CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

UNINFORMED SEARCH

nicholas.teh@u.nus.edu

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

CONTENT SUMMARY

nicholas.teh@u.nus.edu

KEY CONCEPTS

- **Modelling a Search Problem**
  - Define the state, start/goal state, actions, transition model, cost function
  - Compute the size of state space (we will go through this next time)
- **Uninformed search algorithms**
  - Tracing
  - Completeness
  - Optimality
  - Time/Space Complexity

MODELLING A SEARCH PROBLEM

- **State**: Include essential variables (+ Initial and/or Goal State)
- **Actions**: Possible actions available to an agent at each state
- **Transition Model**: Description of what each action does (specified by a function that returns a state)
  RESULT(STATE, ACTION) = Resulting STATE
- **Goal Test**: Determine whether the state is a goal state (check whether it matches)
- **Cost function/Path cost**: Function that assigns a cost to each action (if Left/Right/Up/Down, then maybe cost = 1 is suitable)

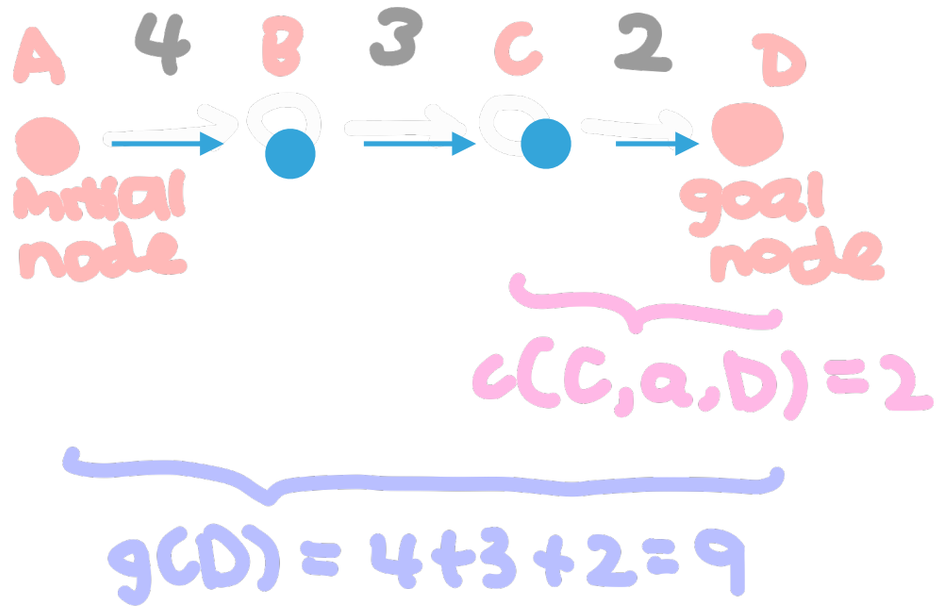
SEARCH: SPECIFYING THE SEARCH SPACE

- The search space is implicitly represented by a graph (with nodes and edges).
- **Branching factor (b)**: generally depends on how many actions are available at each state.
  - *usually* finite (remember write this assumption when required), but theoretically can be infinite.
- **Depth**: How many actions before we reach the goal?
  - Depth of shallowest goal node (d)
  - Maximum length of any path in the state space, i.e. maximum depth (m)
    - Can be finite or infinite.
      OUR ASSUMPTION: Goal node is always at a finite depth.

SEARCH: PROPERTIES OF COSTS

- **Path cost**: Total cost from initial state to some node n
- **Step cost**: Cost from state s to state s' via action a, denoted c(s, a, s')

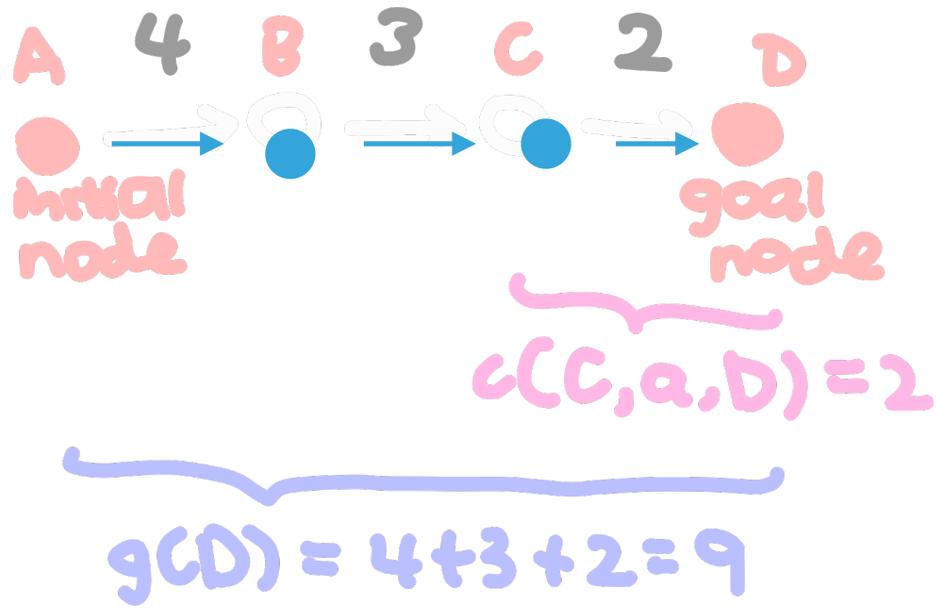
- We have g(s') = g(s) + c(s, a, s')
- Depends on the cost function
  - Uniform cost?
  - Increasing (e.g. with depth)?



SEARCH: PROPERTIES OF COSTS

- **Path cost**: Total cost from initial state to some node n
- **Step cost**: Cost from state s to state s' via action a, denoted c(s, a, s')

- We have g(s') = g(s) + c(s, a, s')
- Depends on the cost function
  - Uniform cost?
  - Increasing (e.g. with depth)?



SEARCH: GRAPH SEARCH VS TREE SEARCH

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

The only difference is keeping ***explored set***, and checking before adding to frontier

SEARCH: TRACING IN GENERAL

Goal test independent of exploration concept

- Frontier: nodes we've ***seen*** but ***not explored***
  - Initially the frontier consists of only the start state
- Select a node from the frontier [If UCS, tie-break is when pulling out of PQ], ***explore*** (or PRINT) it, add neighbours to frontier [If DFS/BFS tie-break is when pushing into frontier]
- Repeat until a goal state is explored

That is, unless the question gives a specific definition for visited/explored contrary to the above.

UNINFORMED SEARCH ALGORITHMS

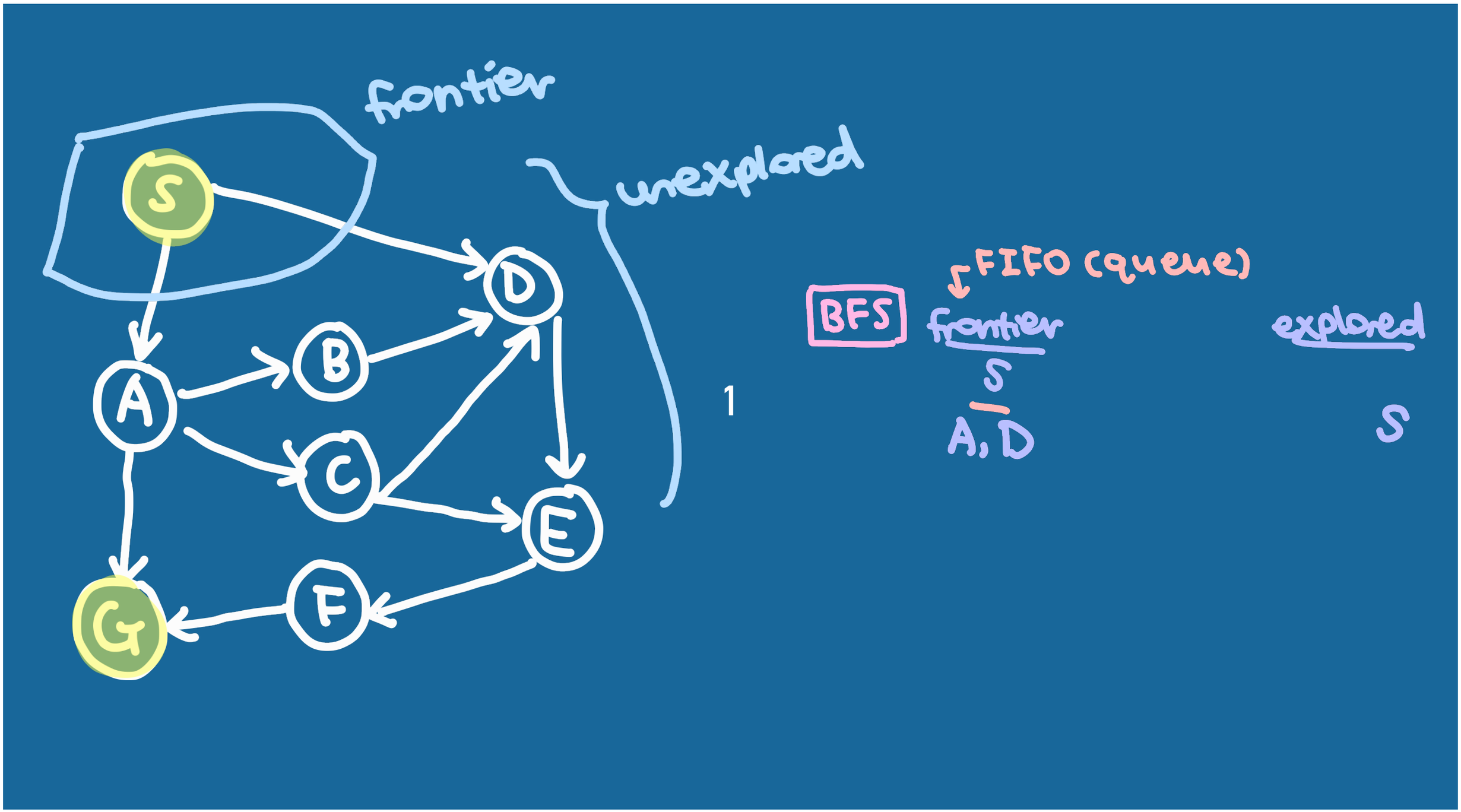
- **Breadth-First Search (BFS)**: Expand the shallowest node first (all neighbours first)
  - Use when the solution is near root or if the tree is deep but solutions are rare.
  - By default, goal test is right after popping, OPTIMIZED means goal test done when pushing into frontier (less nodes explored).
- **Uniform Cost Search (special case of Dijkstra's)**: Expand the least-path-cost unexpanded node (explore cheaper nodes by current path cost first)
  - Equivalent to BFS if all step cost are equal

UCS is Dijkstra's that is focused on finding a single shortest path to a single finishing point rather than the shortest path to every point.

