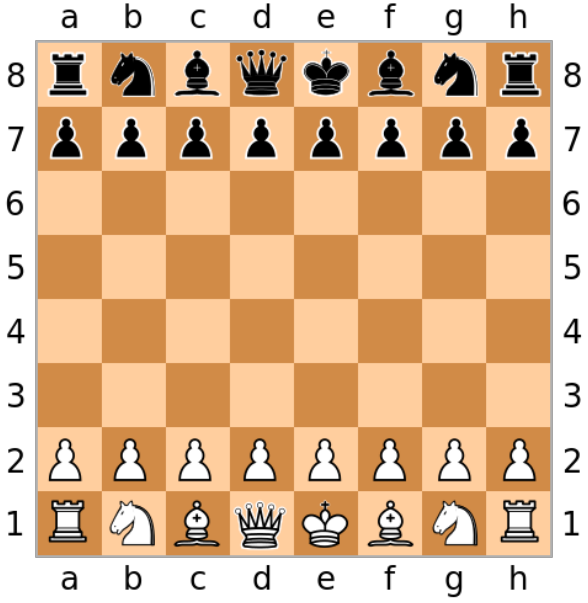
- State is fully observable.

Imperfect information

- Part of state is hidden.

Let's categorize some games





We will study one type of game

- Two-player
  - Must model the other player's goals and strategies.
- Zero-sum
  - One player's gain = the other player's loss
  - It suffices to model one player only.
- Deterministic
  - No need to model random elements. No probabilities.
- Perfect-Information
  - No need to model hidden information.

MINIMAX SEARCH

Learning Outcomes

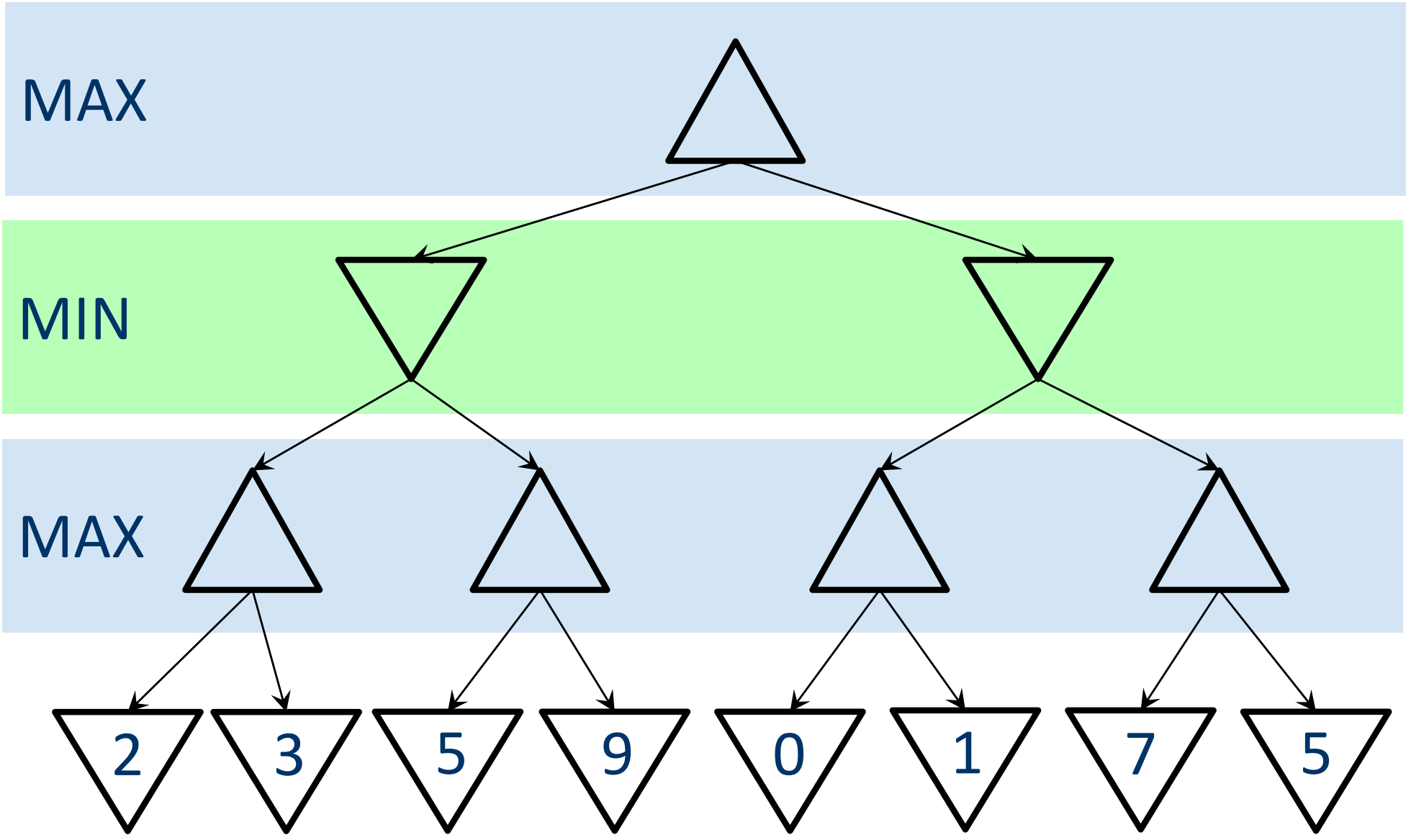
By the end of this section, you should be able to

- Explain the difference between a solution for a standard search and a strategy for an adversarial game.
- Explain the conditions under which the minimax strategy is optimal.
- Execute Minimax Search on a game tree and determine the minimax value of the root node.
- Compare and contrast Depth-First Search and Depth-First Minimax Search.

Two-Player Zero-Sum Game

- Two players: MAX and MIN
- Initial state s_0
- $player(s)$:
  - returns the player who moves in state s.
- $actions(s)$:
  - returns the legal moves in state s.
- $result(s, a)$:
  - returns the next state after taking action a in state s.
- $terminal(s)$:
  - returns True iff s is a terminal state.
- $utility(s)$:
  - returns MAX's payoff in terminal state s.

Game Tree Example



Player's Strategy

- In standard search, a solution is a sequence of moves leading to a goal state.
- In a game, the strategy (for MAX) specifies
  - a move for the initial state.
  - a move for all possible states arising from MIN's response.
  - all possible responses to all of MIN's responses to MAX's previous moves.
- In short, a strategy specifies what move to make at every contingency.

Minimax Strategy

An optimal strategy leads to outcomes

at least as good as any other strategy.

The minimax strategy is optimal

assuming that the opponent is playing optimally.

If the opponent isn't playing optimally, we can have a better strategy that exploits the opponent's weaknesses.

Minimax Value

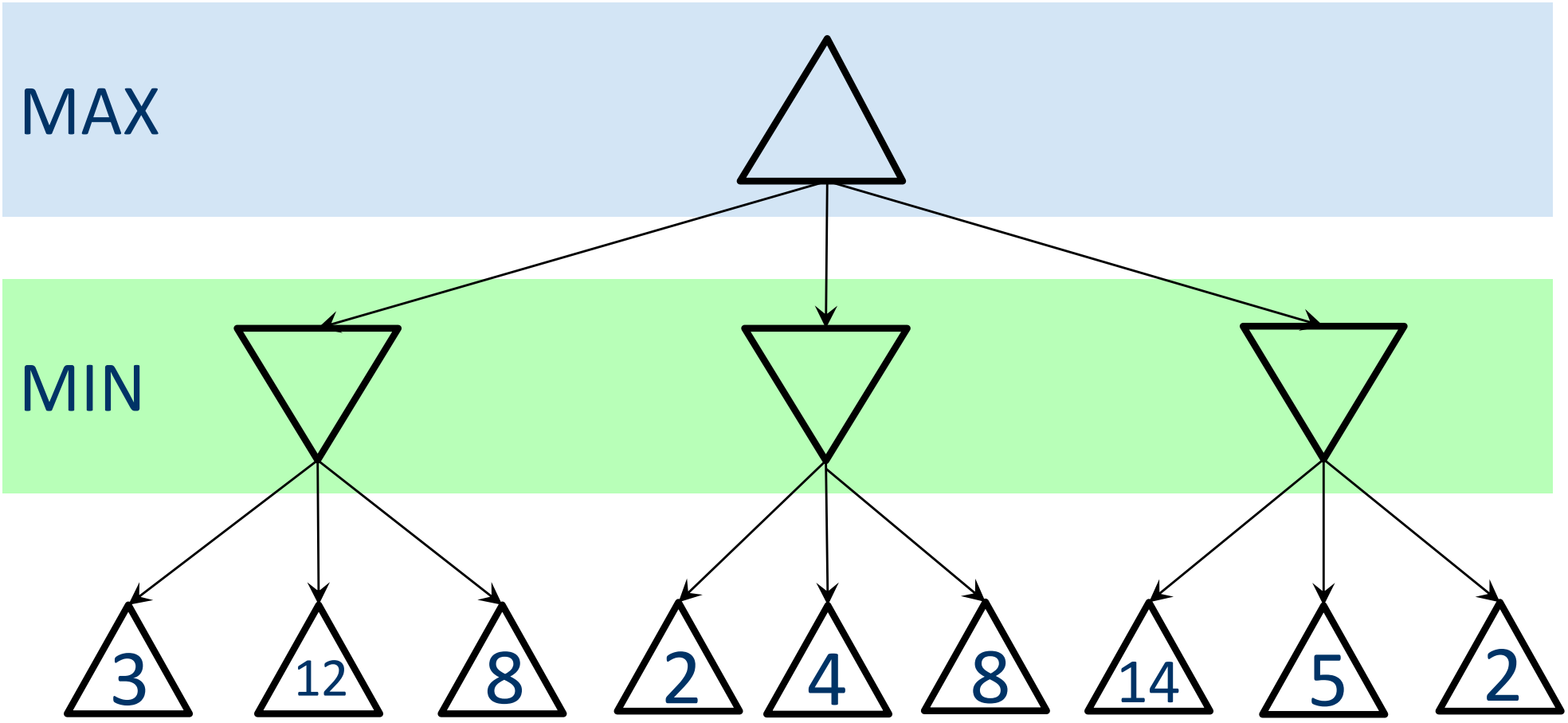
minimax-value(s) =

- utility(s), if s is a terminal state,
- max_{s' in succ(s)} minimax-value(s') if s is a MAX node.
- min_{s' in succ(s)} minimax-value(s') if s is a MIN node.

