CS3243 - Introduction to Artificial Intelligence

Tutorial 3

Theodore Leebrant

Tutorial Group 3

Key Concepts

Overview

- **Modelling a search problem**
  - Define state, start/goal states, set of actions, transition model
  - Compute size of state space
- **Uninformed search space**
  - Tracing
  - Completeness
  - Optimality
  - Space and Time Complexity

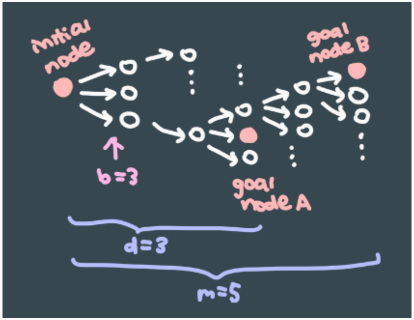
Modelling a Search Problem

- **State**: include essential variables (+ Initial / Goal state(s))
- **Actions**: possible actions that the agent can do
- **Transition model**: description of what each action does (specified by a function that returns a state).
  `RESULT(STATE, ACTION) => RESULT_STATE`
- **Goal test**: determine whether the state is a goal state
- **Cost function**: a function that assigns a cost to each action

Specifying Search Space

The search space is implicitly represented by a graph, with nodes and edges.

- **Branching factor (b)**: how many actions are available at each state.
- **Depth**: Depth of shallowest goal node **(d)**, maximum depth of any path in the state space **(m)**.



Properties of cost

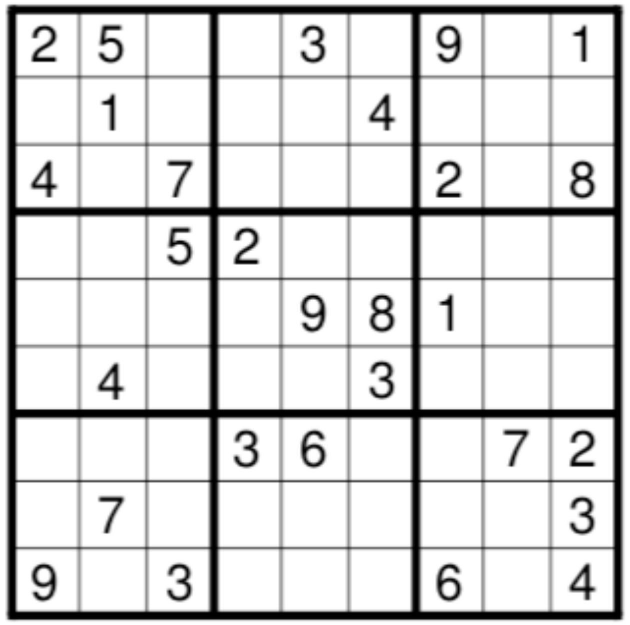
The confusing lecture part - path cost

Usually, $g(n)$ is the current path cost from the initial state to node n (as defined in AIMA)

In lecture, Prof defined:
$g(n)$ as the *optimal* path cost from initial state to n
$\hat{g}(n)$ as the current (minimum) path cost to n
$\hat{g}_{pop}(n)$ as the minimum path to n when we pop

Step cost - $c(s, a, s')$: the cost from s to s' when taking action a.

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

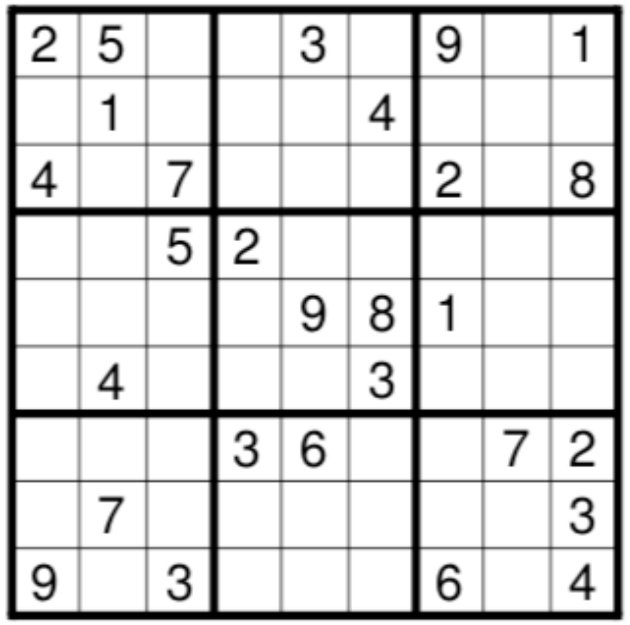
ANSWER:

- **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.

Uninformed Search Algorithms

All the algorithms you learn are in the family of whatever-first search:

```
put s into data_structure
while data_structure is not empty:
    take node from data_structure
    for each neighbour:
        if goal_test:
            return true
        else:
            put neighbour into data_structure
return false
```

This is a tree-based implementation without memory of searched space.
For DFS the data structure is a stack; BFS: queue; UCS: Priority queue