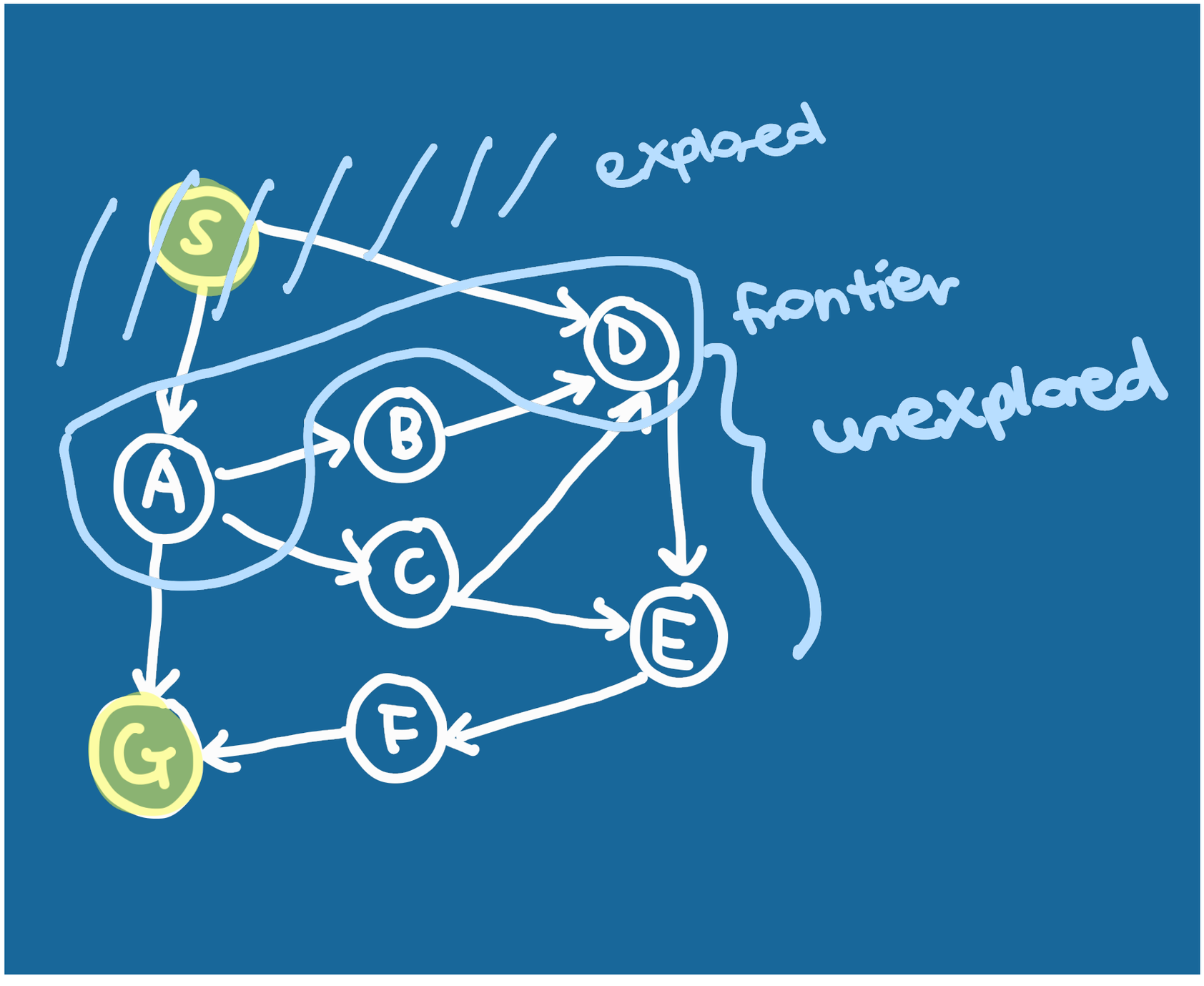
UNINFORMED SEARCH ALGORITHMS

- **Depth-First Search (DFS)**: Expand the deepest unexpanded node
  - Use when you don't care how you reach a node, you just want to reach it, and when the solution/goal node is very deep, or when all solutions are at same depth
- By default, use a stack (if ties broken alphabetically, output sequence is reverse alphabetical for children), OPTIMIZED means the (recursion) backtracking variant of DFS that does not require storage of all children.

TRACING UNOPTIMISED BFS (GRAPH SEARCH)

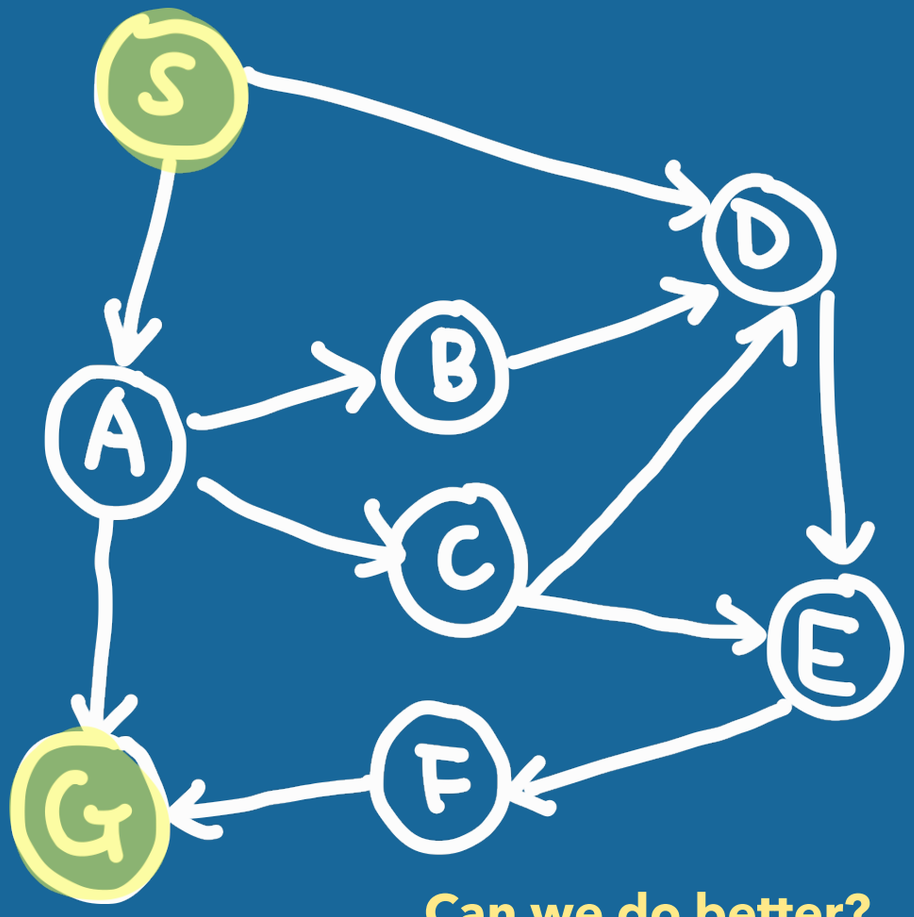


TRACING UNOPTIMISED BFS (GRAPH SEARCH)



- Frontier separates state space into explored and unexplored regions
- Every path from initial state to unexplored state must pass through a state in the frontier
- **Every step** moves 1 state from **frontier to explored**, and some states from **unexplored to frontier**
- Algorithm systematically explores states until a goal state is reached. BFS always has the shallowest path to every node on the frontier.

TRACING UNOPTIMISED BFS (GRAPH SEARCH)



Can we do better?
When are nodes expanded/ explored? When are they generated?