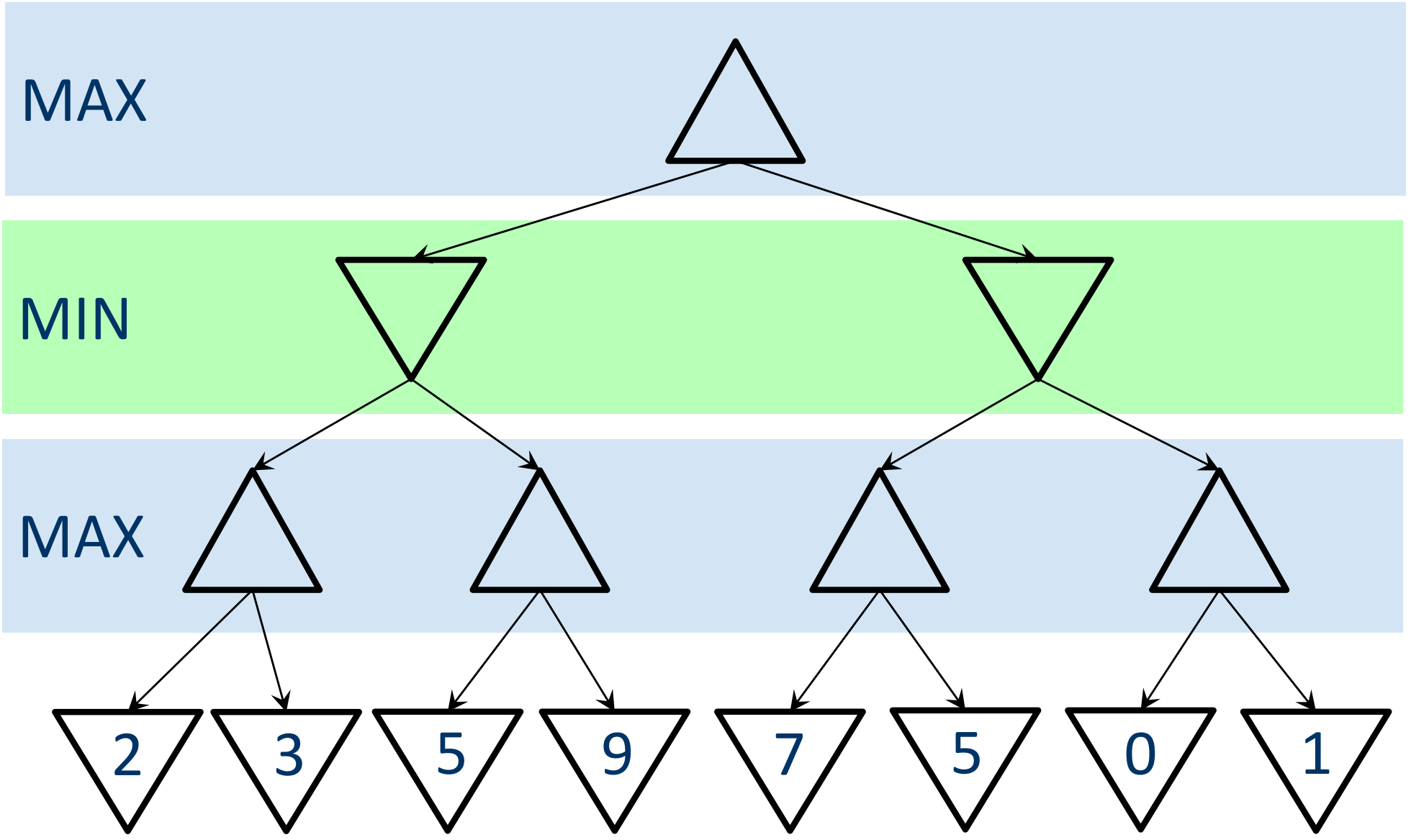
How can we determine the minimax value of the root?

- Start from the terminal states.
- Propagate the values up the tree.
  - As a MAX node, take max of its children's values.
  - At a MIN node, take min of its children's values.

Example 1: Compute Minimax Values



Example 2: Compute Minimax Values



DEPTH-FIRST MINIMAX

Why Depth-First Minimax?

- So far, computing the minimax value requires
  - Building the entire game tree
  - Backing up values
- Save space by using a depth-first version of minimax.
  - Avoids representing the exponentially sized game tree
  - Allows us to prune some states using alpha-beta algorithm.

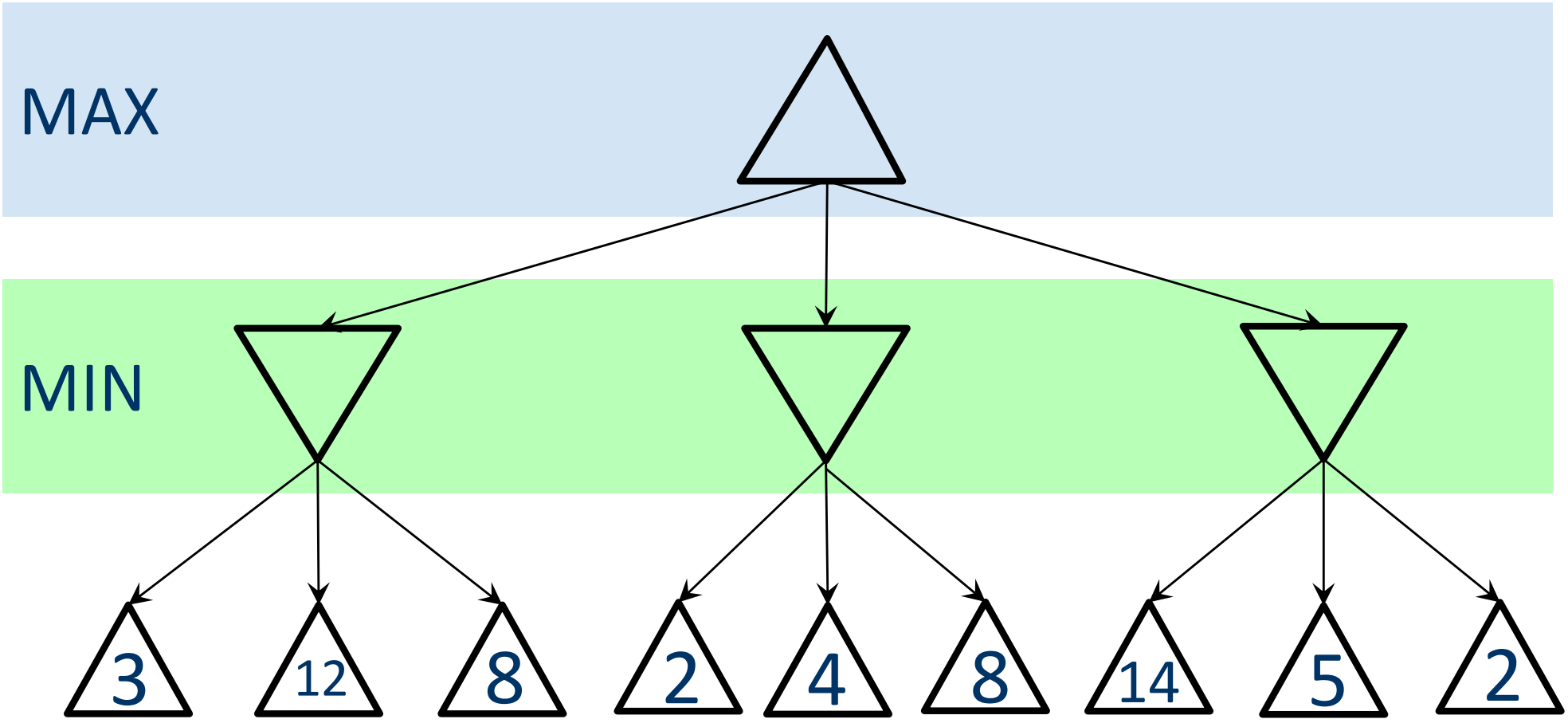
Depth-First Minimax Pseudocode

```
function MINIMAX-DECISION(state) returns an action
    return argmax_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL(state) then return UTILITY(state)
    v ← −∞
    for each action in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, action)))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL(state) then return UTILITY(state)
    v ← ∞
    for each action in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, action)))
    return v
```

Q1: Depth-First Minimax



Time and Space Complexity

- m is the max depth of the tree.
- b is the branching factor.