Uninformed Search Algorithms

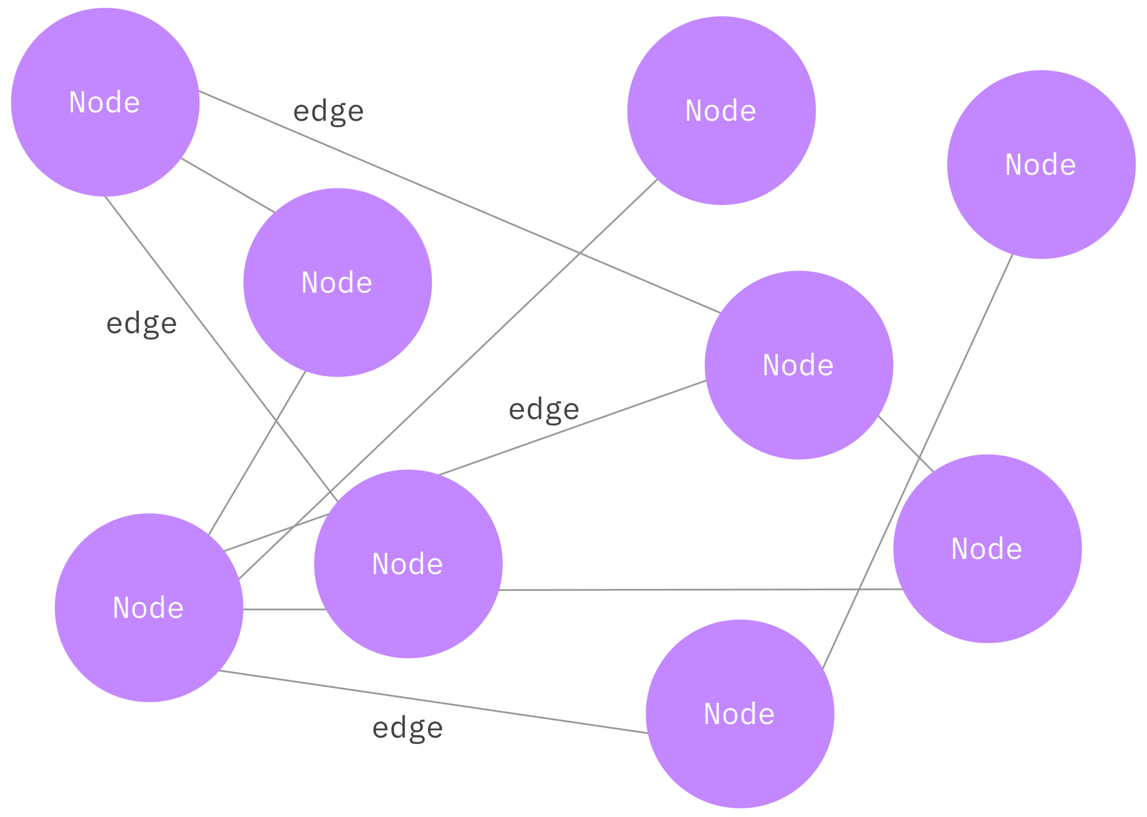
```
put s into data_structure
while data_structure is not empty:
    take node from data_structure
    mark node
    for each neighbour:
        if goal_test:
            return true
        else if neighbour unmarked:
            put neighbour into data_structure
return false
```

This is the graph-based implementation.

The term whatever-first search is taken from Jeff Erickson's Algorithm (and is not a term you should use in exam)

TUTORIAL 2 QUESTION 2 (STATE VS NODE)

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



TUTORIAL 2 QUESTION 2 (STATE VS NODE)

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

ANSWER:

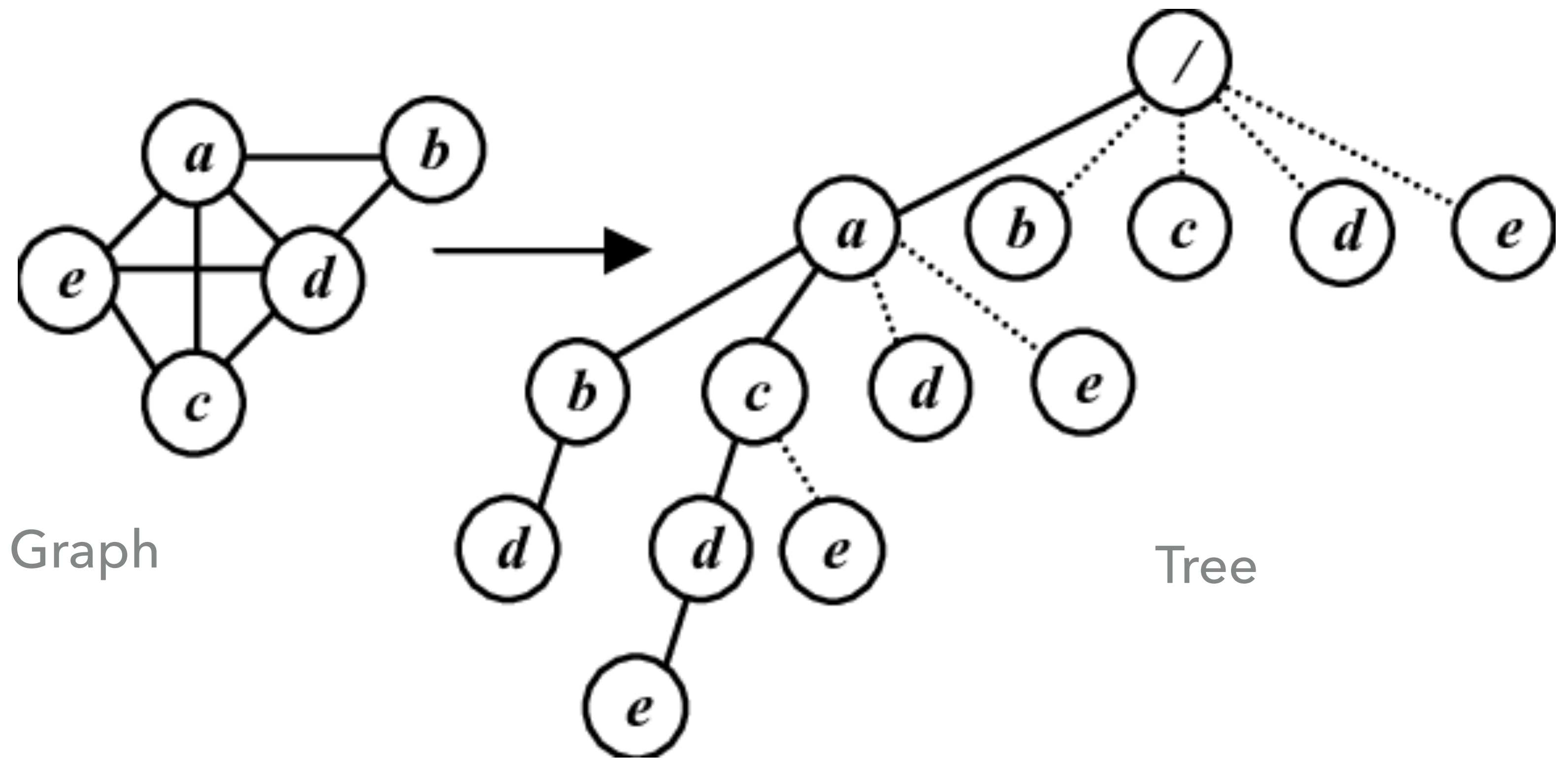
- **State**: a representation of the environment (freeze time and save it - that's a state of the problem)
- **Node**: Data structure to represent a state (we can have multiple nodes referring to the same state)
- Edges from a node to another will then model the transition from a state, taking an action, 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.