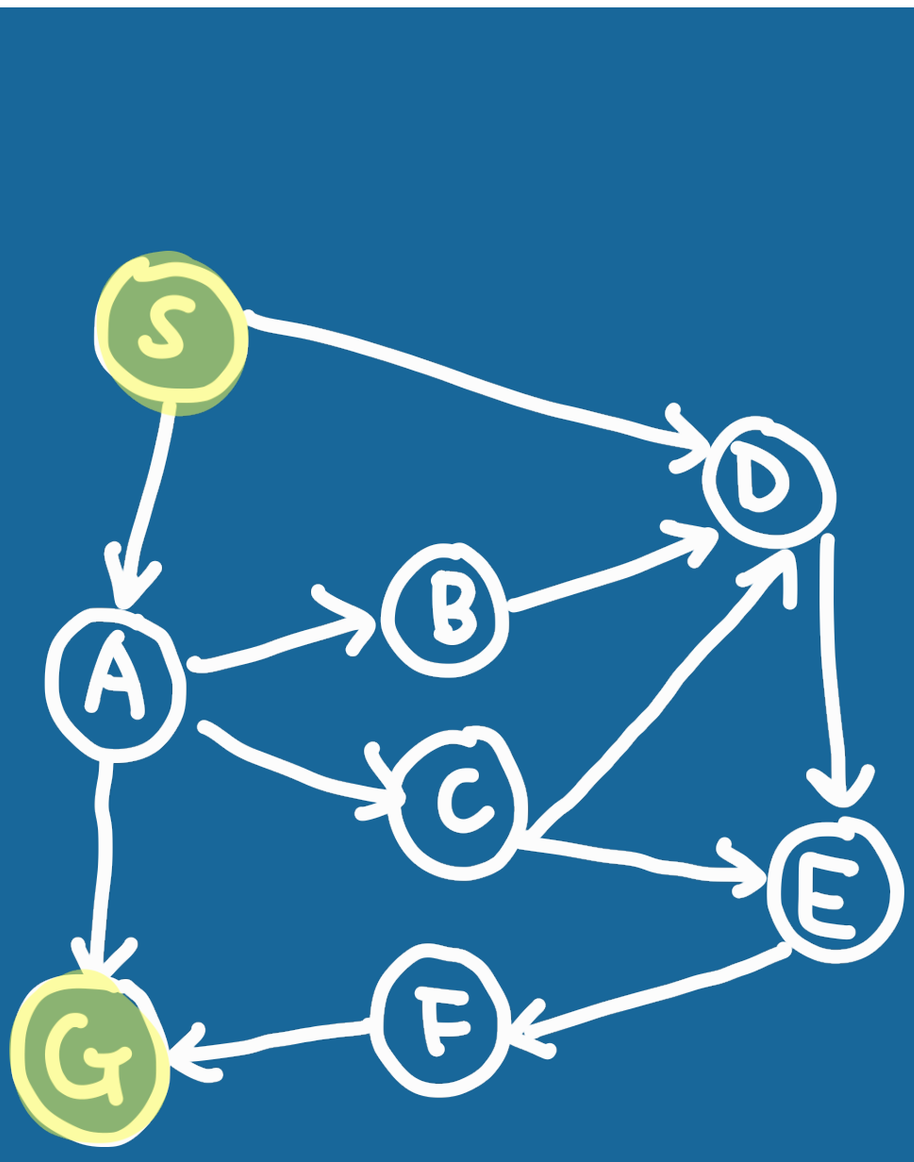
BFS

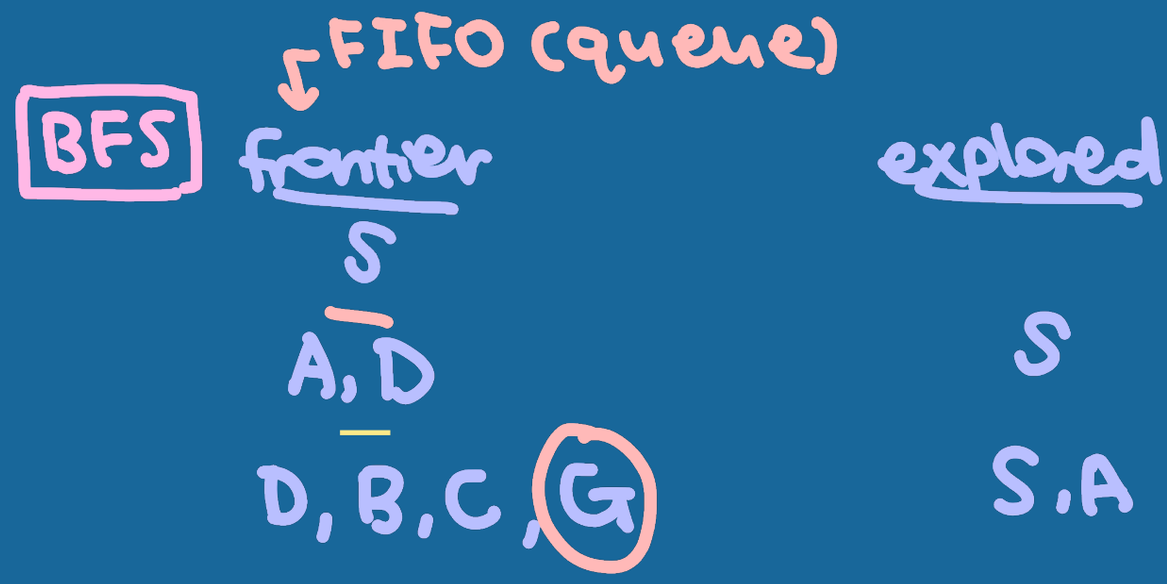
FIFO (queue)

frontier	explored
S	
A, D	S
D, B, C, G	S, A
B, C, G, E	S, A, D
↑ D is not added again ∵ It is explored	
C, G, E	S, A, D, B
G, E	S, A, D, B C
	S, A, D, B, C, G

goal test returns true

TRACING UNOPTIMISED BFS (GRAPH SEARCH)

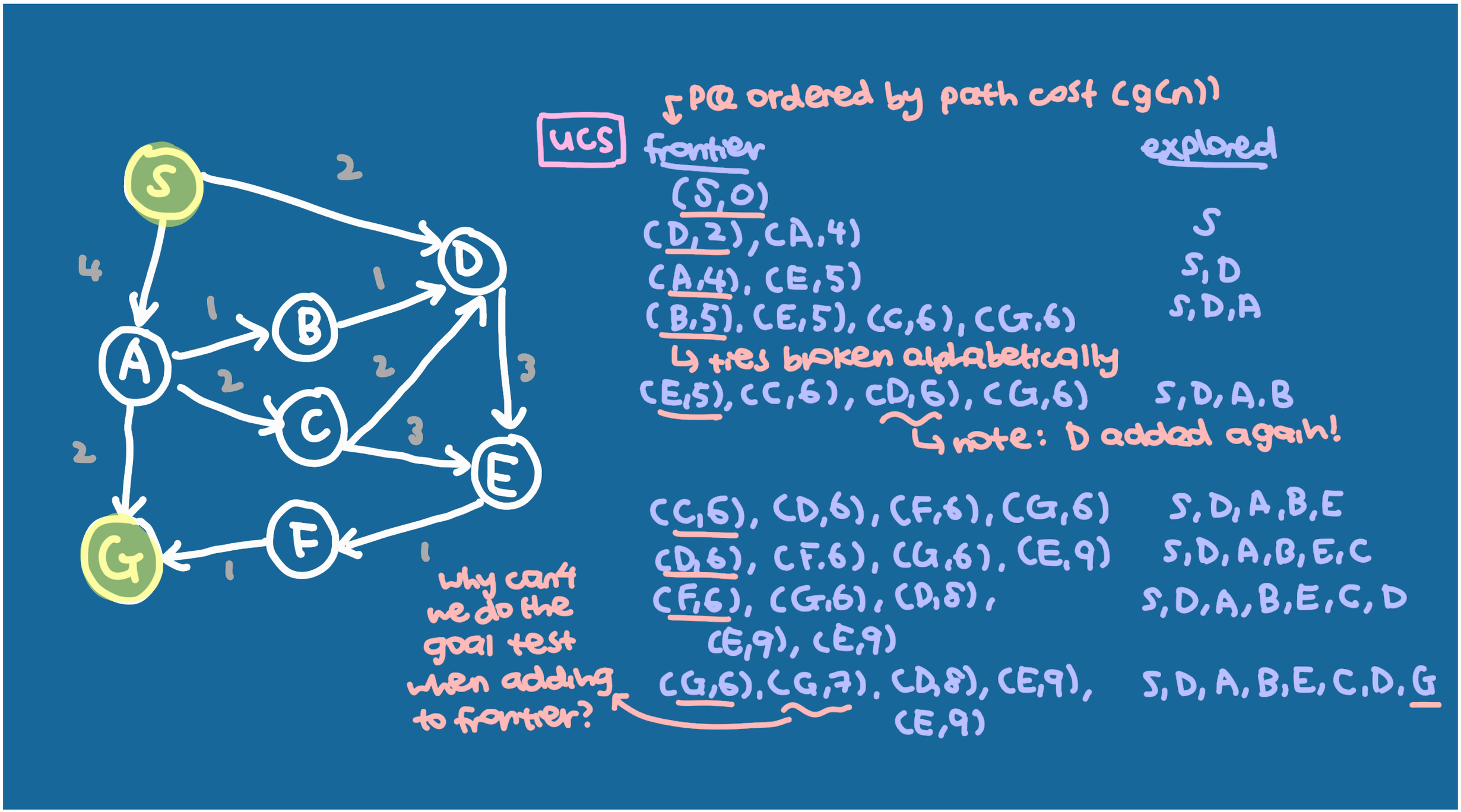




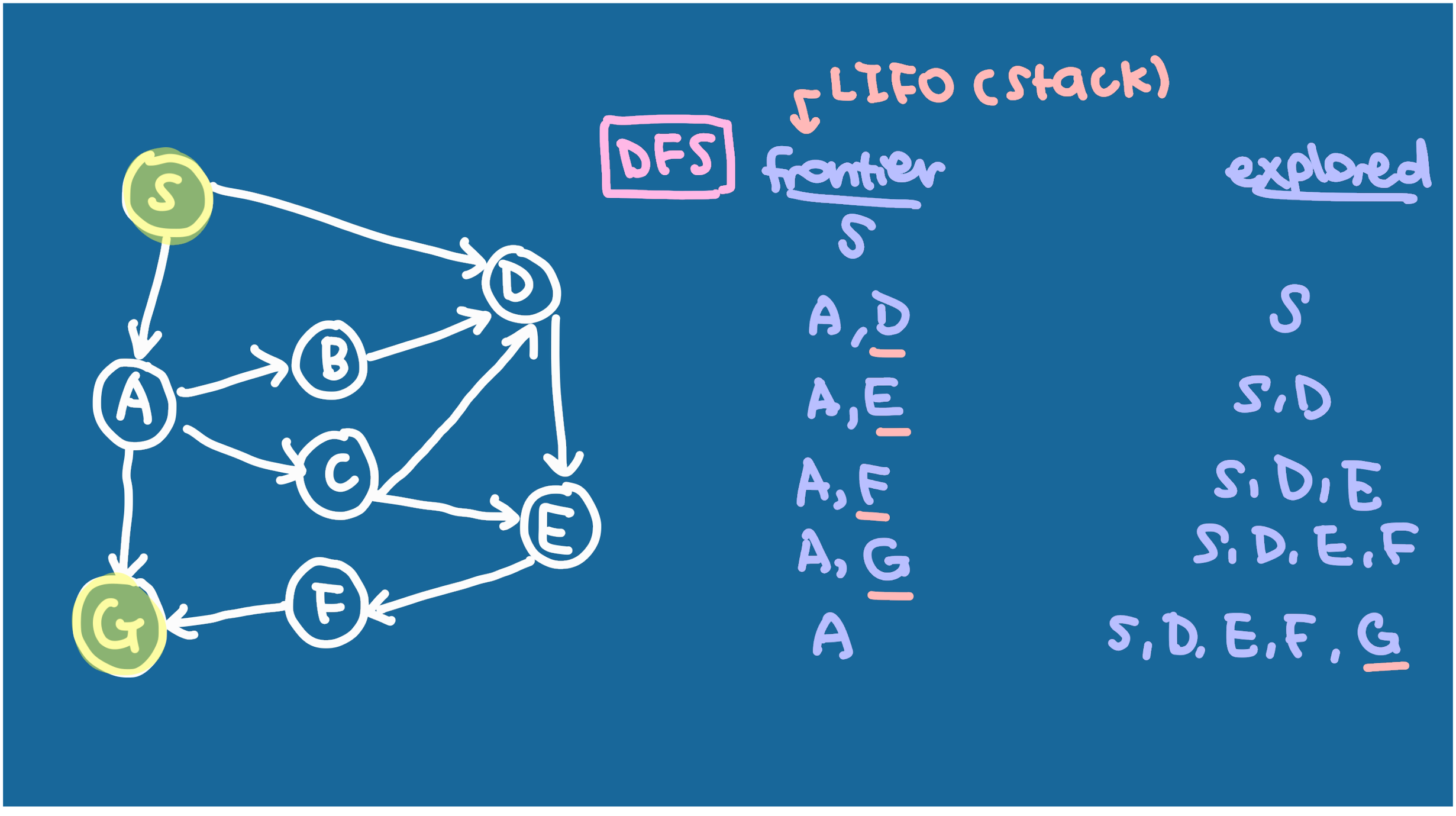
Optimised algorithm: goal test applied when nodes are added to frontier (Q4)

Time complexity reduced from $O(b^{d+1})$ to $O(b^d)$!