- Space complexity

$$O(bm)$$

- Time complexity is

$$O(b^m)$$

- Must traverse entire search tree to evaluate all moves.

Compare and Contrast

Depth-First Search

Depth-First Minimax

ALPHA-BETA PRUNING

Learning Outcomes

By the end of this section, you should be able to

- Explain the roles of alpha and beta in the alpha-beta pruning algorithm.
- Explain the reasoning behind pruning at a MAX/MIN node.
- Execute Alpha-Beta Pruning on a game tree, indicate all the pruned branches, and determine the minimax value of the root node.
- Explain the correctness and space complexity of alpha-beta pruning.

Problem with Minimax

- Must visit the entire search tree.
- # nodes visited is exponential in the depth of the tree.
- Cannot eliminate the exponent, but can cut it in half.
- Can compute the minimax value without looking at every node in the tree.
  - Prune branches that do not influence our computation.

Values in Alpha-Beta Pruning

- v = the current value for a node.

- α = value of best (highest-value) choice so far for MAX.
  - MAX guarantees that the minimax value $\geq \alpha$
  - α is a lower bound.

- β = value of best (lowest-value) choice so far for MIN.
  - MIN guarantees that the minimax value $\leq \beta$.
  - β is an upper bound.

- Whenever $\alpha \geq \beta$, pruning happens!
  - Whoever decides first determines the outcome.

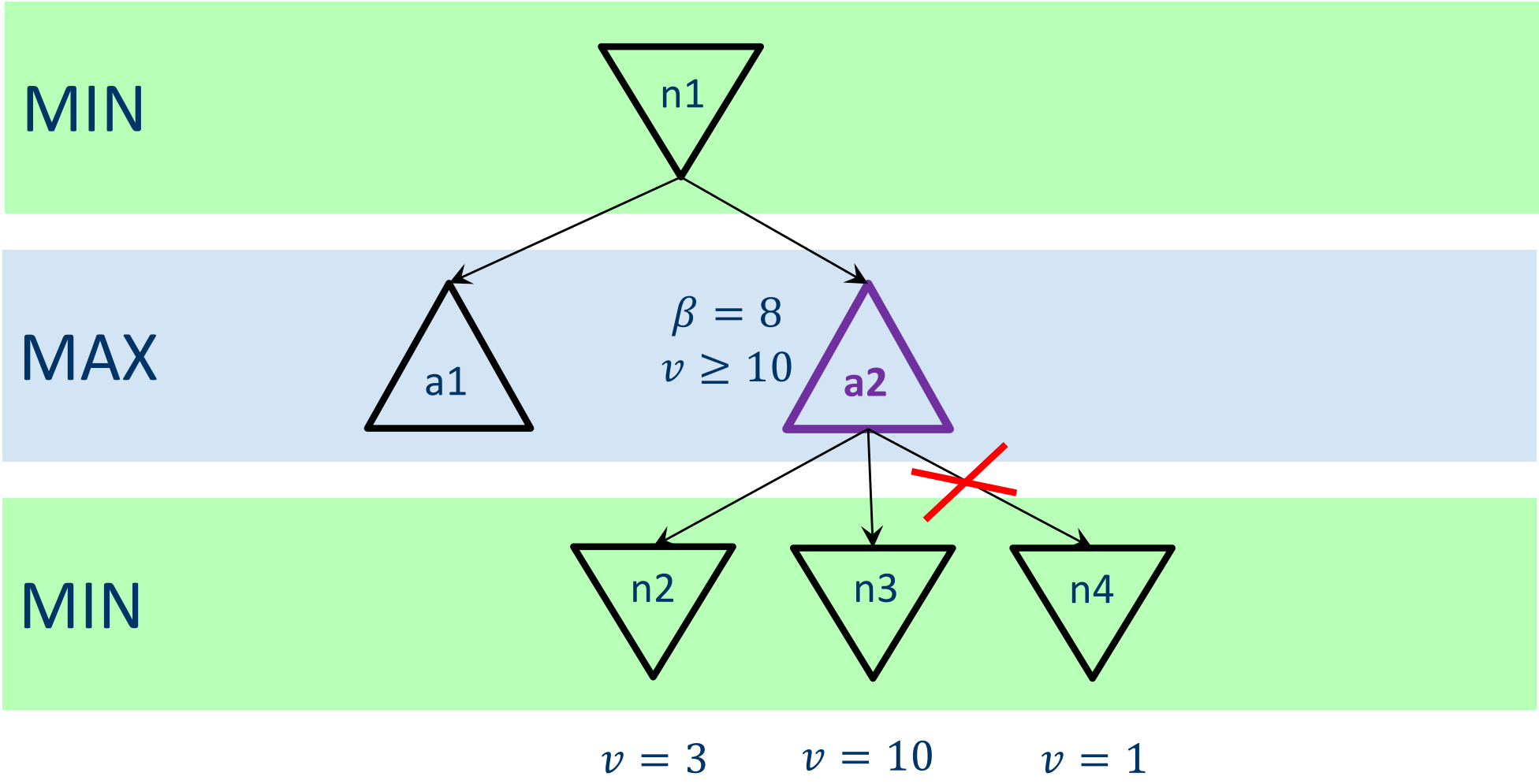
Pruning at a MAX node

- At a MAX node,
- If $v \geq \beta$, prune remaining children of current node.

Reasoning:

- MIN can guarantee a value of at most β.
- Current MAX node has value greater than β.
- Thus, the current node would NOT be reached.
  - MIN would not allow it!

Pruning at a MAX Node



At node a2, MIN can guarantee a value of ≤ 8, but MAX has a value of ≥ 10.
Thus, **a2 would NEVER be reached (MIN would NOT allow it).**

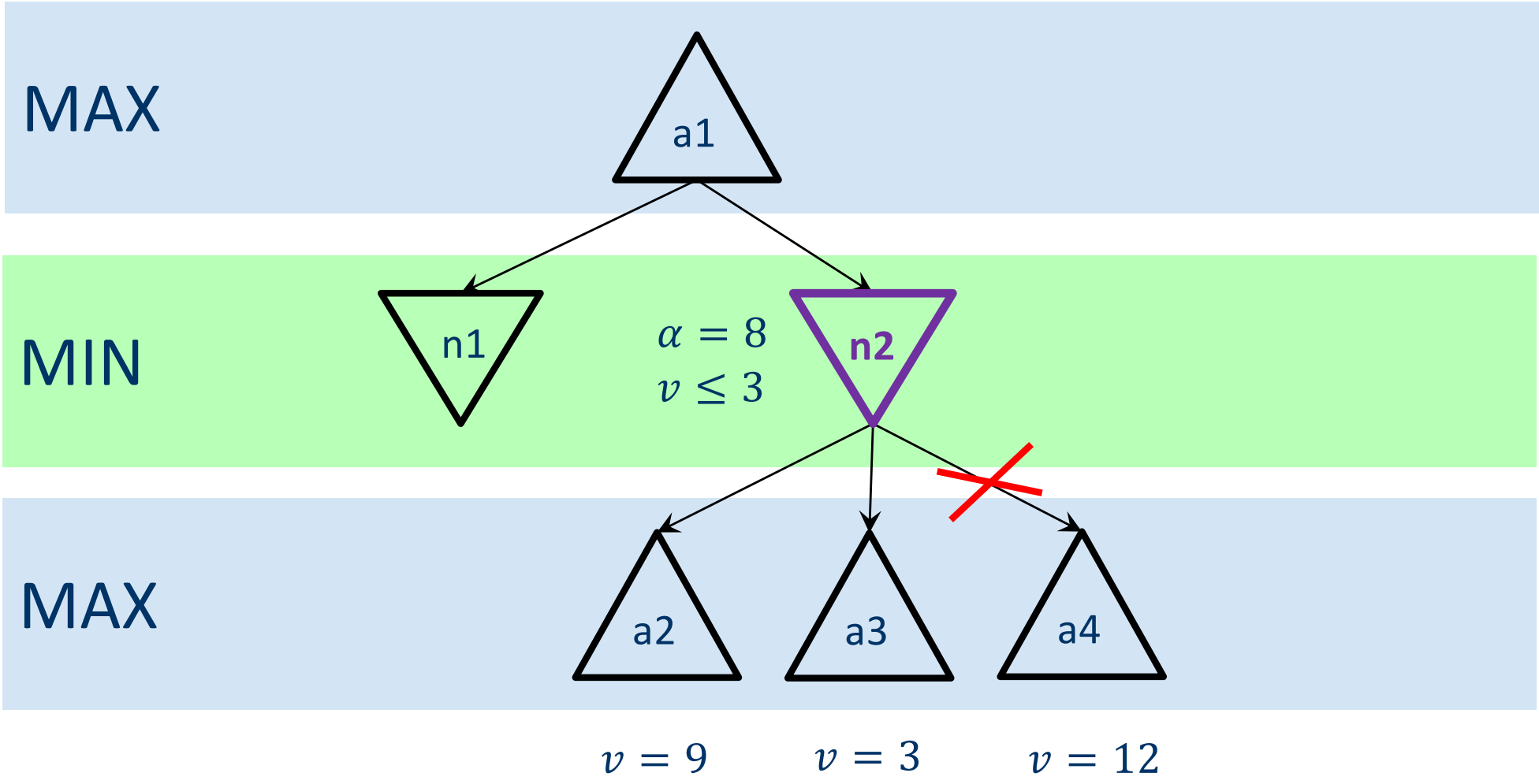
Pruning at a MIN Node

- At a MIN node,
- If $v \leq \alpha$, prune remaining children of current node.