TUTORIAL 2 QUESTION 3 (TREE SEARCH VS GRAPH SEARCH)

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

(A tree is a special kind of graph)



TUTORIAL 2 QUESTION 3 (TREE SEARCH VS GRAPH SEARCH)

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

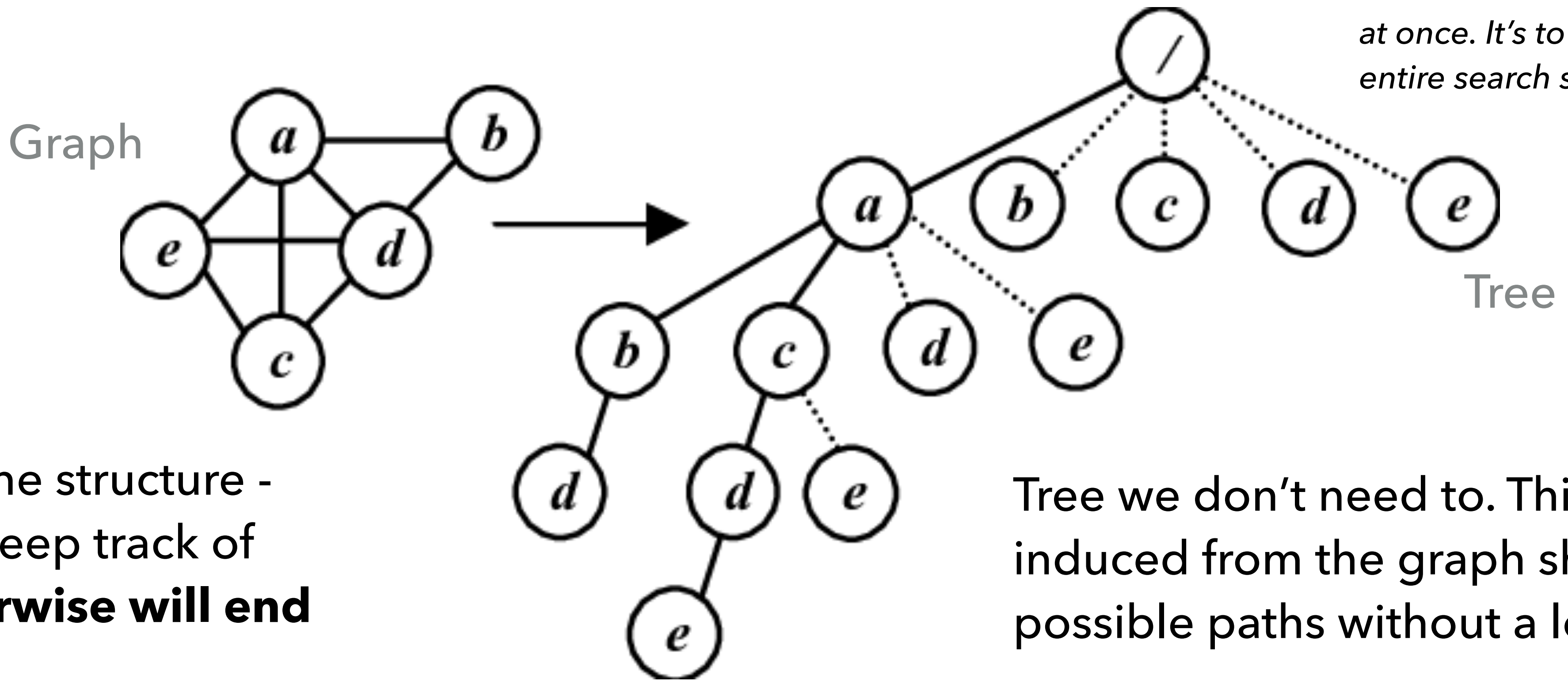
ANSWER:

- 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

TUTORIAL 2 QUESTION 3 (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!

TUTORIAL 2 QUESTION 3 (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

Uninformed Search Algorithms

BFS: Expands shallowest node first. Use when you know solution is near root or ree is deep but solutions are rare.

UCS: Expand the least-path-cost unexpanded node (explore cheaper nodes by current path cost first). Equivalent to BFS if all step costs are equal. It is a special case of Dijkstra's.