TRACING UCS (SELF-REVIEW)



TRACING DFS (SELF-REVIEW)



COMPLETENESS AND OPTIMALITY

- A search algorithm is **complete** if whenever there is a path from the initial state to the goal, the algorithm will find it (in finite time).
  - For uninformed search algorithms, only BFS and UCS are complete (subject to conditions).
- A search algorithm is **optimal** if it is always able to find a least cost solution. (*completeness have to be achieved first*)
  - For uninformed search algorithms, only UCS is optimal.

COMPLETENESS AND OPTIMALITY

Property	BFS	UCS	DFS
Complete	Yes[1]	Yes[2]	No
Optimal	No[3]	Yes	No
Time	$\mathcal{O}(b^d)$	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$	$\mathcal{O}(b^m)$
Space	$\mathcal{O}(b^d)$	$\mathcal{O}\left(b^{1+\left\lfloor \frac{C^*}{\varepsilon} \right\rfloor}\right)$	$\mathcal{O}(bm)$

- BFS is complete if *b* is finite.
- UCS is complete if *b* is finite and step cost ≥ ε.
- BFS is optimal if all step costs are identical

The time complexities here are generally for the optimised versions (BFS/DFS)

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

MODELLING SEARCH PROBLEMS

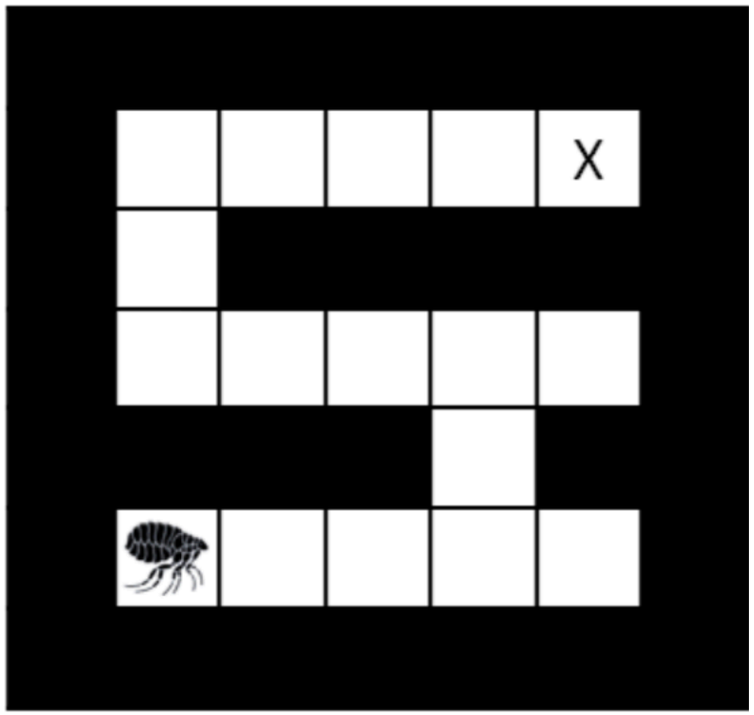
nicholas.teh@u.nus.edu

MODELLING A SEARCH PROBLEM

- **State**: Include essential variables (+ Initial and/or Goal State)
- **Actions**: Possible actions available to an agent at each state
- **Transition Model**: Description of what each action does (specified by a function that returns a state)
  RESULT(STATE, ACTION) = Resulting STATE
- **Goal Test**: Determine whether the state is a goal state (check whether it matches)
- **Cost function/Path cost**: Function that assigns a cost to each action (if Left/Right/Up/Down, then maybe cost = 1 is suitable)

EXAMPLE: FLEA IN A MAZE

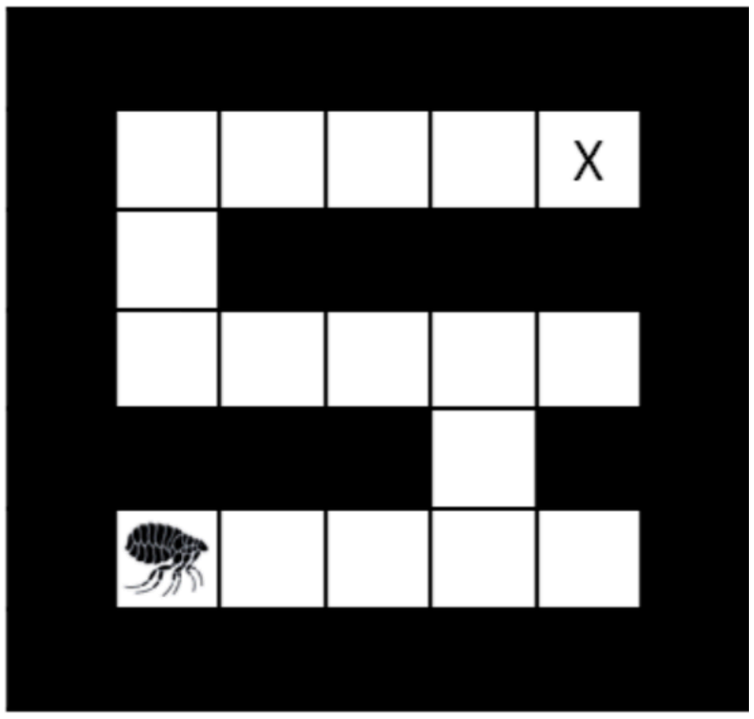
- **Modelling a search problem (Example)**
  - You control a single flea, which must reach a target location X. However, in addition to moving along the maze, your flea can jump on top of the walls. When on a wall, the flea can walk along the top of the wall as it would when in the maze. It can also jump off of the wall, back into the maze. Jumping onto the wall has a cost of 2, while all other actions (including jumping back into the maze) have a cost of 1. Note that the flea can only jump onto walls that are in adjacent squares (either north, south, west, or east of the flea).



What's the states, actions, transition model, goal test and cost function?

EXAMPLE: FLEA IN A MAZE

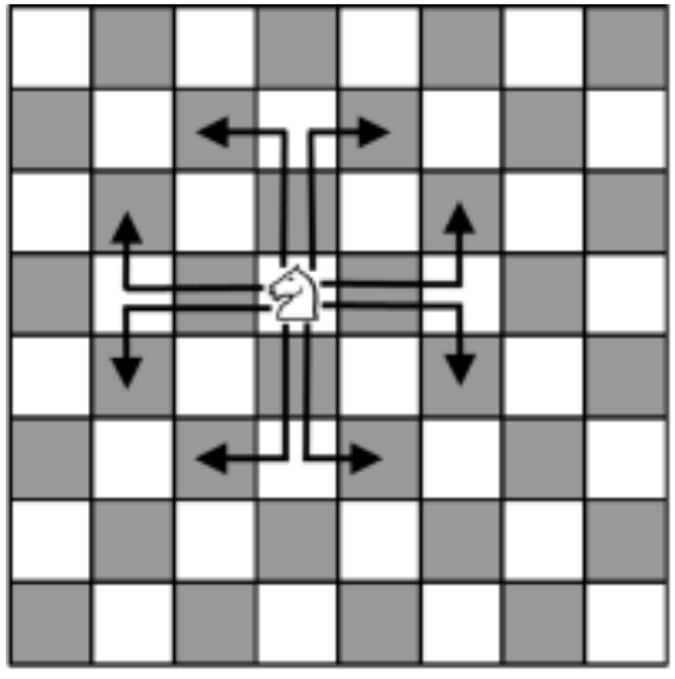
- States: location of flea as an (x, y) coordinate. **Initial state** of flea as given in question.
- Actions: L/R/U/D, jump up, etc. Flea cannot exceed boundaries, and only jump from floor
- Transition Model: Add/subtract x or y coordinate accordingly
- Goal Test: Check if flea is at X
- Cost Function: 1 if moving along maze walls, 2 if jumping onto walls



What's the states, actions, transition model, goal test and cost function?

PRACTICE QUESTION: KNIGHTS' TOUR

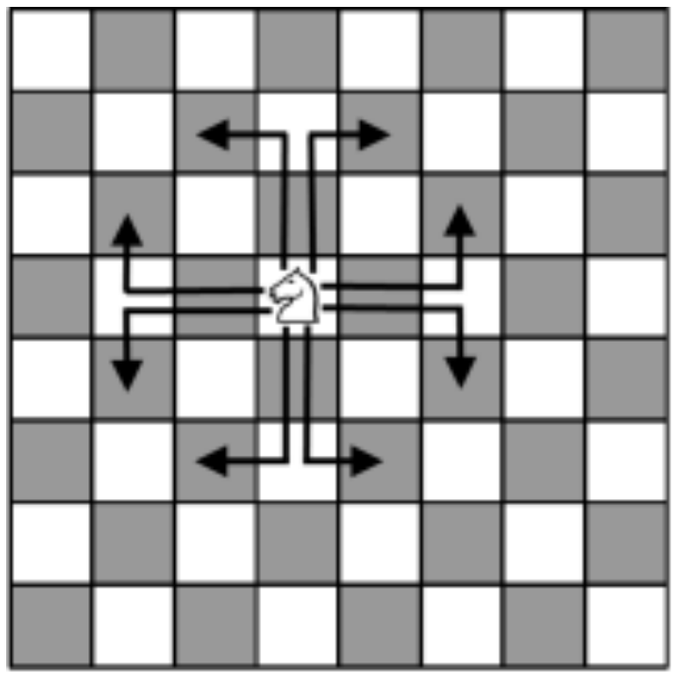
- In International Chess, the Knight piece is able to move as shown in the figure below.