Reasoning:

- MAX can guarantee a value of at least α.
- Current MIN node has value less than α.
- Thus, the current node would NOT be reached.
  - MAX would not allow it!

Pruning at a MIN Node



At node n2, MAX can guarantee a value of ≥ 8, but MIN has a value of ≤ 3.
Thus, **n2 would NEVER be reached (MAX would NOT allow it).**

Alpha-Beta Pruning Pseudocode

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state,−∞,+∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL(state) then return UTILITY(state)
    v ← −∞
    for each action in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, action), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL(state) then return UTILITY(state)
    v ← +∞
    for each action in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, action), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Alpha-Beta Pruning Pseudocode

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state,−∞,+∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each action in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, action), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each action in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, action), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

Did you notice? We never return α and β to the parent node.

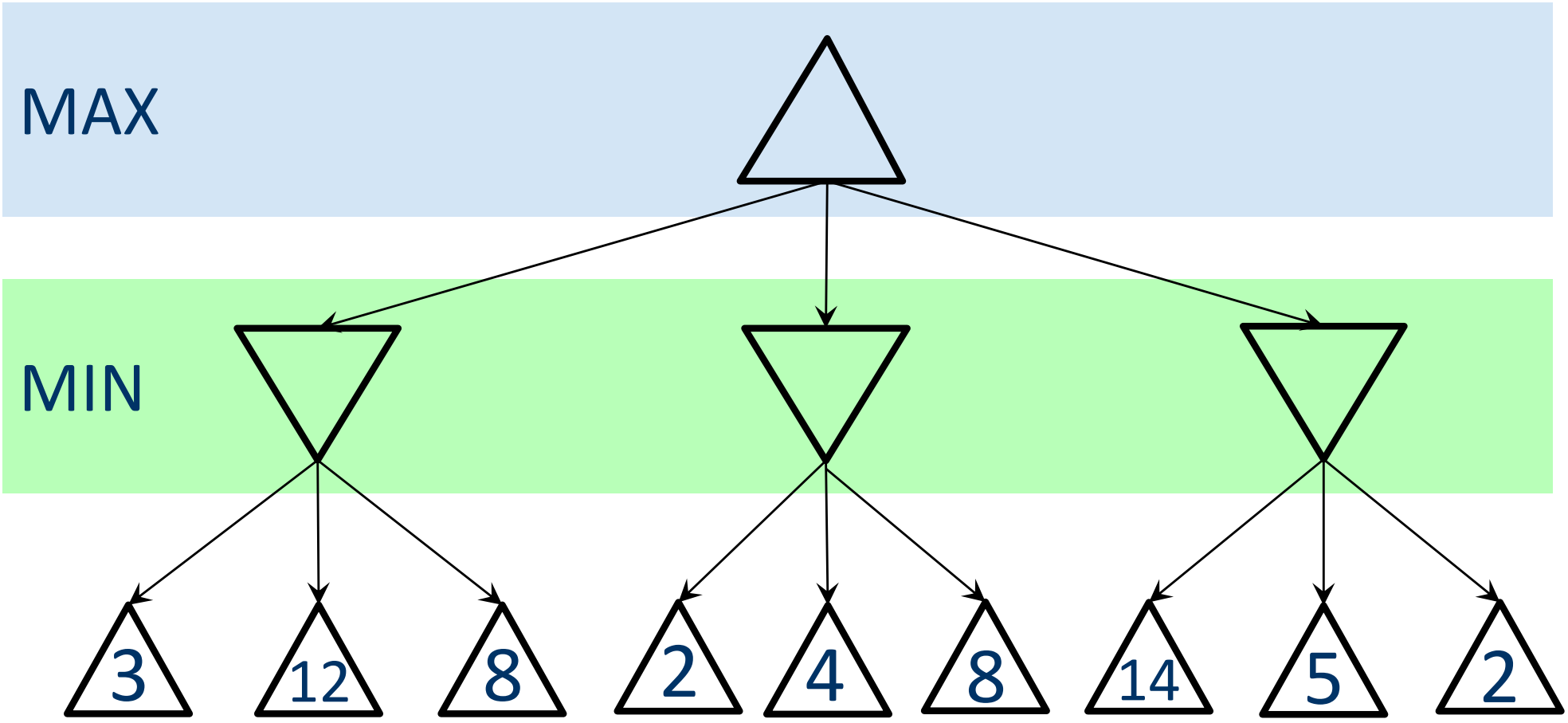
Pruning at a MAX node

MAX increases α.

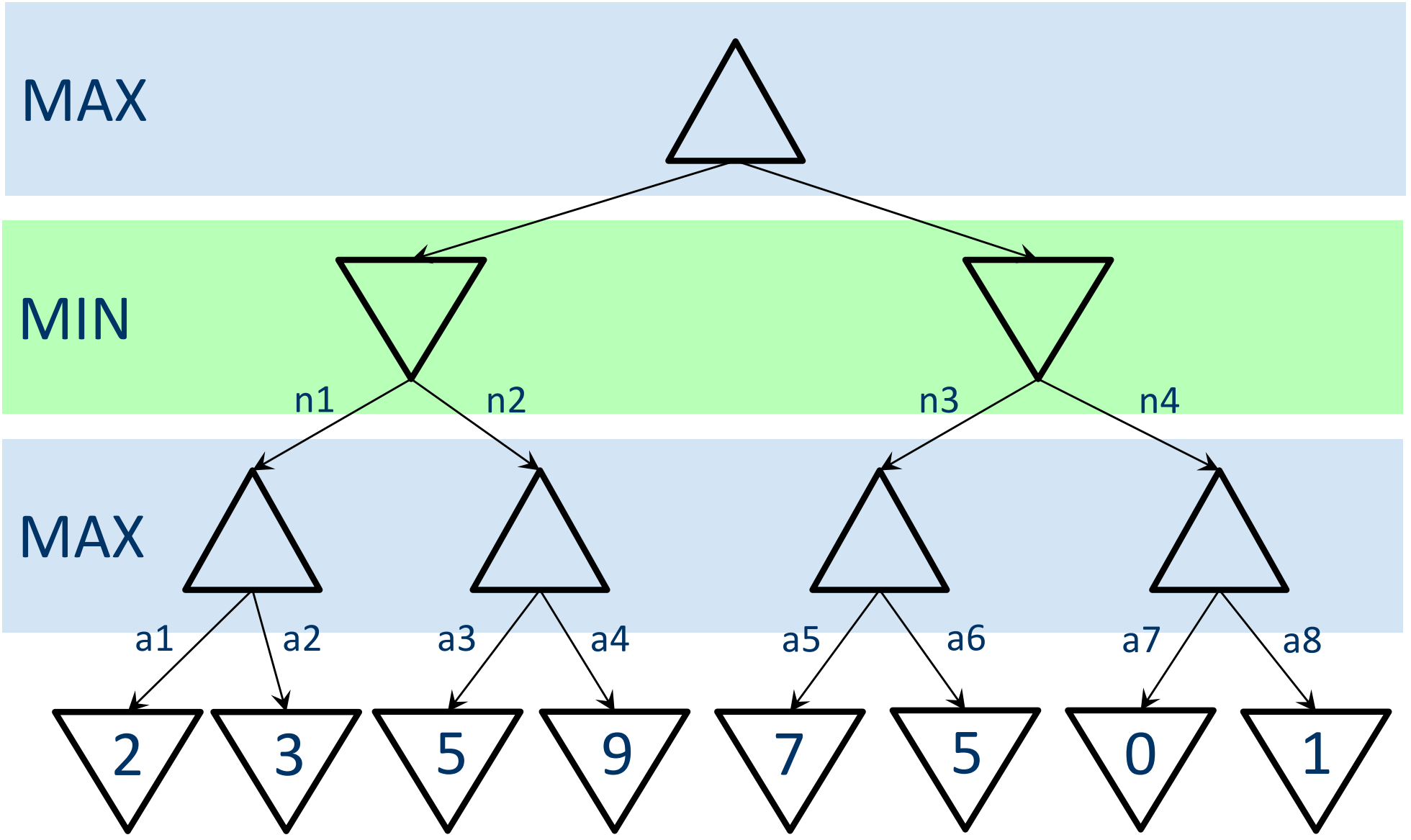
Pruning at a MIN node

MIN decreases β.

Example 1: Alpha-Beta Pruning



Example 2: Alpha-Beta Pruning



Alpha-Beta Pruning Properties

- Can pruning result in a different outcome than minimax search?
  - No. Pruning only eliminates states that we did not have to visit in the first place.

- How much can be pruned when searching?
  - Can reduce the branch factor from b to $\sqrt{b}$ w/ perfect pruning.
  - In theory, we can search **twice as deep**.

EXTENSIONS TO ALPHA-BETA PRUNING

Learning Outcomes

By the end of this section, you should be able to

- Explain how the move ordering affects the complexity of alpha-beta pruning.
- Given a game tree, change the node ordering to maximize/minimize pruning.
- Describe strategies to enhance the performance of alpha-beta pruning (move orderings, handling repeated states, and using a cut-off test and an evaluation function).
- Describe strategies to design an evaluation function with desirable properties.