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

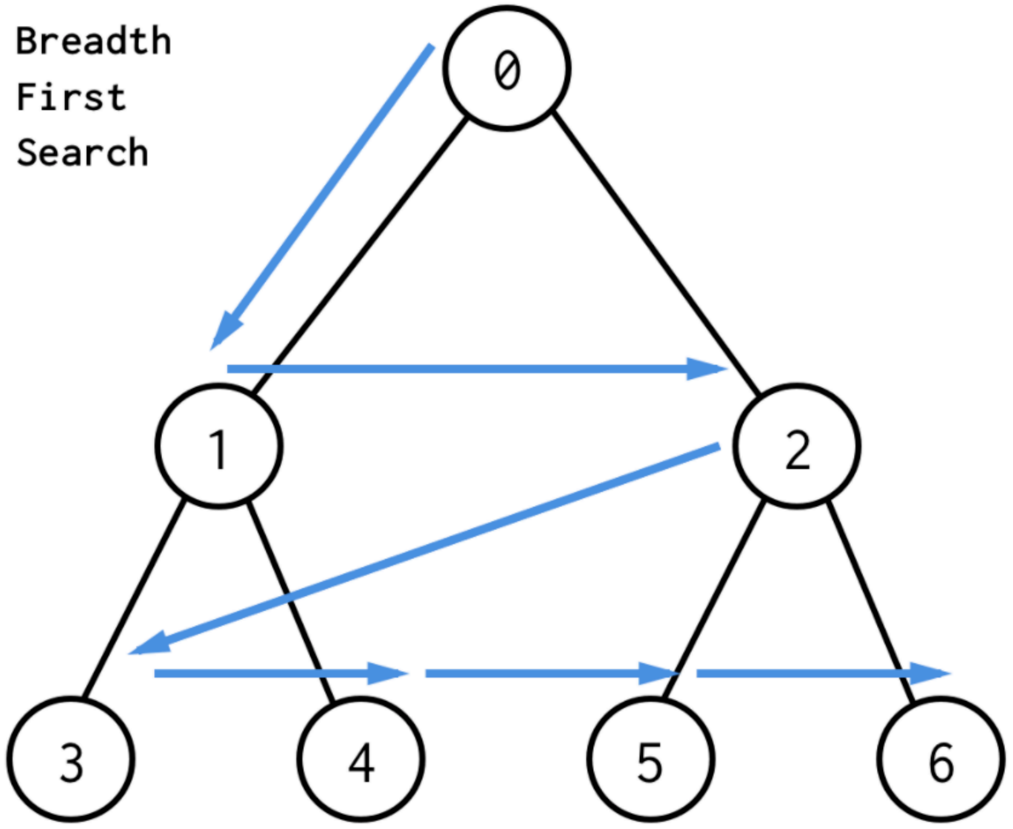
Completeness and Optimality

- **Complete:** if there is a path from start to the goal node, the algorithm will find it.
- **Optimal:** always able to find the least cost solution. Implies completeness.

BFS: Complete if b is finite, optimal if all step costs identical.
UCS: Complete and optimal if b is finite and all step costs $\geq \varepsilon$ for some $\varepsilon > 0$.

TUTORIAL 2 QUESTION 4 (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)?**



TUTORIAL 2 QUESTION 4 (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

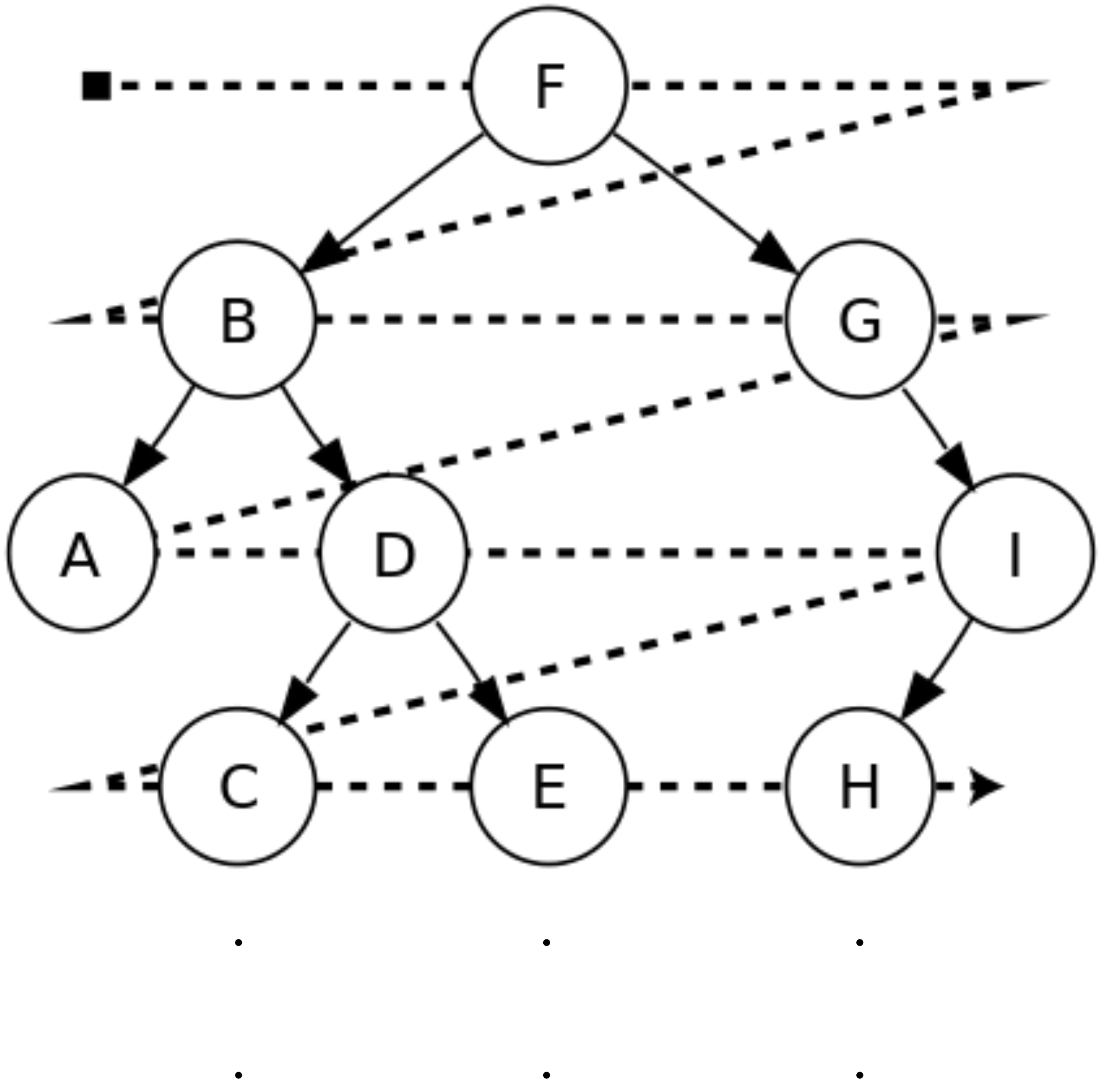
- Save space!
- If I check whether a state is the goal BEFORE pushing to frontier queue, then I can immediately return, no need to expand everything else in the frontier before reaching it.