- Given a chessboard of arbitrary size, n × n (e.g., the chessboard in Figure 1 is an 8 × 8 board), let a Knight's Tour be defined as follows:
- A knight is positioned at some starting position (ROW, COL), where 1 ≤ ROW, COL ≤ *n*
- To complete the tour, it must visit every square on the board (i.e., every position (r, c) where 1 ≤ r, c ≤ n) **EXACTLY ONCE**

PRACTICE QUESTION: KNIGHTS' TOUR

- **Model the Knight's Tour as an Uninformed Search Problem. Clearly define the states, actions, transition model, goal test and cost function.**

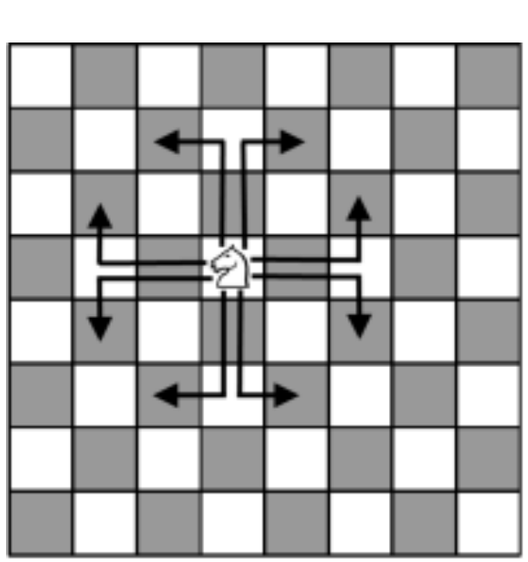


- State:
- Actions:
- Transition model:
- Goal test:
- Cost function:

PRACTICE QUESTION: KNIGHTS' TOUR

- **State**

- Have to specify initial state as well (even though not specifically asked!)

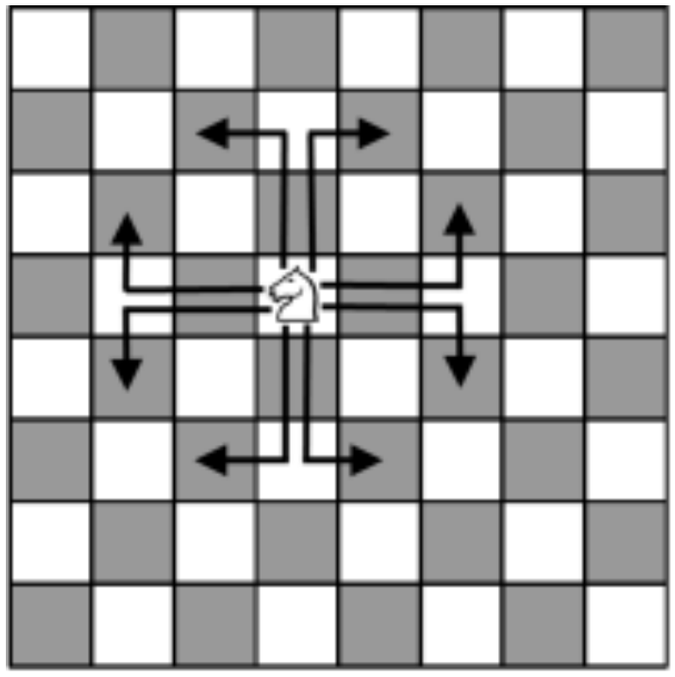


- The knight in a particular position (ROW,COL) and a set of visited squares.

- Initial State: The knight in the specified starting position. Visited squares set is empty.

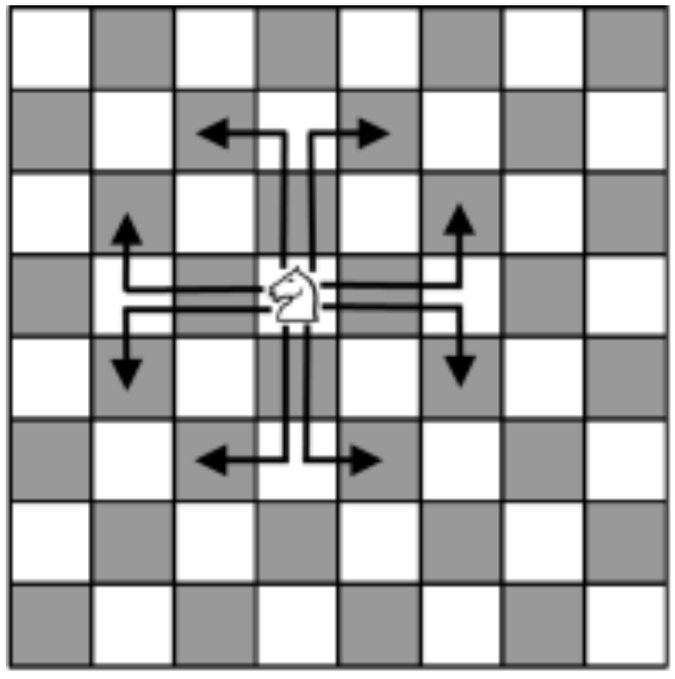
PRACTICE QUESTION: KNIGHTS' TOUR

- **Actions**
- The knight moving to a specified position following the movement rules of a knight (as in figure), without exiting the chessboard.
- You can define restrictions on actions here, or in the transition model. Two kinds of interpretation!



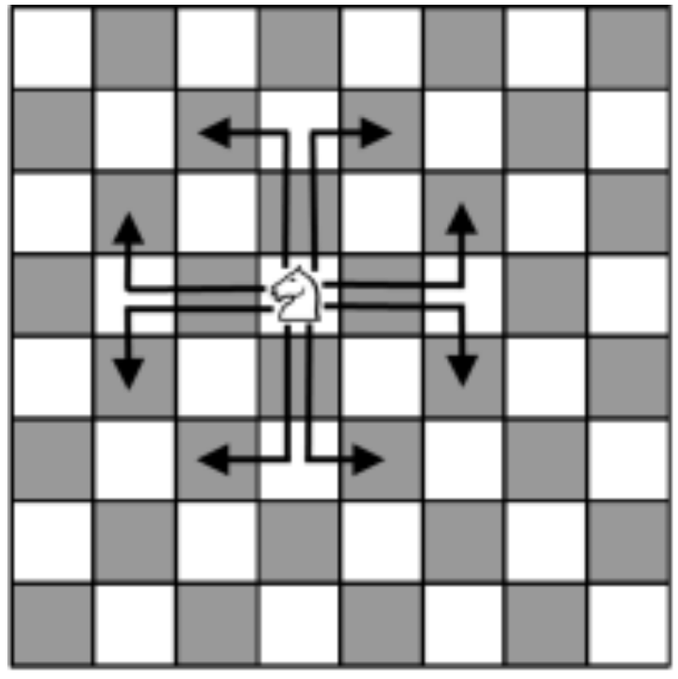
PRACTICE QUESTION: KNIGHTS' TOUR

- **Transition model**
- Take in the coordinate of the knight, decide on a new coordinate that can be reached by the knight and has not been visited yet. Update the coordinates of the knight according to the new position. Add the visited coordinate to the set.



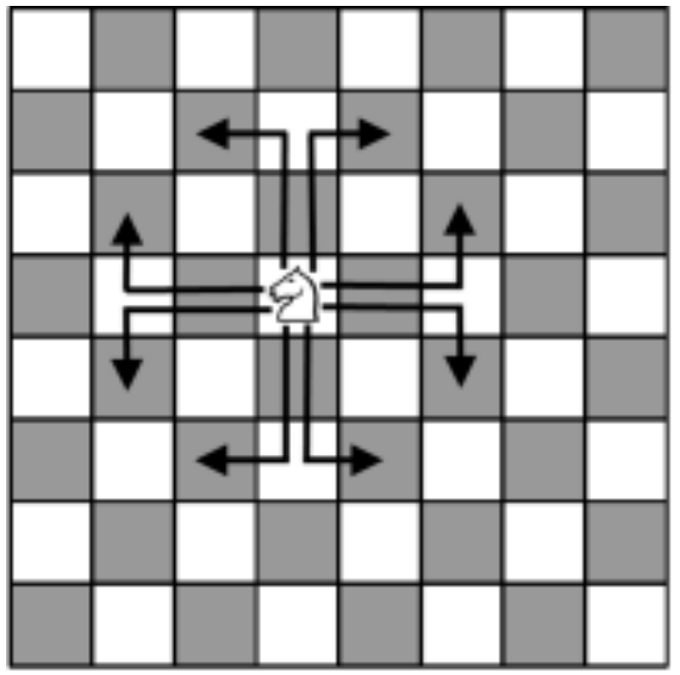
PRACTICE QUESTION: KNIGHTS' TOUR

- **Goal test**
- All the squares/coordinates have been visited exactly once.



PRACTICE QUESTION: KNIGHTS' TOUR

- **Cost function**
- Each step cost is 1.
- In fact, any positive constant will do.
- Note this may not always be the case (even though most problems are so)
- I'll show an example next time.



TUTORIAL 2 QUESTION 1 (GENERATING SUDOKU)

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

ANSWER:

- **State**: Any partial valid assignment of numbers to squares such that the constraints mentioned are satisfied. (More formally, a matrix, …)
- **Initial State**: Completely filled out valid assignment of numbers (1, …, 9) to squares such that the constraints are satisfied

  Goal State: A partial valid assignment of numbers to squares such that the constraints mentioned are satisfied, and there's only one way to complete the puzzle

TUTORIAL 2 QUESTION 1 (GENERATING SUDOKU)

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

- **Action**: Removing a number from the grid
- **Transition Function**:
  Take in a state (partially filled grid), and an action to be taken, the corresponding number to be removed will result in a new state that's the same as before, with just the number at that square removed.