Real-Time Decisions

- Alpha-beta dramatically improves over minimax.
- But it is still not good enough sometimes.
  - Need to search to terminal states for part of search space.
  - Need to make decisions quickly.
- Solutions:
  - Evaluation function + cutoff tests.
  - Move ordering.
  - Caching states.

Cut-off test + Evaluation function

Problem:

- α-β pruning must search to terminal states for part of the search space.
- Searching to terminal states is impractical!

Solution:

- Cutting off the search earlier, and
  - Terminal test -> cutoff test
  - Decides when to apply the evaluation function.
- Applying an evaluation function at cut-off.
  - Utility function -> evaluation function
  - Estimates the expected utility of the state.

Alpha-Beta Pruning with Evaluation Function

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state,−∞, +∞, 0)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β, depth) returns a utility value
    if CUTOFF-TEST(state, depth) then return EVAL(state)
    v ← −∞
    for each action in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, action), α, β, depth + 1))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β, depth) returns a utility value
    if CUTOFF-TEST(state, depth) then return EVAL(state)
    v ← +∞
    for each action in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, action), α, β, depth + 1))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

The Evaluation Function

- Returns an estimated utility of a state.
  - Just as the heuristic function estimates the distance to the goal.
- Program performance depends strongly on the quality of the evaluation function.

- For example, the evaluation function can
  - Return the actual utility for any terminal state, and
  - Return an estimated utility for any non-terminal state.

Desirable Properties of Evaluation Functions

- Ensures correct behaviour for terminal states.
- Estimated utilities of non-terminal states should strongly correlate with the actual chances of winning.
- Fast to compute!

Designing an Evaluation Function

- Use expert knowledge.
- Learn from experience.
- A weighted combination of features.
  - linear combination: $EVAL(s) = w_1 f_1(s) + \cdots + w_n f_n(s)$
  - non-linear combination.

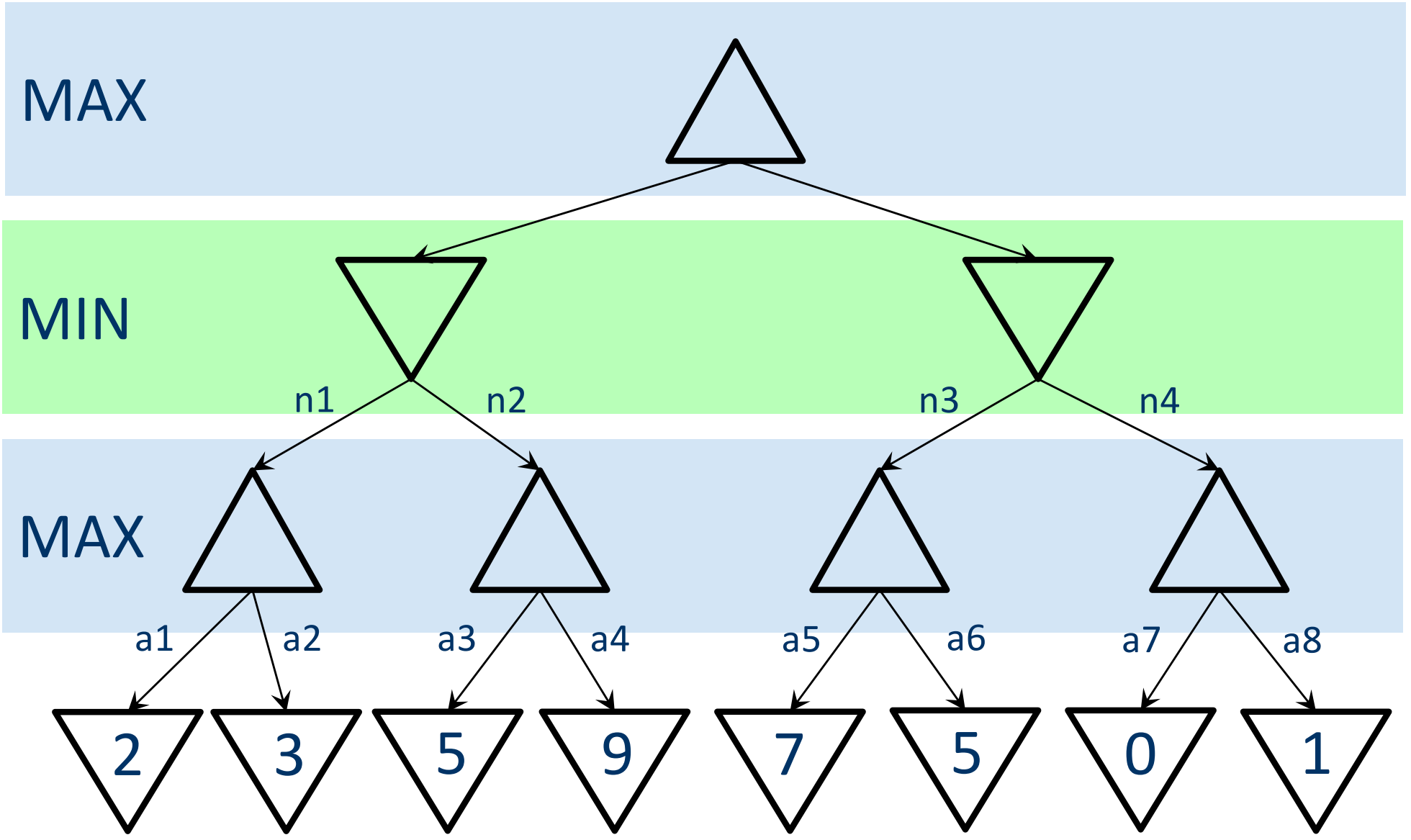
Features for estimating the state's utility

- Tic-tac-toe:
  - # of length 3 runs that are left open for each player
- Chess:
  - Each piece has a material value.
    - pawn (1), knight/bishop (3), rook (5), queen (9).
  - "Good pawn structure" or "King safety" may be worth ½ a pawn.
  - DeepBlue's evaluation function used thousands of hand-crafted features.
- Go:
  - AlphaGo's evaluation function used neural networks.

Move Ordering

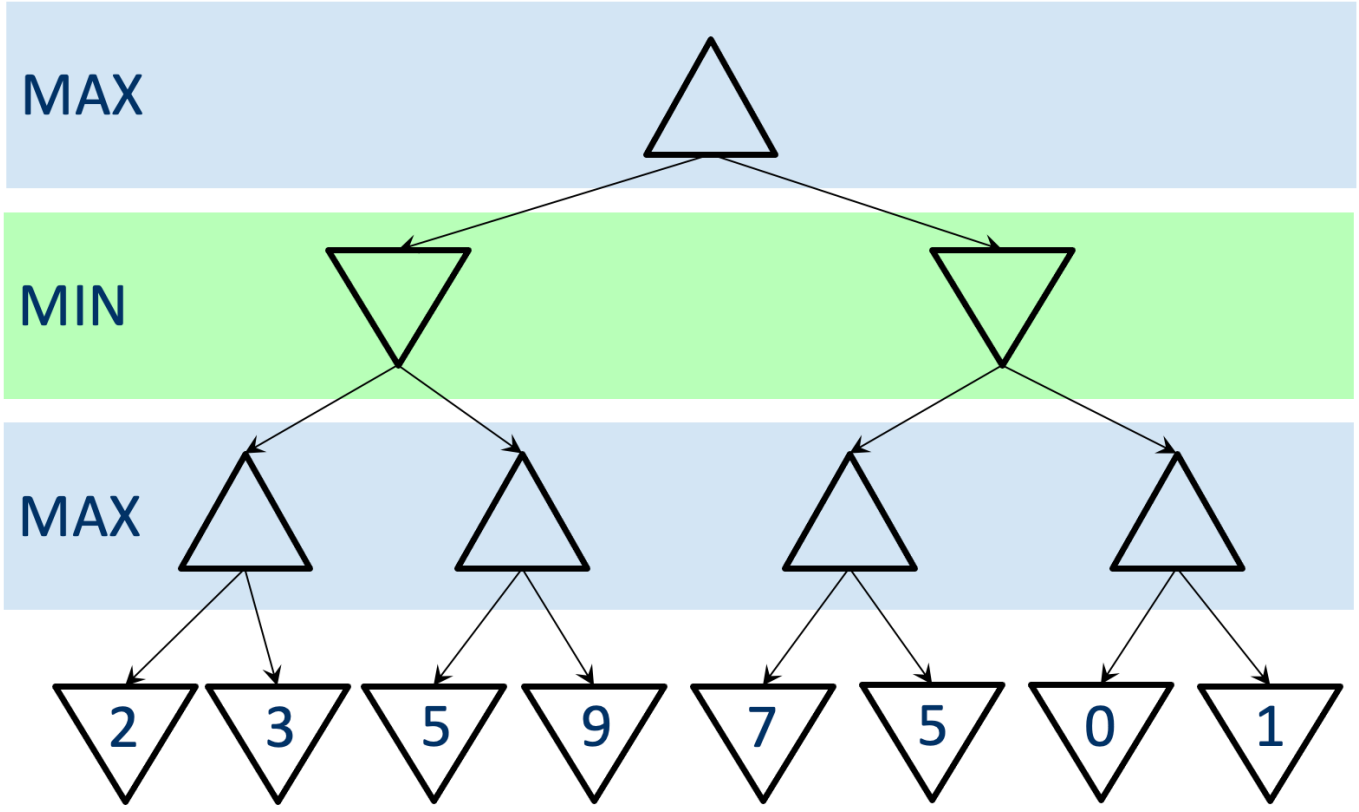
- The effectiveness of alpha-beta pruning is highly dependent on the order in which the nodes are visited.
- Ideally, we want to **visit the best child first**.
- With perfect ordering, time complexity of alpha-beta pruning becomes $O(b^{m/2})$ or $O(\sqrt{b}^{m})$.
  - The branching factor becomes $\sqrt{b}$ instead of b.

