# Search 2: Strategies	start time:

This activity introduces some ideas and techniques used in artificial intelligence (AI) to search for possible solutions to problems.

Before you start, share this document with your team member(s) and then complete the form below to assign the role of speaker.

Team Role	Team Member
Speaker: shares your team's ideas with the class.	Makenna, Arogya, Nic

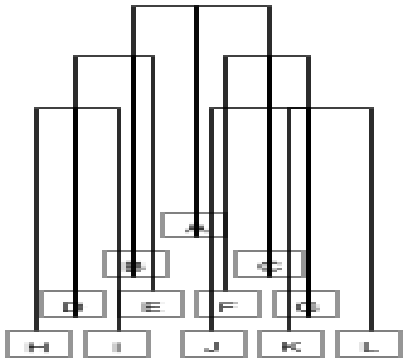
(12 min) A. Search Structure	start time:

1. (2 min) At any time during search, part of search space has been explored.
The boundary between this and the unexplored search space is the **frontier**.
Explain why the frontier (and its size and shape) is an important aspect of search.

Frontier tells us how many more searches are required aka which nodes have been searched and the connection between them

2. (2 min) A search of state space can be represented as a **tree** with a node for each state that has been explored. Many search algorithms use tree-based data structures.

a.	What is each **edge** (between two nodes)?	A searched connection
b.	Which state is the **initial state**?	A
c.	Which states are on the **frontier**?	All Children without children
d.	How do frontier states differ from non-frontier states?	Whether they were searched or not



3. (2 min) A search that explores all of state space is a **complete** or **exhaustive** search.
Why might **exhaustive search** not be practical?

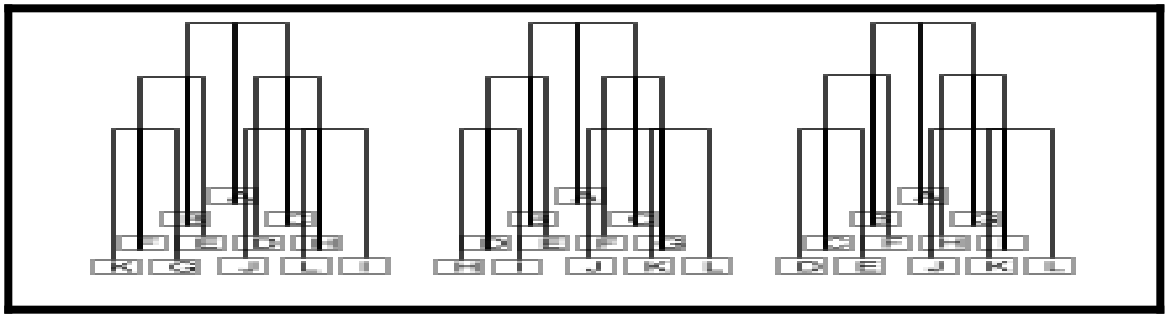
A large dataset that would take many many years to search through completely, anything that could rapidly change (aka many times a second while searching), the goal is very far away from the initial state

```
def search( problem ):
  # a.
  frontier = makeEmptyList()
  frontier.add( problem.getInitialState() )
  # b.
  while ( ! frontier.isEmpty() ):
    # c.
    nextState = frontier.removeNext()
    if ( problem.isGoalState( nextState ) ):
      return nextState
    # d.
    neighbors = problem.getNeighbors( nextState )
    frontier.addAll( neighbors )
  # e.
  return None
```

4. (5 min) The pseudo-code above shows the general structure of many search algorithms. Match each letter above with the appropriate comment below.

add neighbors to frontier and repeat	d
get next state from frontier; if goal is found, return it	c
if frontier is empty, return null	e
repeat until frontier is empty	b
start with a frontier containing just the initial state	a

(10 min) B. Search Strategies	start time:



1. (3 min) The pseudo-code shows when to add and remove frontier states, but not how to choose the next state from the frontier. This can be done in several ways. The above diagrams above illustrate three stategies, where the states are visited in alphabetical order. In **column 1 of the table below**, choose the diagram (*left*, *middle*, *right*) that best matches each strategy.