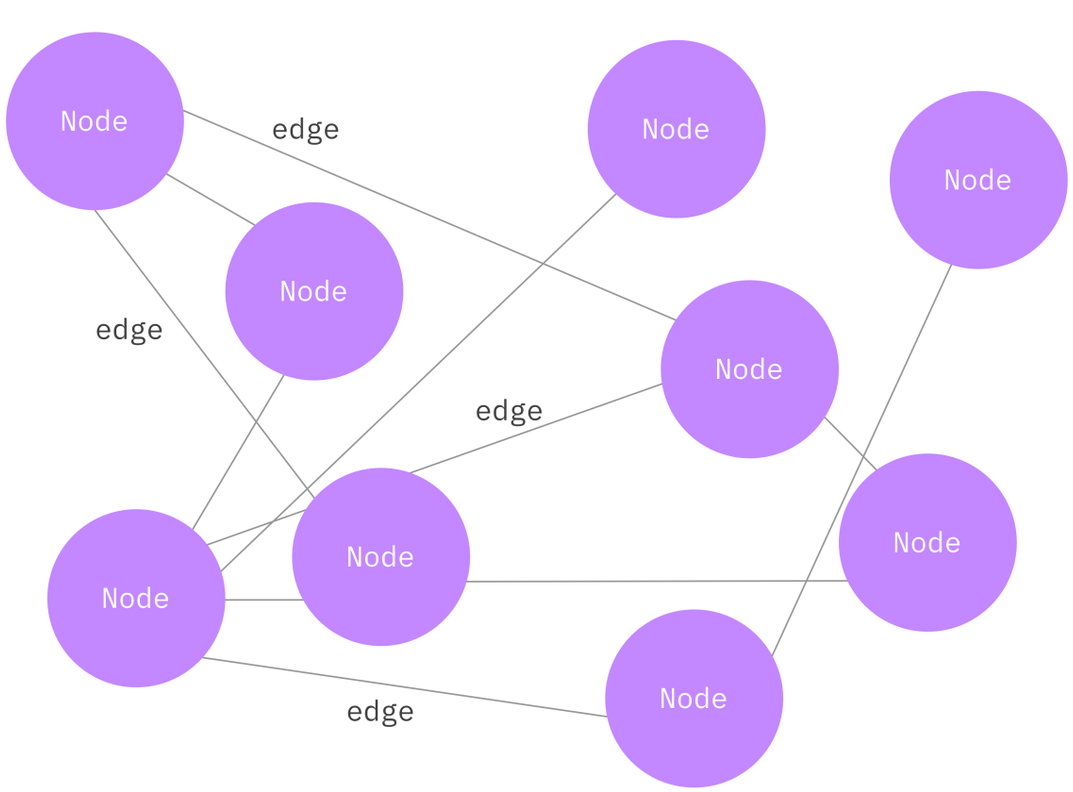
CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

UNINFORMED SEARCH ALGORITHMS

nicholas.teh@u.nus.edu

STATE VS NODE

- **Explain the difference between a state and a node.**



STATE VS NODE

- **Explain the difference between a state and a node.**

ANSWER:

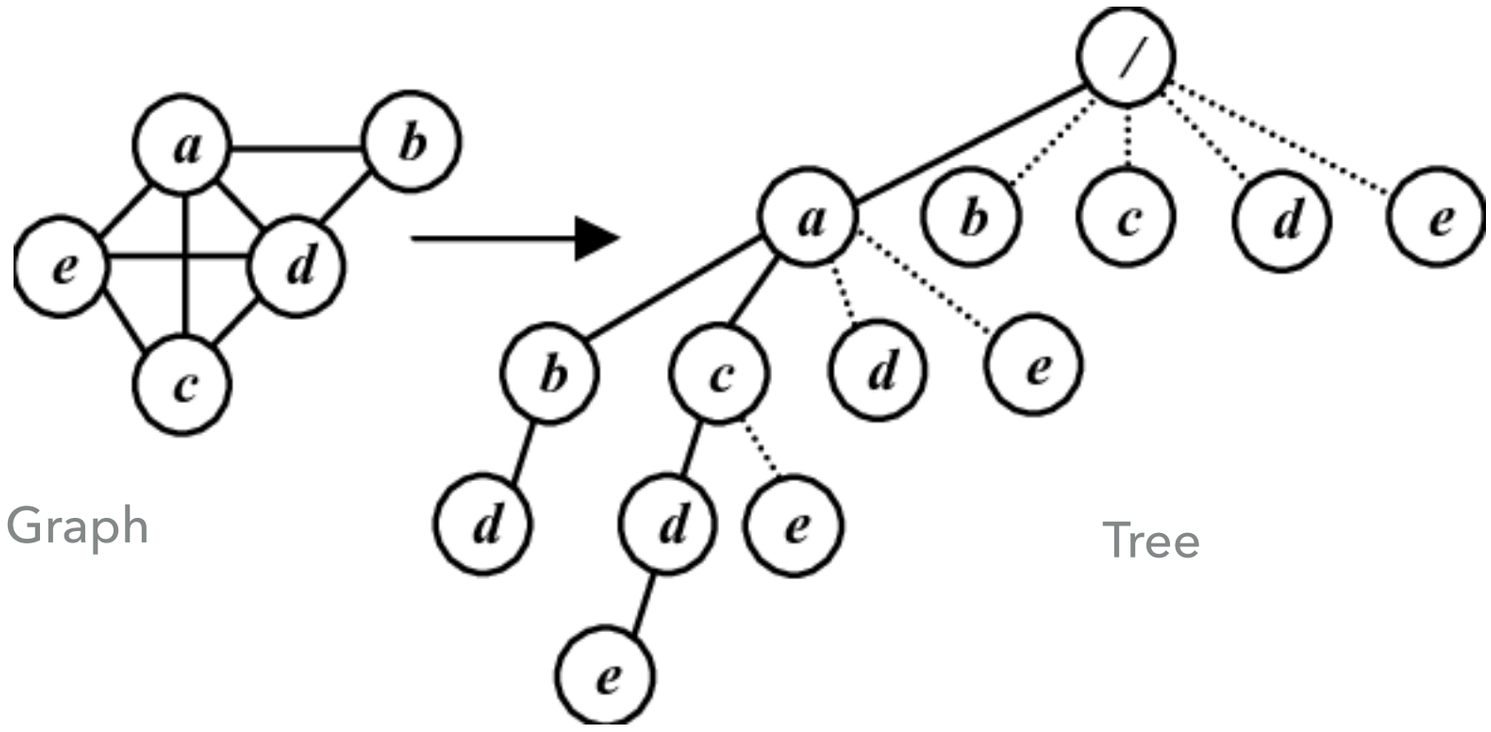
- **State**: a representation of the environment (freeze time and save it - that's a state of the problem)
- **Node**: Data structure to represent a state (we can have multiple nodes referring to the same state, but one node cannot represent multiple states)
- Edges from a node to another will then model the transition from a state, taking an action, to go to another state.

A collection of nodes will then form a graph, we implement search algorithms to search for the goal node/state, starting from the initial node/state.

TREE SEARCH VS GRAPH SEARCH

- **Describe the difference between TREE-SEARCH and GRAPH-SEARCH.**

  (A tree is a special kind of graph)



Graph

Tree

TREE SEARCH VS GRAPH SEARCH

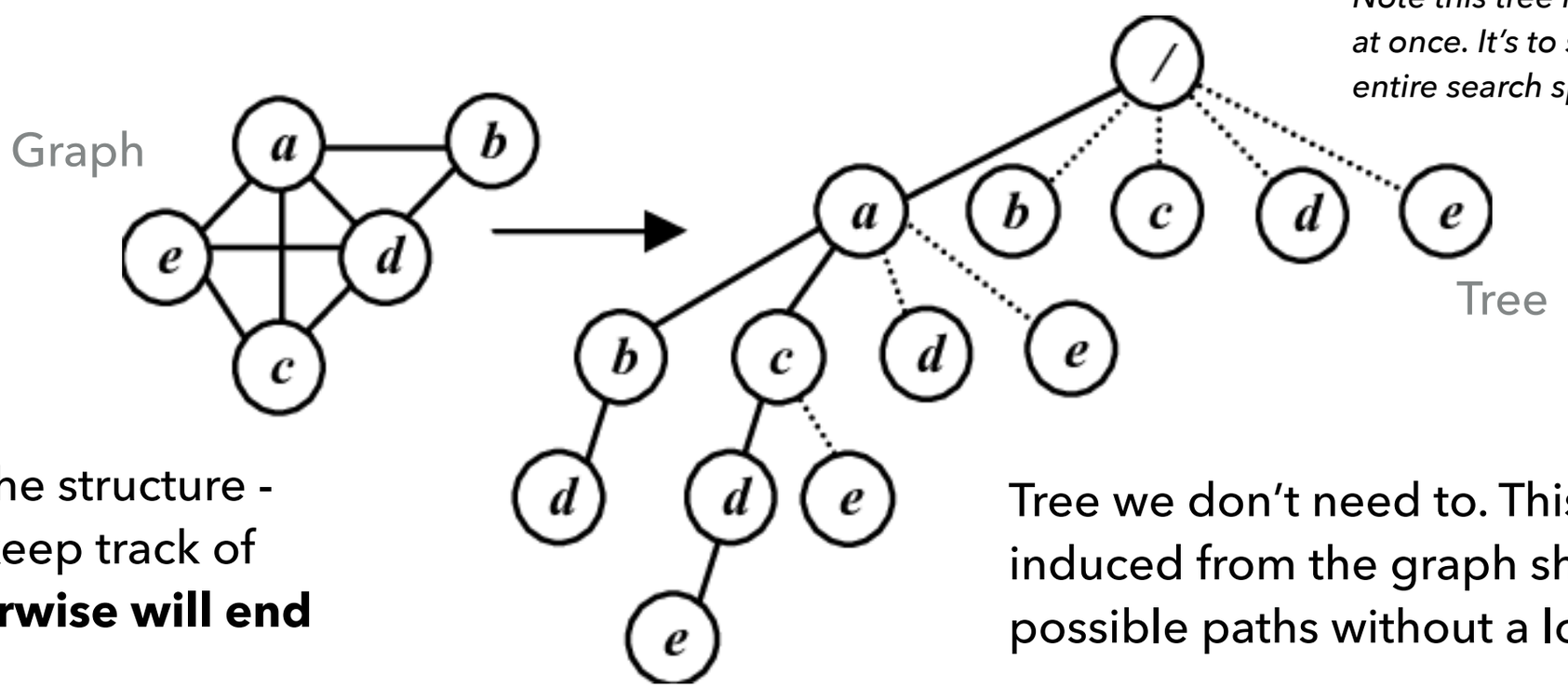
- **Describe the difference between TREE-SEARCH and GRAPH-SEARCH.**

ANSWER:

- The difference between the two is how we are traversing the search space (that is represented as a graph) to search for our goal state.
- Graph search maintains a list of visited nodes, so it only visits each node once
- Tree search does not, so it may revisit the same node multiple times

TREE SEARCH VS GRAPH SEARCH

- **Describe the difference between TREE-SEARCH and GRAPH-SEARCH.**



Note this tree is not generated at once. It's to show you the entire search space

Also evident from the structure - graph we need to keep track of visited nodes, **otherwise will end up in infinite loop**

Tree we don't need to. This tree induced from the graph shows us all possible paths without a loop.

But **this also means that we don't cut halfway** - we may visit same node multiple times until the goal node is found.

Different ways of searching!