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**.

TUTORIAL 2 QUESTION 5 (ALSO AY19/20 S2 MIDTERM QUESTION)

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



TUTORIAL 2 QUESTION 5 (ALSO 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.

TUTORIAL 2 QUESTION 6 (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.**

TUTORIAL 2 QUESTION 6 (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).

Question 7 (Appeared in 2019/2020 ST1 exam)

a. S-A-D-B-C-G (unoptimized); S-A (optimized)
b. S-D-A-B-E-C-D-F-G
c. S-D-E-F-G

TUTORIAL 2 QUESTION 8

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

TUTORIAL 2 QUESTION 8

- **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 8

- **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 8

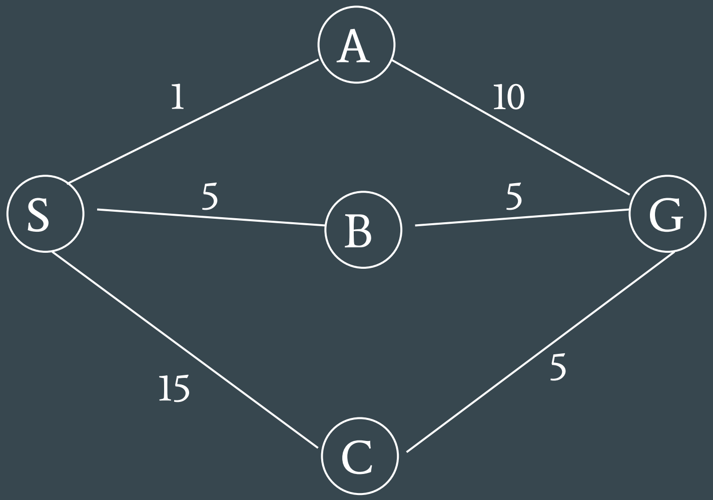
- **Prove that the UNIFORM-COST-SEARCH algorithm is optimal as long as each step cost exceeds some small positive constant ε.**

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.

Q9

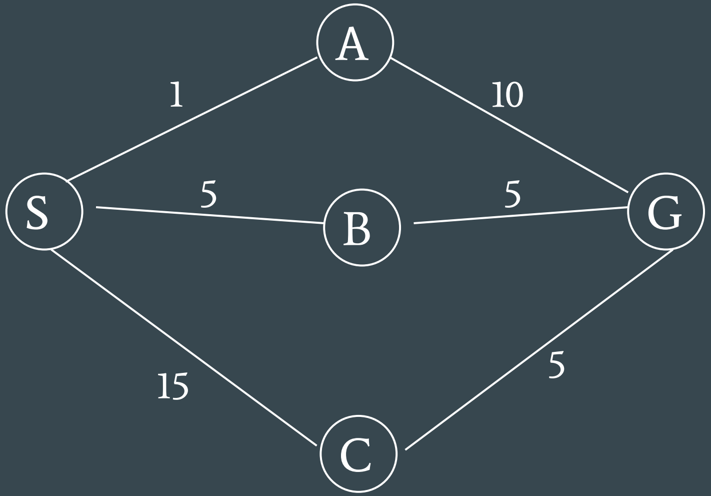
(a) **Trace** the application of the uniform-cost search algorithm on this graph.



Q9

- Implement as graph-search

  (because tree search generates duplicate nodes which may lead to infinite loops)



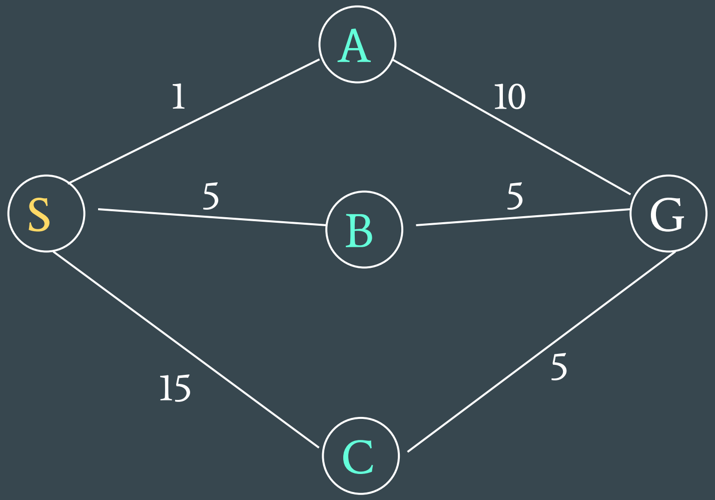
Q9

Trace

Frontier: [S(0)]