	Strategy Description	**1. Diagram**	**2. Name**
a.	Visit all states that are 1 edge away, then all states that are 2 edges away, etc.	Middle	Breathe first search
b.	Follow child states as deeply as possible. When the search reaches a state without children, it backtracks to an unvisited child and continues.	Right	Depth-first search
c.	Choose any frontier state at random.	Left	Random search

2. (2 min) These 3 strategies are called **random-first**, **depth-first**, and **breadth-first**.
In **column 2**, choose the name that best matches each description.

3. (4 min) Of these 3 strategies, decide which one(s):

a.	would find the solution with the minimum path length first?	BFS
b.	would be useful for comparison, but not very practical?	RFS
c.	would be likely to find solutions with very long path lengths, before solutions with shorter path lengths?	DFS
d.	would tend to stay in nearby areas of the frontier, rather than jump between different parts of the frontier?	BFS
e.	would most likely use a **queue** (first-in, first-out)?	BFS
f.	would most likely use a **stack** (last-in, first-out)?	DFS

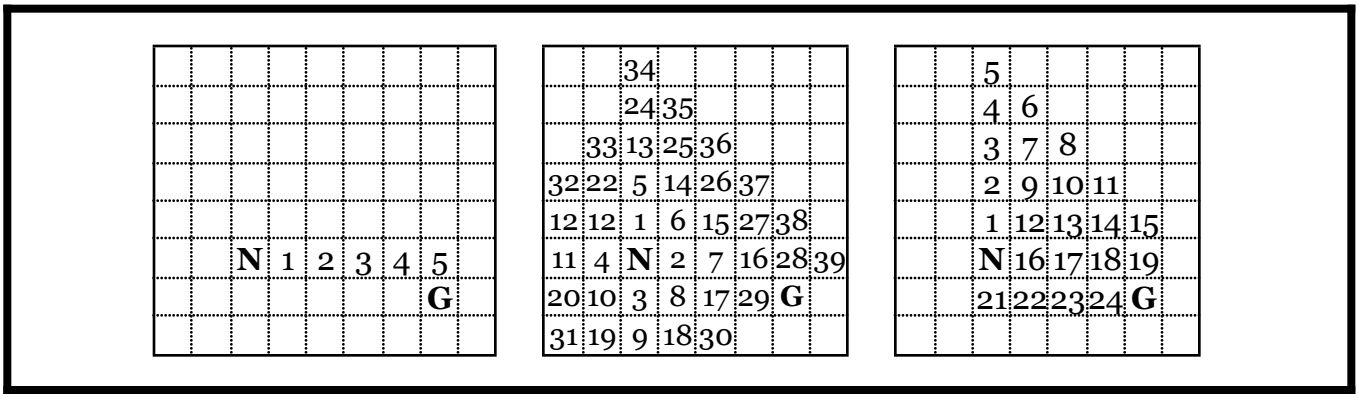
(14 min) C. Search Variations	start time:

1. (2 min) To save time, a search tries to avoid child states that can be excluded easily. Techniques to ignore or remove tree branches for efficiency are called **pruning**.
For example, a search could keep a list of every state it visits.

a.	How could this list save time?	Prevents searching through items, not in the frontier
b.	What problem could arise in a very large state space?	Takes a lot of space and can become inefficient to search through

2. (2 min) If **depth-first search** is limited to a **maximum path length**, it is called **depth-bounded search**. (Recall that **path length** is the number of actions on a path.)

a.	What is an advantage of depth-bounded search?	We can save time by not visiting unnecessary nodes if the bound is correctly estimated
b.	What might go wrong if the depth bound is too small?	We don't find a solution that exists further down
c.	What might go wrong if the depth bound is too large?	We can loop and waste time trying a path that won't work



3. (3 min) 3 search state space diagrams are shown above, starting at the initial state N and ending at the goal state G. Which one (*left, middle, right*) shows a possible sequence of search states for:

a.	a **breadth-firs**t search strategy	Middle
b.	a **depth-bounded** search strategy (lucky)	Left
c.	a **depth-bounded** search strategy (not lucky)	Right

4. (1 min) What was the maximum depth bounding the above depth-bounded searches?