TREE SEARCH VS GRAPH SEARCH

- **Describe the difference between TREE-SEARCH and GRAPH-SEARCH.**

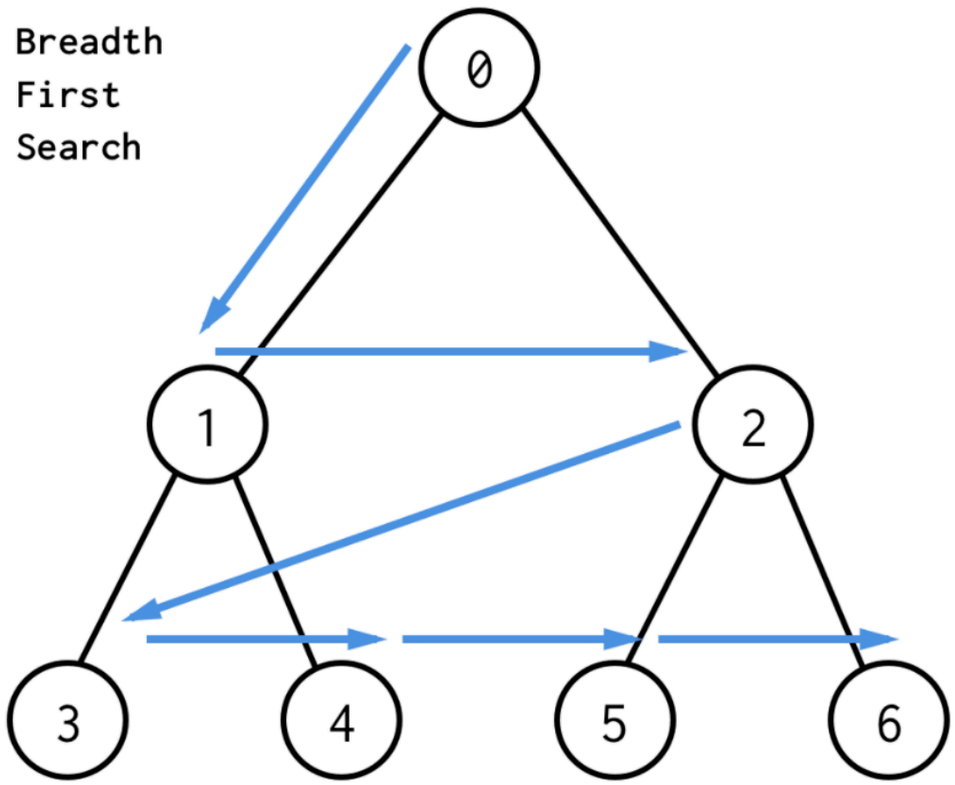
Advantage of using graph search: save on computational time, we don't revisit nodes again (not so in tree search)

Disadvantage of using graph search: Uses more space (keep track of visited nodes)

SPACE-TIME Tradeoff

BFS GOAL TEST

- **Why does the BREADTH-FIRST-SEARCH algorithm apply the goal-test when pushing to the frontier (as opposed to popping from the frontier like the default tree-search and graph- search algorithms)?**



BFS GOAL TEST

- **Why does the BREADTH-FIRST-SEARCH algorithm apply the goal-test when pushing to the frontier (as opposed to popping from the frontier like the default tree-search and graph- search algorithms)?**

ANSWER

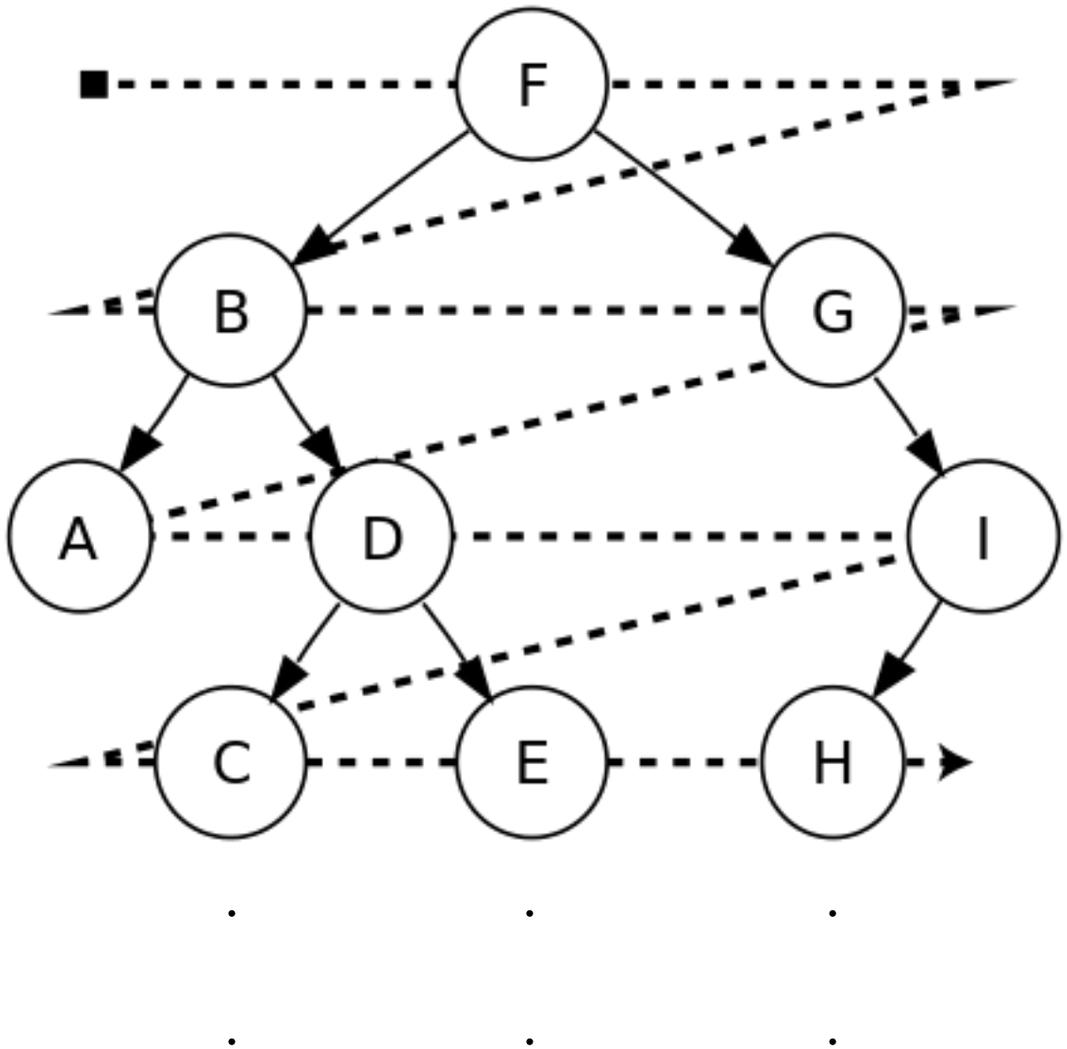
- Save space!
- If I check whether a state is the goal BEFORE pushing to frontier queue, then I can immediately return, no need to expand everything else in the frontier before reaching it.

ANSWERING TRUE/FALSE PROVE/DISPROVE QUESTIONS

- **State** whether it's **true** or **false**.
- **Define** key terms within the question.
- Extract key terms within the definition relevant to answering the question / **explicitly describe** how it matches/contradicts phrases in the statement.
  OR
  Find counterexample
- **Conclude**.

AY19/20 S2 MIDTERM QUESTION

- **TRUE/FALSE: The BFS algorithm is complete if the state space has infinite depth but finite branching factor.**



AY19/20 S2 MIDTERM QUESTION

- **TRUE/FALSE: The BFS algorithm is complete if the state space has infinite depth but finite branching factor.**

ANSWER

- **True.**
- Follows directly from the definition of BFS.
- A search algorithm is complete if whenever there is a path from the initial state to the goal, the algorithm will find it.
- Finite branching factor means BFS won't get stuck in one level, and so if the goal node is at a finite level such that there exists a path, BFS will find it.

AY19/20 S2 MIDTERM QUESTION

- **TRUE/FALSE: Given that a goal exists within a finite search space, the BFS algorithm is optimal if all step costs from the initial state are non-decreasing in the depth of the search tree. That is, for any given level of the search tree, all step costs are greater than the step costs in the previous level.**

ALSO AY19/20 S2 MIDTERM QUESTION

- **TRUE/FALSE: Given that a goal exists within a finite search space, the BFS algorithm is optimal if all step costs from the initial state are non-decreasing in the depth of the search tree. That is, for any given level of the search tree, all step costs are greater than the step costs in the previous level.**

ANSWER

- **False**. Question talked a lot about step costs between levels, not within level. (It's true BFS expand level by level, so **level-wise** its ok..)
- Consider 2 nodes at the **same level**, but with different cost. It may be that BFS expand the one with higher cost through arbitrary tie breaking.
- Because BFS doesn't check for cost like some others (e.g. UCS).

TUTORIAL 2 QUESTION 3

- **Prove that the UNIFORM-COST-SEARCH algorithm is optimal as long as each step cost exceeds some small positive constant ε.**

TUTORIAL 2 QUESTION 3

- **Prove that the UNIFORM-COST-SEARCH algorithm is optimal as long as each step cost exceeds some small positive constant ε.**

ANSWER

- In order for it to be optimal, it has to be complete.
- (1) Prove **completeness**
- (2) Prove **optimality** *given completeness*

TUTORIAL 2 QUESTION 3

- **Prove that the UNIFORM-COST-SEARCH algorithm is optimal as long as each step cost exceeds some small positive constant ε.**

ANSWER

- **Prove completeness**: Given that every step cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach a solution if it exists.

TUTORIAL 2 QUESTION 3

- **Prove that the UNIFORM-COST-SEARCH algorithm is optimal as long as each step cost exceeds some small positive constant ε.**

ANSWER

- **Prove optimality**: Assume UCS is not optimal. Then there must be an (optimal) goal state with path cost smaller than the found (suboptimal) goal state (invoking completeness). However, this is impossible because UCS would have expanded that node first by definition. Contradiction.
- You can also use the ε-contour example, showing that the algorithm always expands nodes with least cost first, so will radiate from the initial node outward like contours - i.e., always see a goal in an inner contour before an outer one.

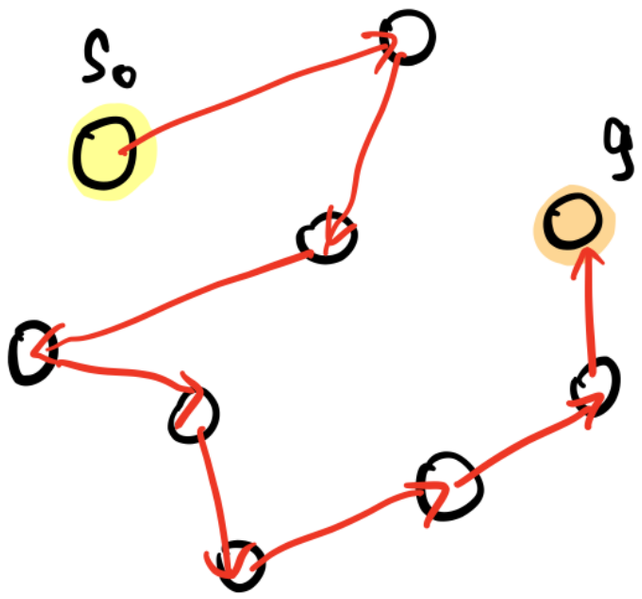
TUTORIAL 2 QUESTION 4

- **Prove that *any deterministic search algorithm*, will, in the worst case, search the entire state space.**
- Equivalent to proving the following theorem:

Let *A* be some complete, deterministic search algorithm. Then for any search problem defined by a finite, connected graph G = <V, E>, there exists a choice of start node s_0 and goal node *g* such that A searches through the entire graph G.