Explored: [ ]

- Explore S, add A(1), B(5), C(15) to frontier



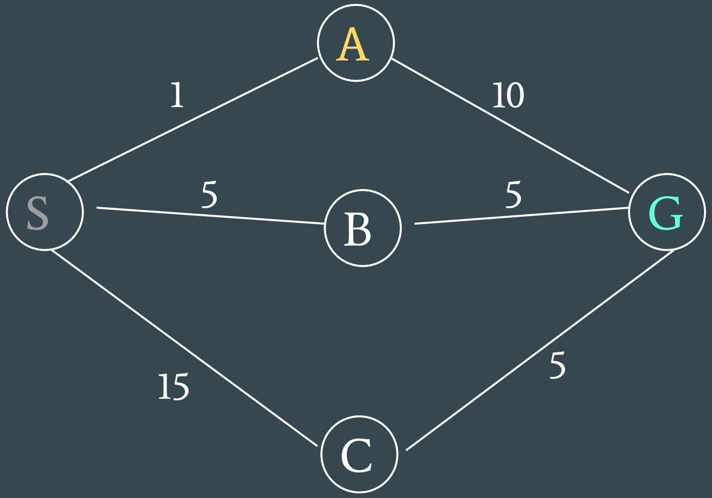
Q9

Trace

Frontier: [A(1), B(5), C(15)]

Explored: [S(0)]

- Explore A(1), add G(11) to frontier



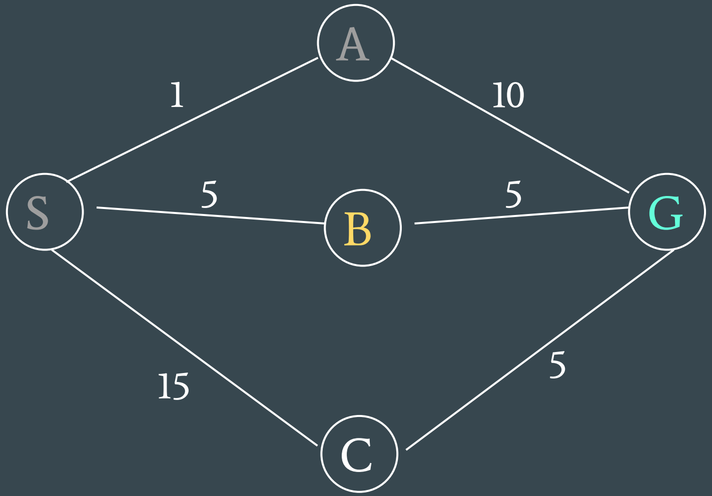
Q9

Trace

Frontier: [B(5), G(11), C(15)]

Explored: [S(0), A(1)]

- Explore B(5), replace G(11) by G(10) in frontier



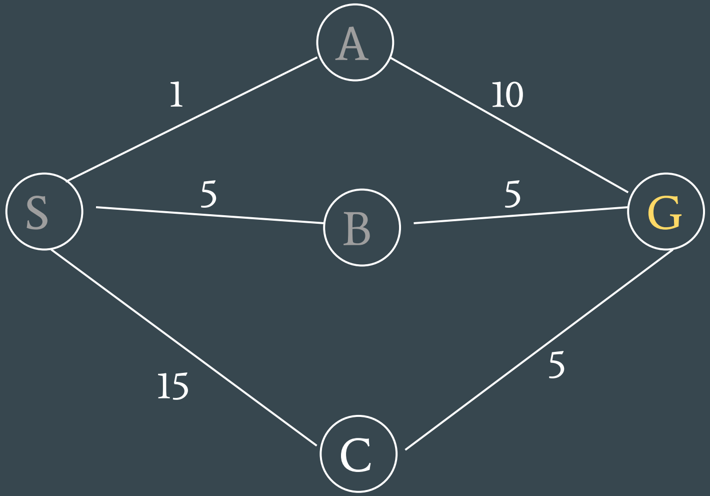
Q9

Trace

Frontier: [G(10), C(15)]

Explored: [S(0), A(1), B(5)]

- Explore G(10), goal test succeeds



Q9

(b) When A generates G which is the goal with a path cost of 11, why doesn’t the algorithm halt and return the search result since the goal has been found? With your observation, discuss how uniform-cost search ensures that the shortest path solution is selected.

Q9

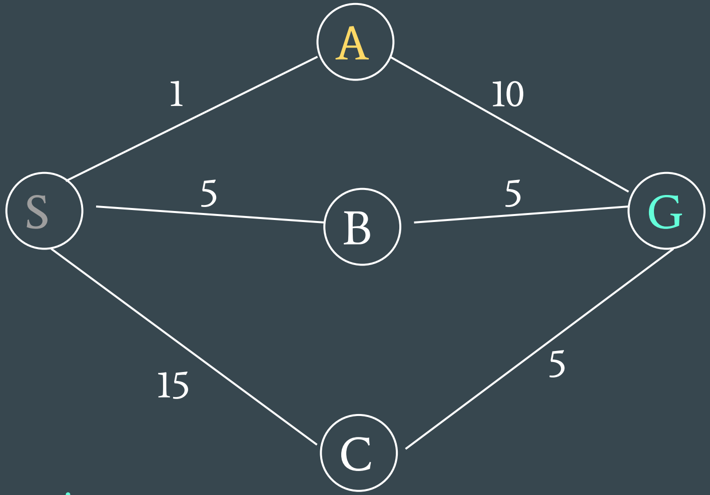
After G is generated, it will be sorted together with the other nodes in frontier.

Note that goal test in UCS is done on exploration, not generation!

Resulting frontier: [B(5), G(11), C(15)]

Since node B has lower path cost than G, it is then selected for expansion.