5

5. (2 min) The **path cost function** could return the **path length**,
but it could be different if states or actions could have different costs
(e.g. the cost to visit a city, or the distance between cities).

Explain how **breadth-first search** might not return the minimum cost path.

Assuming that all paths are equal cost, if costs are different use a priority queue.

Breadth-first would return a path with the lowest number of nodes first but this may not be the path with the lowest cost. It may never find the path with the lowest total cost.

6. (4 min) Another strategy could balance **path cost** instead of **length**.
Note: this is **different** from always choosing the next action or state with the lowest cost.
Describe how this **lowest-cost-first search** works.

Lowest cost first search looks for the next lowest and lowest overall. By looking at the current cost and adding the cost for the next path available, it can use a priority queue to choose the lowest cost path overall.

(14 min) D. Search Direction	start time:

In a **word ladder** puzzle, the goal is to find a sequence of real words, from a **start word** to an **end word**, where each word differs by just one letter from the previous one. For example, a word ladder from PATH to MIST is:

PATH ⇒ MATH ⇒ MASH ⇒ MAST ⇒ MIST

1. (2 min) Make a word ladder from COLD to WARM.
COLD -> CORD -> CARD -> WARD -> WARM

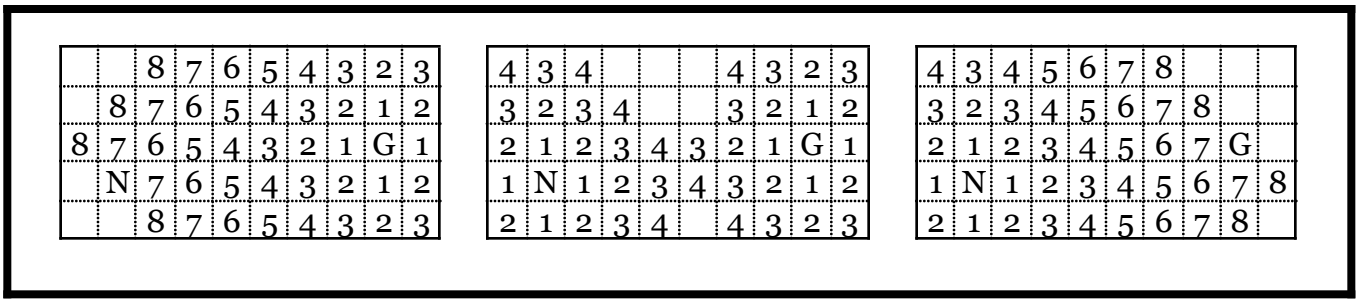
2. (2 min) In a word ladder puzzle:

a.	What is the **initial state**?	The first word
b.	What is the **goal state**?	The final word
c.	Does the structure of the puzzle change if we swap the initial and goal states?	No, it is still a four-letter word
d.	How many letters are in the English alphabet?	26
e.	How many 4-letter sequences are possible?	26^4 if unique
f.	There are ~5000 4-letter words in English. What percentage of 4-letter sequences are words?	5000/26^4 * 100 = approx 1.9%

3. (3 min) Assume that every state has applicable actions that lead to 2 new states. Starting from the initial state, how many states are:

a.	**1 action away**?	2
b.	**2 actions** away?	4
c.	**3 actions** away?	8
d.	**N actions** away?	2^n

e.	**within 1 action**?	3 = 1 + 2
f.	within **2 actions**?	5 = 1 + 4
g.	within **3 actions**?	9 = 1 + 8
h.	within **N actions**?	$1 + 2^n$



4. (2 min) 3 state space diagrams are shown above. Which one (*left, middle, right*) shows:

a.	path length from the initial state N?	Right
b.	path length from the goal state G?	Left
c.	path length from the initial and goal states?	Middle

5. (2 min) Explain why it could be much more efficient to search from the initial and goal states at the same time - this is **bi-directional search**.

You close the distance faster, distance being defined as number of changes required to get from goal to initial state.