Example 2

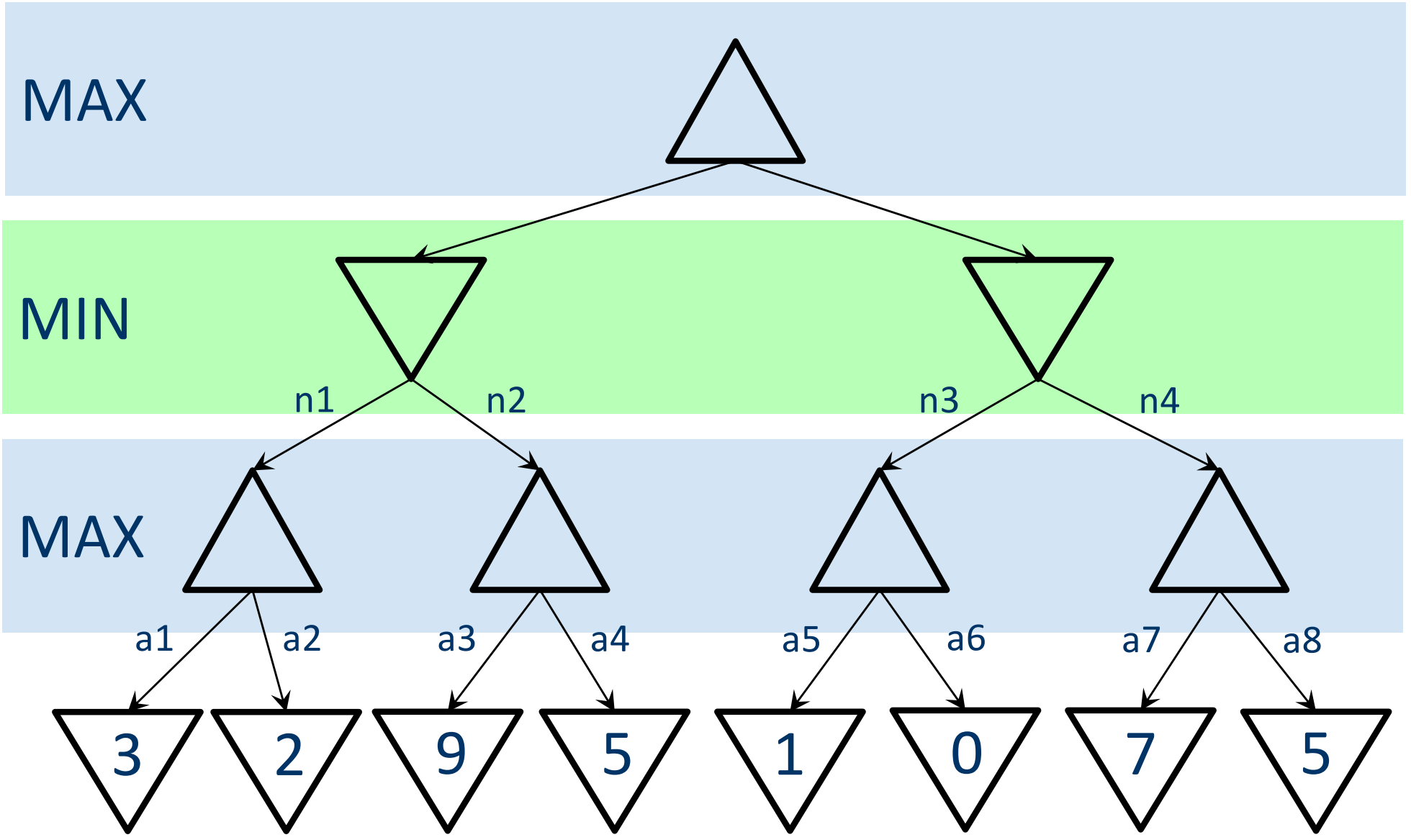


Example 2: Let's Change the Move Ordering

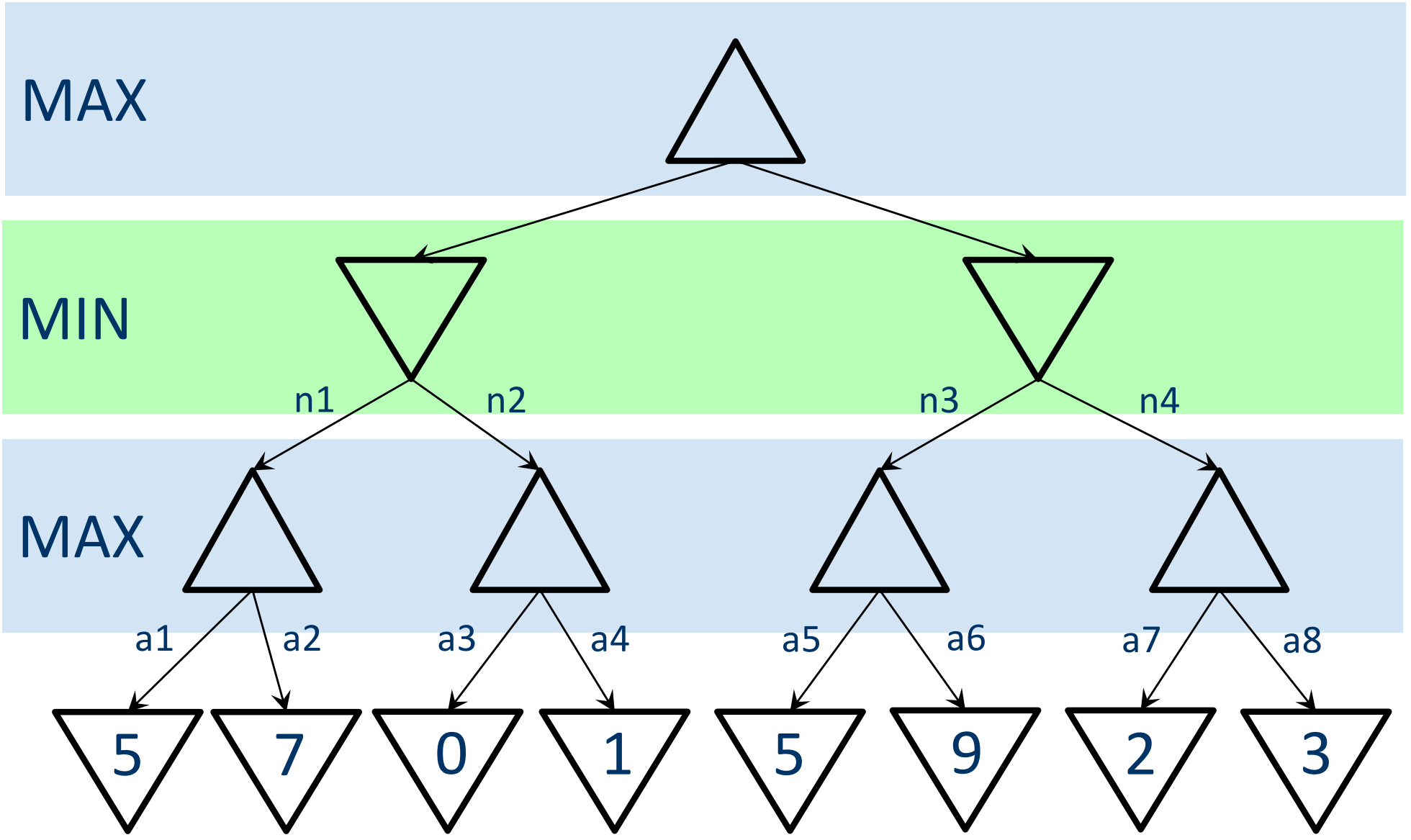
- Could you change move ordering to maximize pruning?
- Could you change move ordering to minimize pruning?