By always selecting the node with the least path cost for expansion (i.e., nodes are expanded in increasing order of path cost), UCS ensures that the first goal node selected for expansion is an optimal solution.



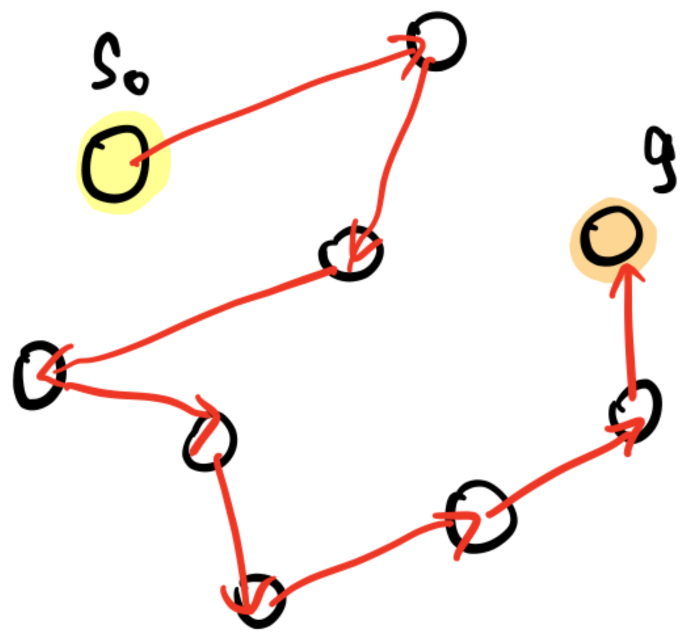
TUTORIAL 2 QUESTION 10

- **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.

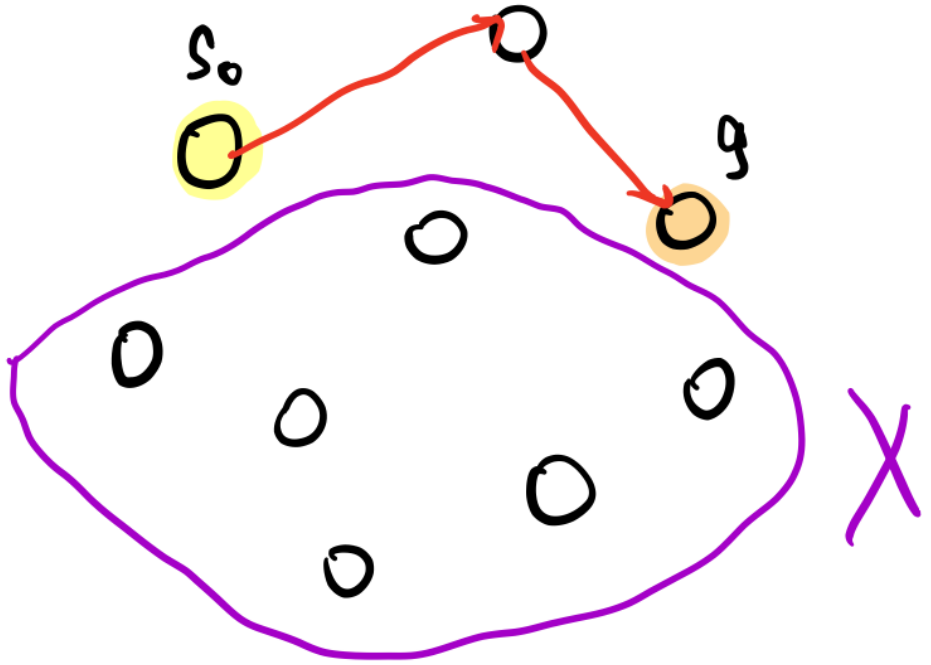
TUTORIAL 2 QUESTION 10

- 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!