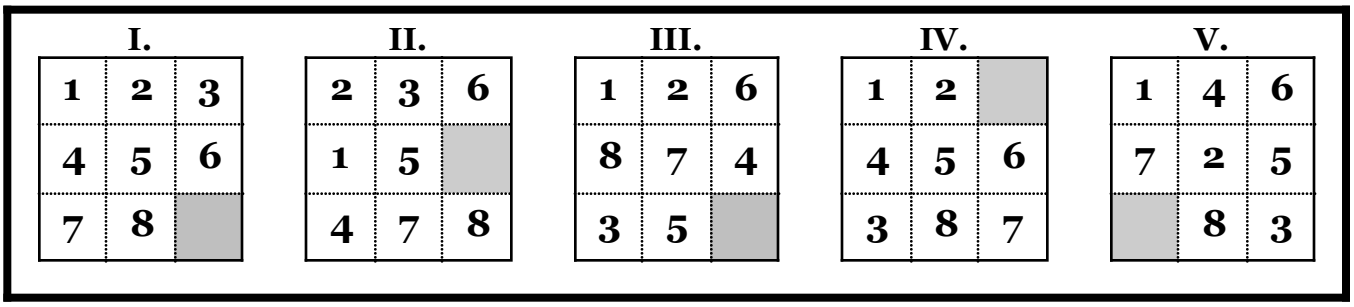
Reduces the state space significantly because its 2^(n/2) instead of 2^n

6. (2 min) Which of these search strategies could use **bi-directional search**?

a.	breadth-first search	Yes
b.	depth-first search	Yes
c.	lowest-cost-first search	No

(20 min) E. Heuristic Search	start time:

Lowest-cost-first, breadth-first, and depth-first search **don't know** if a path is a goal path until a goal state is reached; thus, these strategies are called **blind** or **uninformed**. Uniform-cost search prioritizes states on the frontier using an **evaluation function** which is often the **path cost function**, based on a path's length, states, and actions. Search can be better using functions that **predict** which paths are most promising, even if such predictions are not perfect. There are called **heuristic functions**, and these search strategies are called **informed** or **heuristic search**.



1. (3 min) The figure above shows 5 states for an **8-puzzle**, and I. (left) is the **goal state**.

Answer each question below for each state.		**I.**	**II.**	**III.**	**IV.**	**V.**
a.	Does the state match the goal state?	Y	N	N	N	N
b.	How many of the 8 tiles are in the wrong position?	0	7	6	3	7

2. (4 min) Refer to your answers above for the questions below.

a.	Could **1a** be a good **goal test** (to detect a goal state)?	Yes
b.	Could **1b** be a good **goal test**?	Yes
c.	Could **1a** be a good **heuristic** to find states **near** the goal?	No
d.	Could **1b** be a good **heuristic** to find states **near** the goal?	Yes
e.	Which state (II or IV) is **easier to solve** (closer to the goal)?	II
f.	Would **1b** be a useful heuristic to compare these 2 states?	Yes

3. (2 min) Explain why **1a** and **1b** are not ideal heuristics.

Because we only know some information about the goal state, aka where the numbers are supposed to be but not how to get there

frontier state:	**J**	**K**	**L**	**M**	**N**
cost & heuristic values:	c=2 h=8	c=10 h=2	c=4 h=4	c=8 h=2	c=2 h=10

4. (3 min) The table above lists 5 states (J…N) on the frontier, each with a **cost c** (from the **initial** state) and **heuristic h** (estimated cost to reach a **goal state**).

a.	Which states should be searched next, **based only on c**?	J or N
b.	Of these, which one should be next, **based on h**?	J
c.	Which states should be searched next, **based only on h**?	K or M
d.	Of these, which one should be next, **based on c**?	M
e.	Which states should be next, **based on both c and h**?	L

5. (4 min) Describe how to use a heuristic function in an evaluation function to improve (uninformed) **lowest-cost-first search**. The result is called **A* search**.

Estimate the total cost to be the cost so far plus the estimated heuristic value. (c+h)

6. (2 min) Describe at least one other **heuristic function** that could used for **8-puzzles**.

Based on how far the titles have to move to get to the goal state. - Would be an estimation of the total number of moves needed to solve the puzzle rather than how solved it currently is.

Manhattan distance is another heuristic function. It is the sum of horizontal and vertical distances.

There are many types of heuristic functions, and multiple ways to use them to improve search. We may explore some of them later in the course.