Revised Example 2a



Revised Example 2b



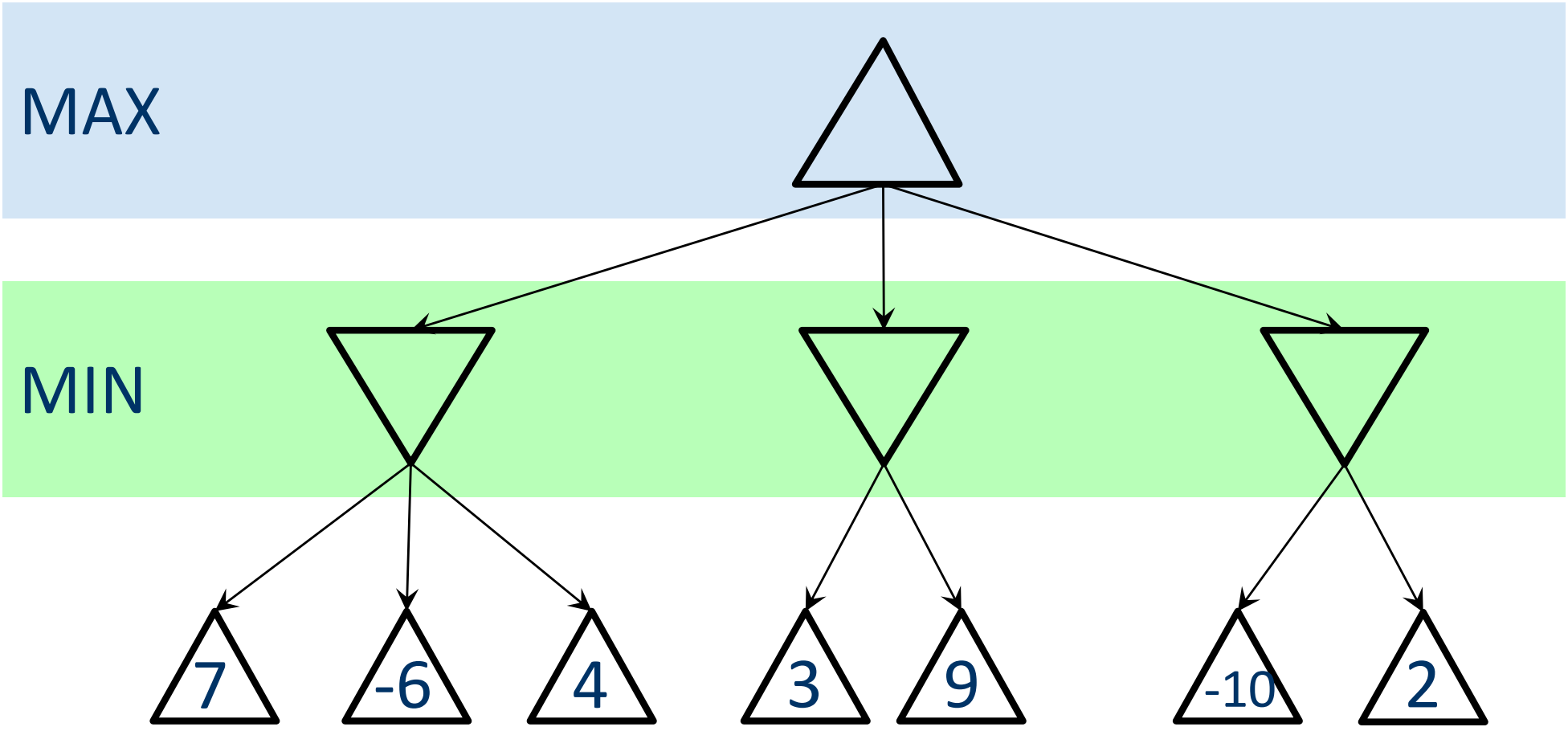
Dynamic Move Ordering

- A heuristic for chess:
  - captures first, then threats, and forward and backward moves
  - Gets you within a factor of 2 of $O(b^{m/2})$.
- Order successors using your evaluation function.
- Gain information from the current move by search.
  - Iterative deepening search
  - Search k moves deep and record the best path of moves.
  - Determine move ordering using the recorded paths.

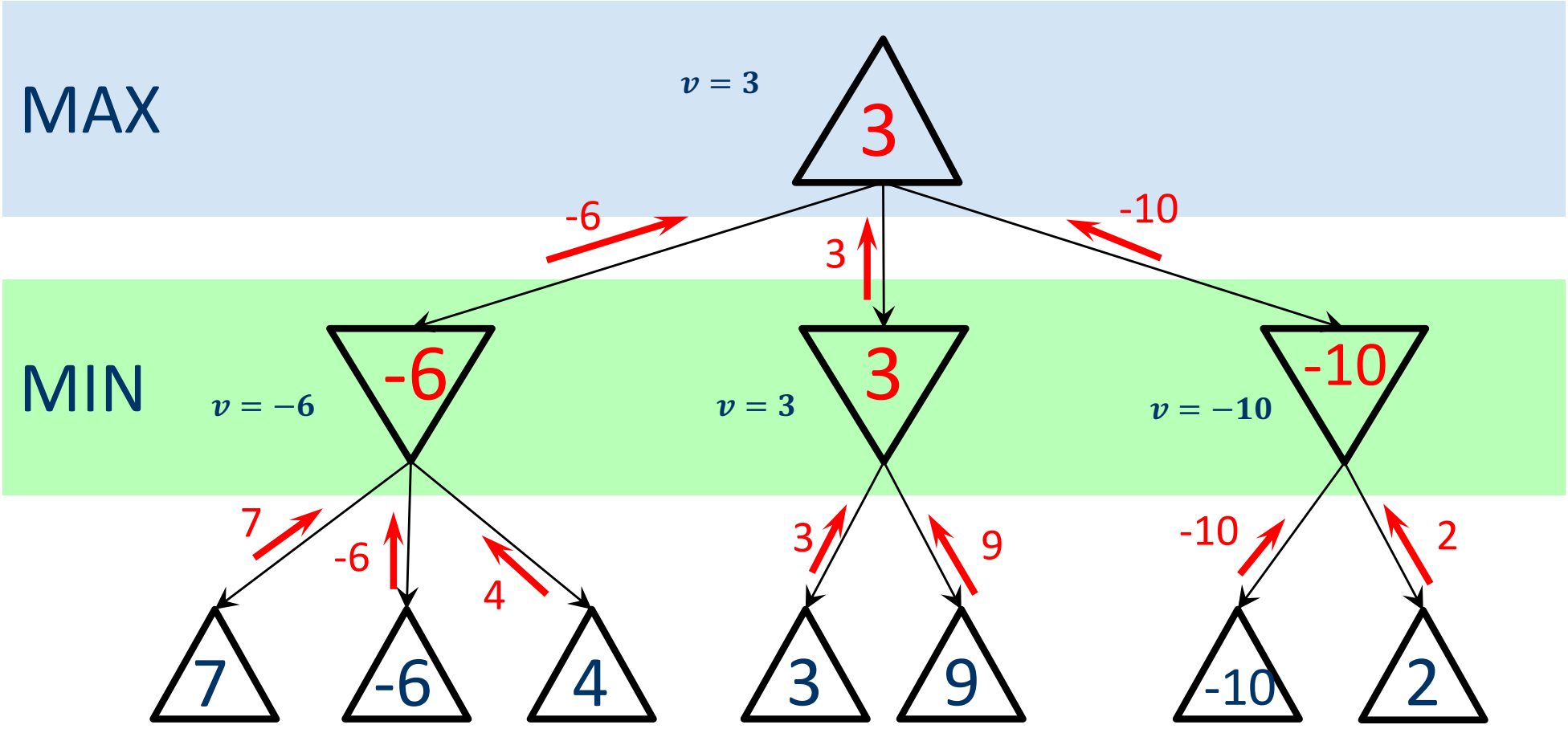
Caching States

- Permutations of moves may lead to the same position.
- Remembering repeated states can dramatically increase the maximum search depth.
  - E.g., double the search depth in chess.
- Store useful information of a state in a dictionary.
  - Like using the explored set to perform pruning.
  - Useful information: minimax value, alpha and beta values.
- May need to prune the table with limited memory.