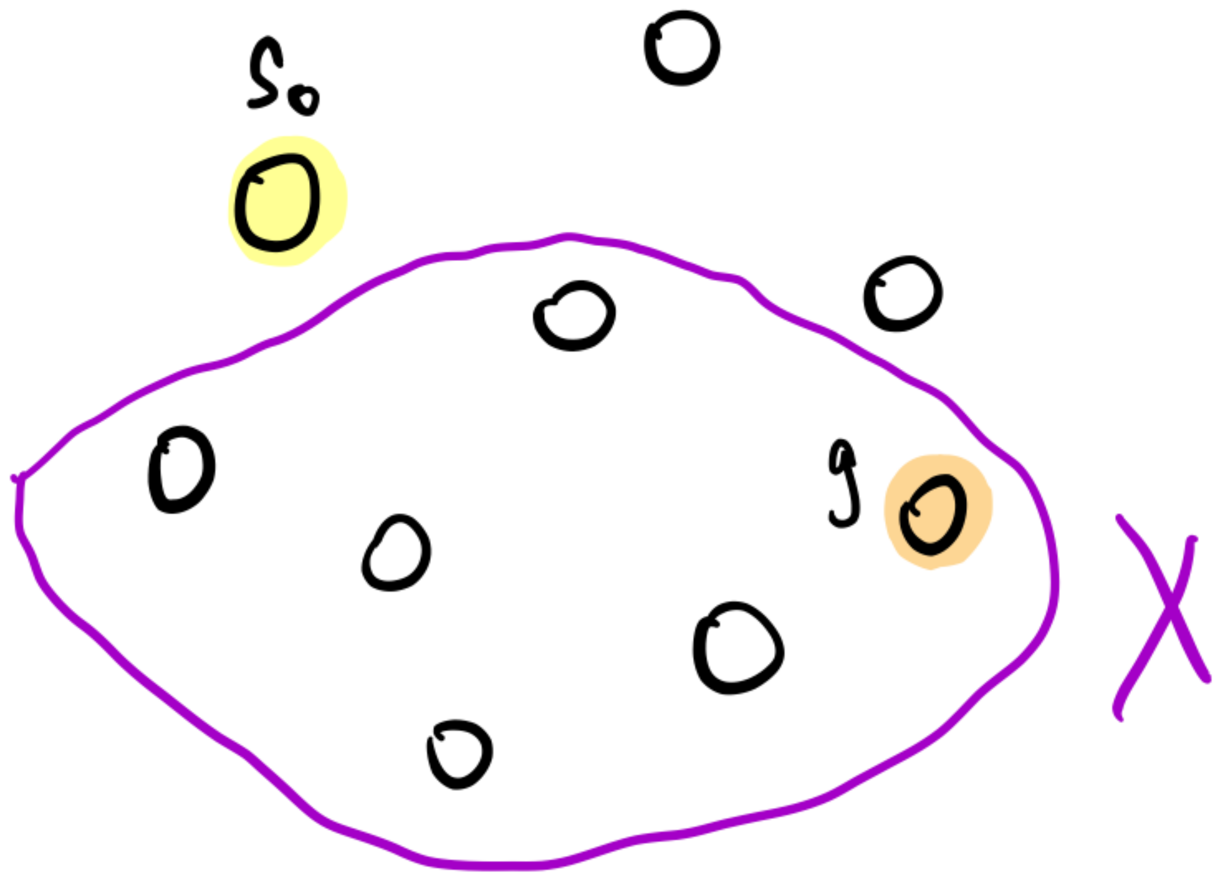
TUTORIAL 2 QUESTION 10

- 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 10

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

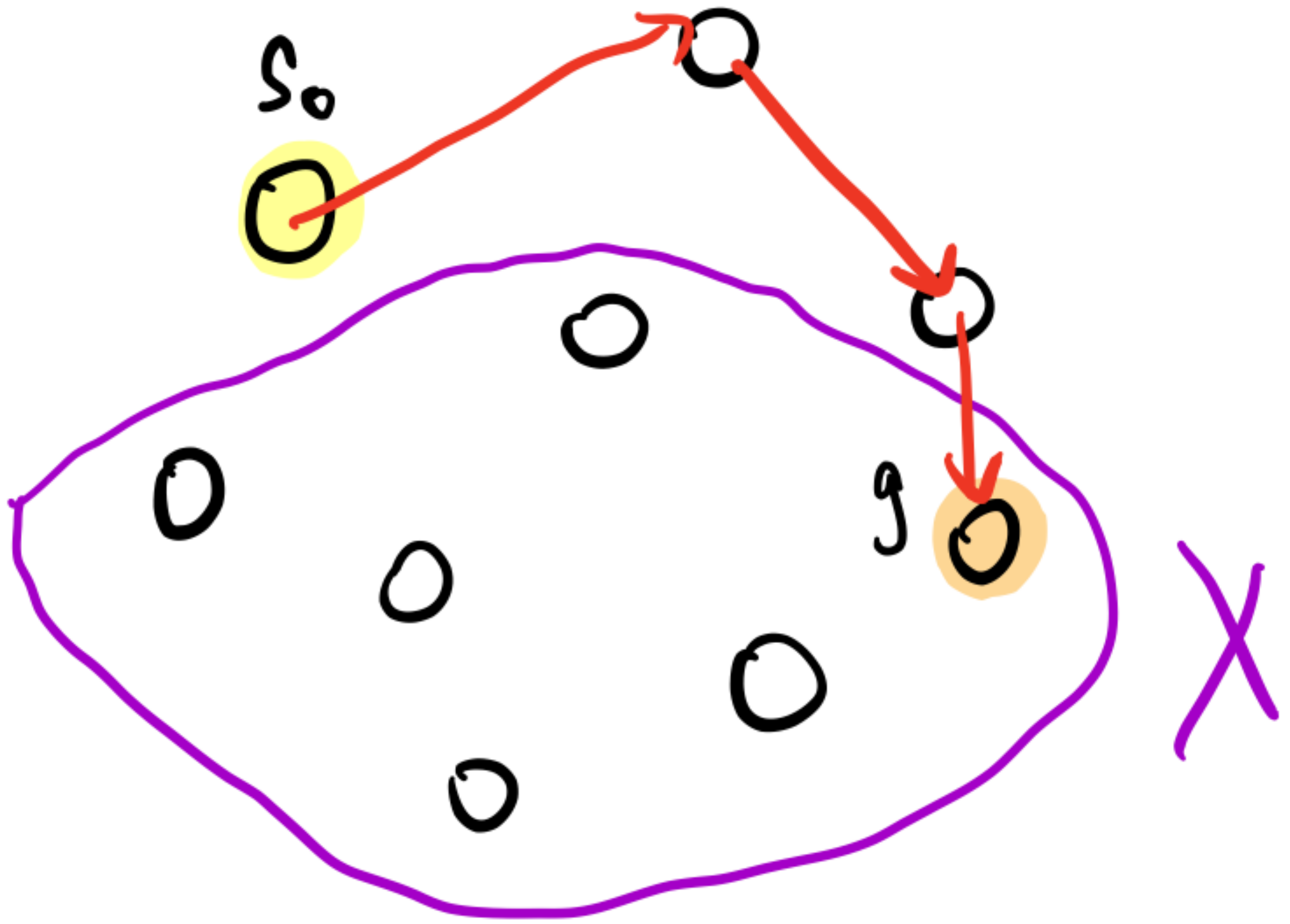


TUTORIAL 2 QUESTION 10

- 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 <u>worst case</u>, I don't "clear" any other nodes in X, just that new goal node.

- Redefine X.



TUTORIAL 2 QUESTION 10

- 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
- Know the 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 :)