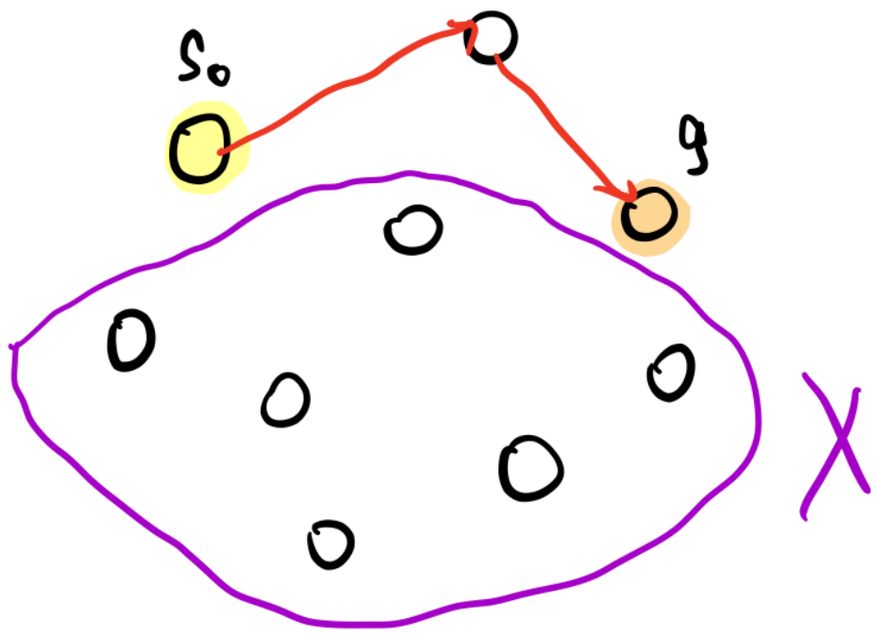
TUTORIAL 2 QUESTION 4

- An adversarial (worst-case construction) approach.
- Suppose I run the search algorithm A with a fixed starting point s_0, and a randomly chosen all node g.
- If it searches through the entire search space, good. I have my two points!



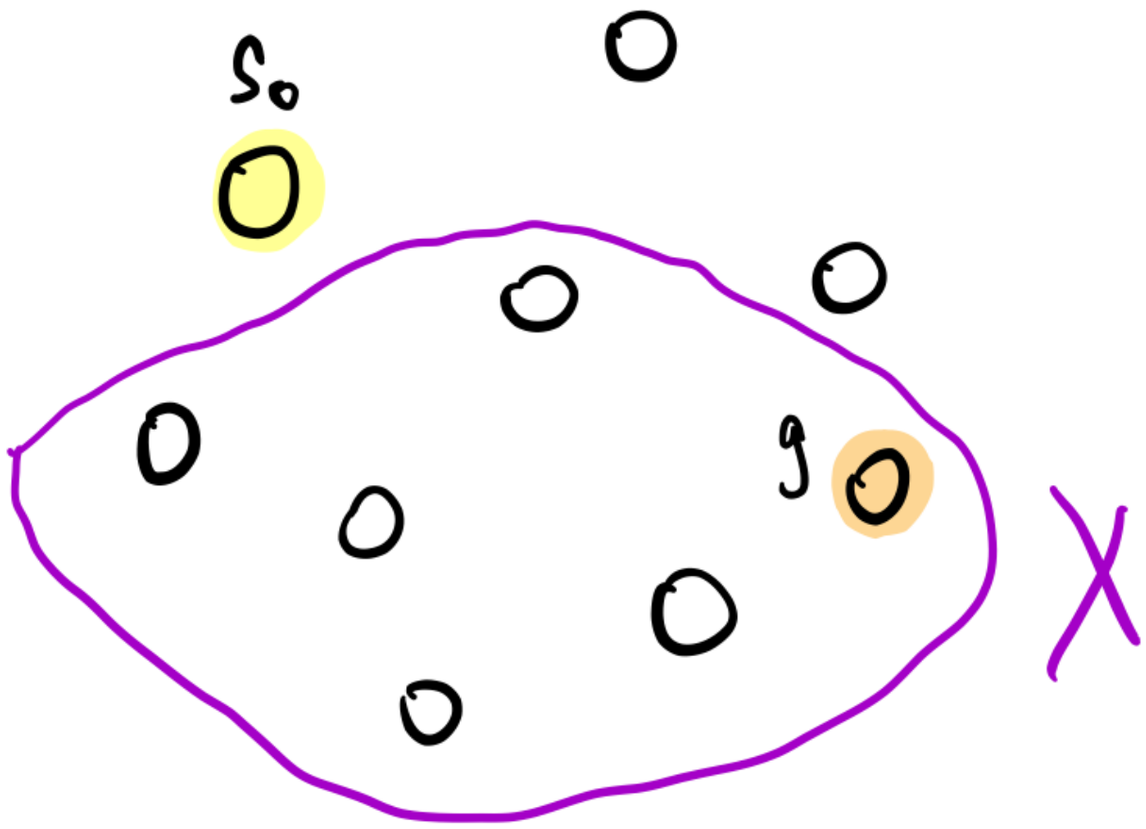
TUTORIAL 2 QUESTION 4

- If it doesn't search through the entire state space, that means there exists a set of vertices that the algorithm does not traverse, and yet it can reach from the start node to goal node. Let this set of vertices be X.



TUTORIAL 2 QUESTION 4

- Now, I change my choice of goal node to be a vertex in this set X.

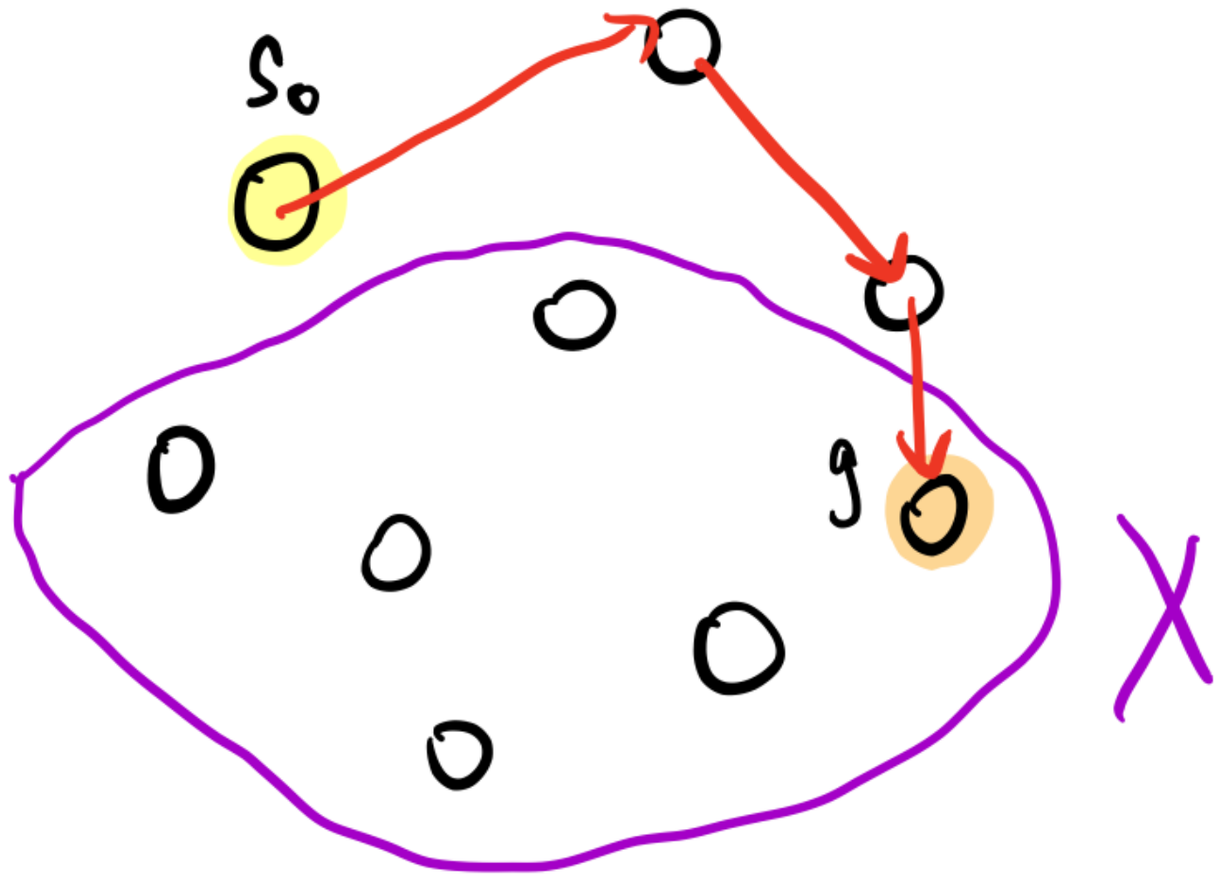


TUTORIAL 2 QUESTION 4

- Since the algorithm is deterministic and complete, it will run the same search order that it did when previous node was the goal, and then search through (some) nodes in X till it reach new g.

  In the worst case, I don't "clear" any other nodes in X, just that new goal node.

- Redefine X.



TUTORIAL 2 QUESTION 4

- But the set of nodes is finite, so at every iteration, I minimally clear one node from X by changing the goal node. (X decreases in size by at least 1 at each step)
- So after a fixed number of iterations (repeat the argument), X will become the empty set and the algorithm would have traversed through all nodes.

TAKEAWAYS

- **Modelling search problems**, and computing size of state space
- Understand searching as a tool to finding goal state
- **Tracing various uninformed search algorithms**, their completeness, optimality and what is it good for.
- Know which works/doesn't work in corner cases (infinite branching factor, infinite depth)
- Answering True/False with justification problems :)