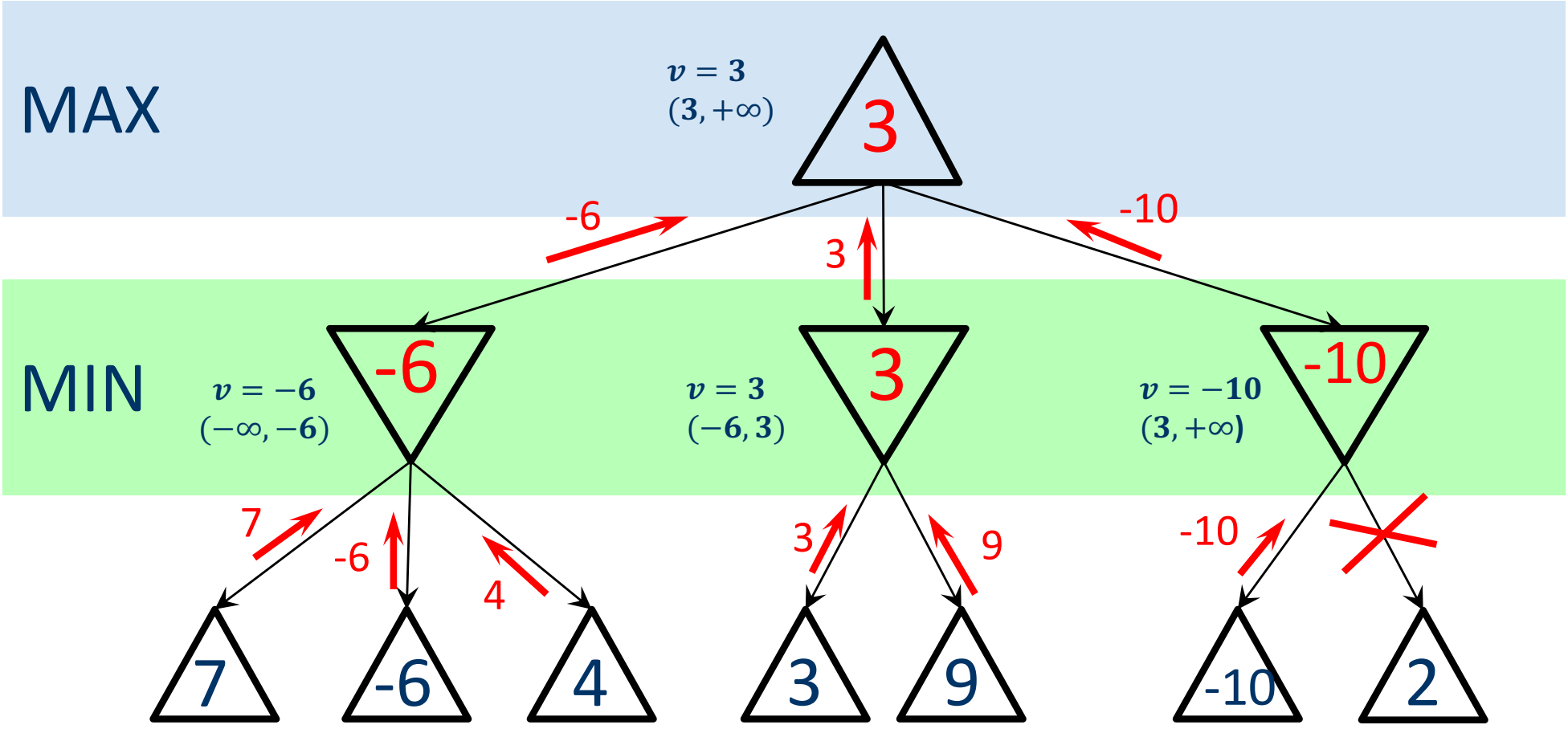
Extra Example 1



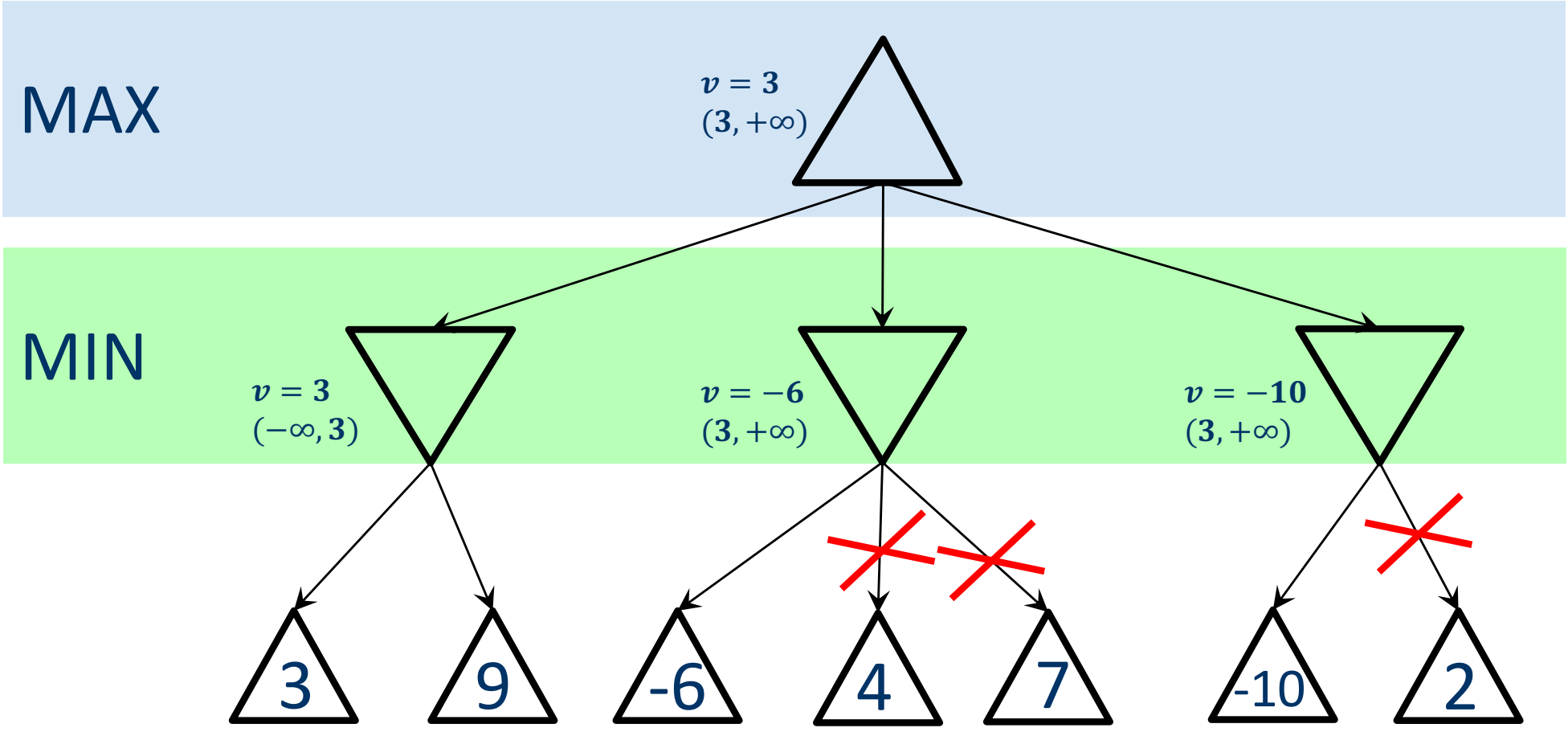
Extra 1: Depth-First Minimax



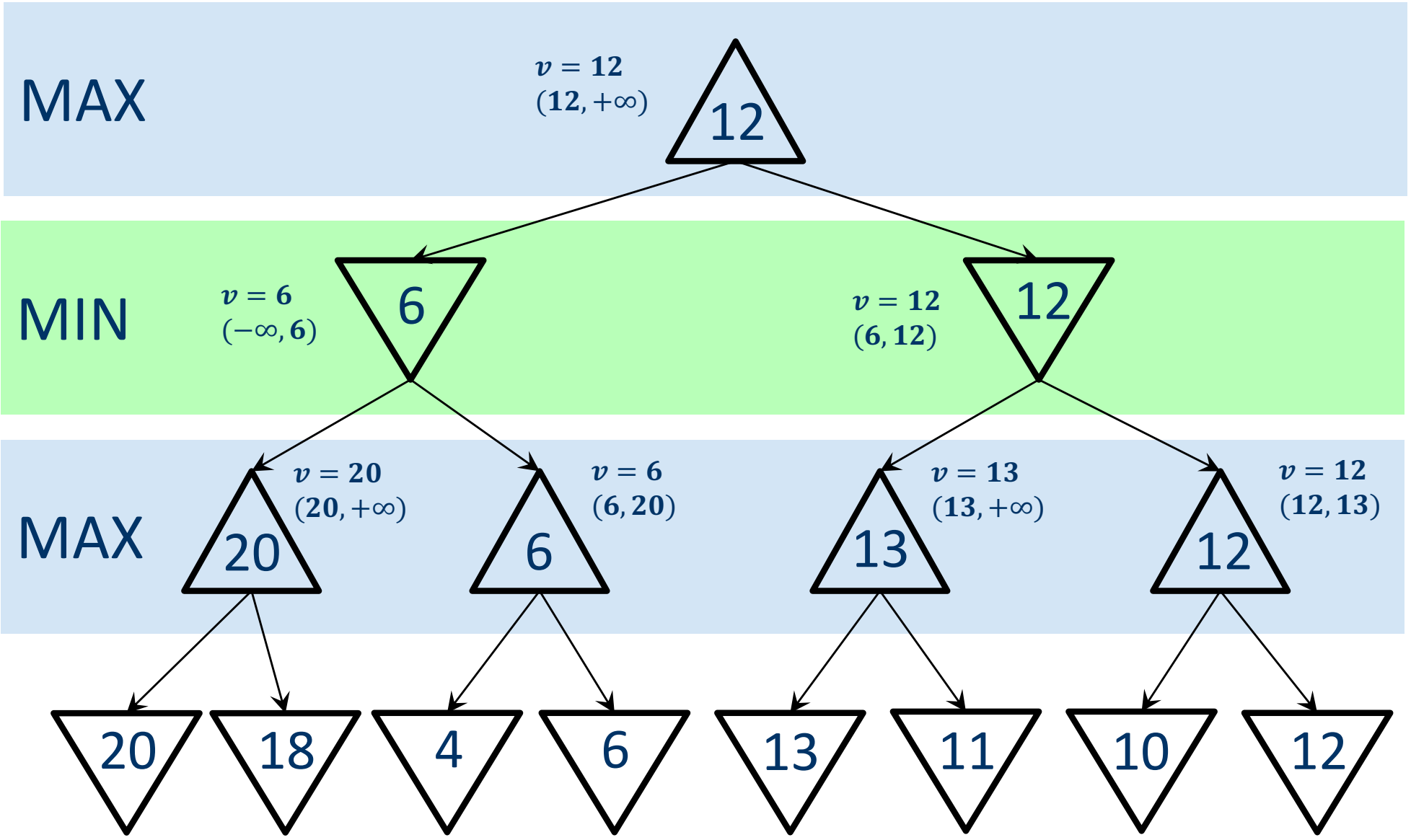
Extra 1: Alpha-Beta Pruning



Extra 1: Move Ordering to Increase Pruning



Extra Example 2



Extra 2: Increase Pruning