

Abstract geometric lines in the top-left corner of the slide, consisting of several thin black lines forming various polygons and intersecting at different points.

ARTIFICIAL INTELLIGENCE

Blind & Guided Search Algorithms



DISCLAIMER

I have tried to pull all information from the slides, the textbook, and other authorized resources, but I cannot guarantee the veracity of any information in the slides hereafter.

See presentation notes for sources (book pages are for 4th ed.)

ALGORITHM CLASSIFICATION

1. Search Algorithms

1. Blind/uninformed search
2. Guided/informed/
heuristic search



SEARCH ALGORITHM TERMINOLOGY

Because it can never just be simple.

WHAT IS YOUR PROBLEM?

State Space (Typically a lot)

State
`{Floor3Main}`

Initial State
`{Floor1Main}`

Goal State(s)
`{Room303}`

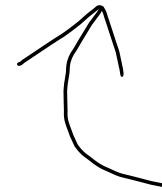


Can be >1

`{Room204,
Room217, Room302,
Room303, Room310,
Room311, Room402,
Room410, Floor1-
2Stairway,
Floor2-3Stairway,
Floor3-4Stairway,
Elevator1,
Elevator2,
Floor1Main,
Floor2Main,
Floor3Main,
Floor4Main,
Floor1MensBathroo
m,
Floor2WomensBathr
oom, ...}`

Actions

`ACTIONS(Floor3Main)
= {GoToRestrooms,
GoToTables,
GoToVendingMachines,
GoToRoom302,
GoToRoom303,
Room310, Room311,
GetLost, Cry}`



Applicable in s

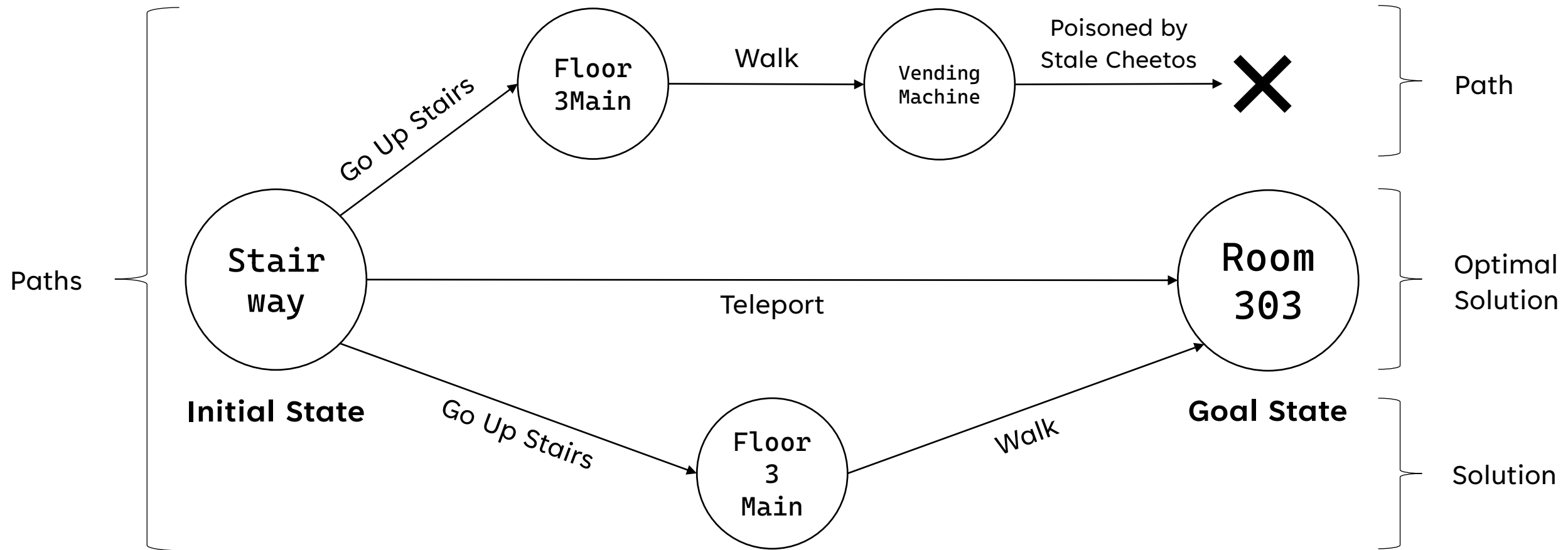
Transition Model

`RESULT(Floor3Main,
GoToRoom303) = Room303`

Action Cost Function

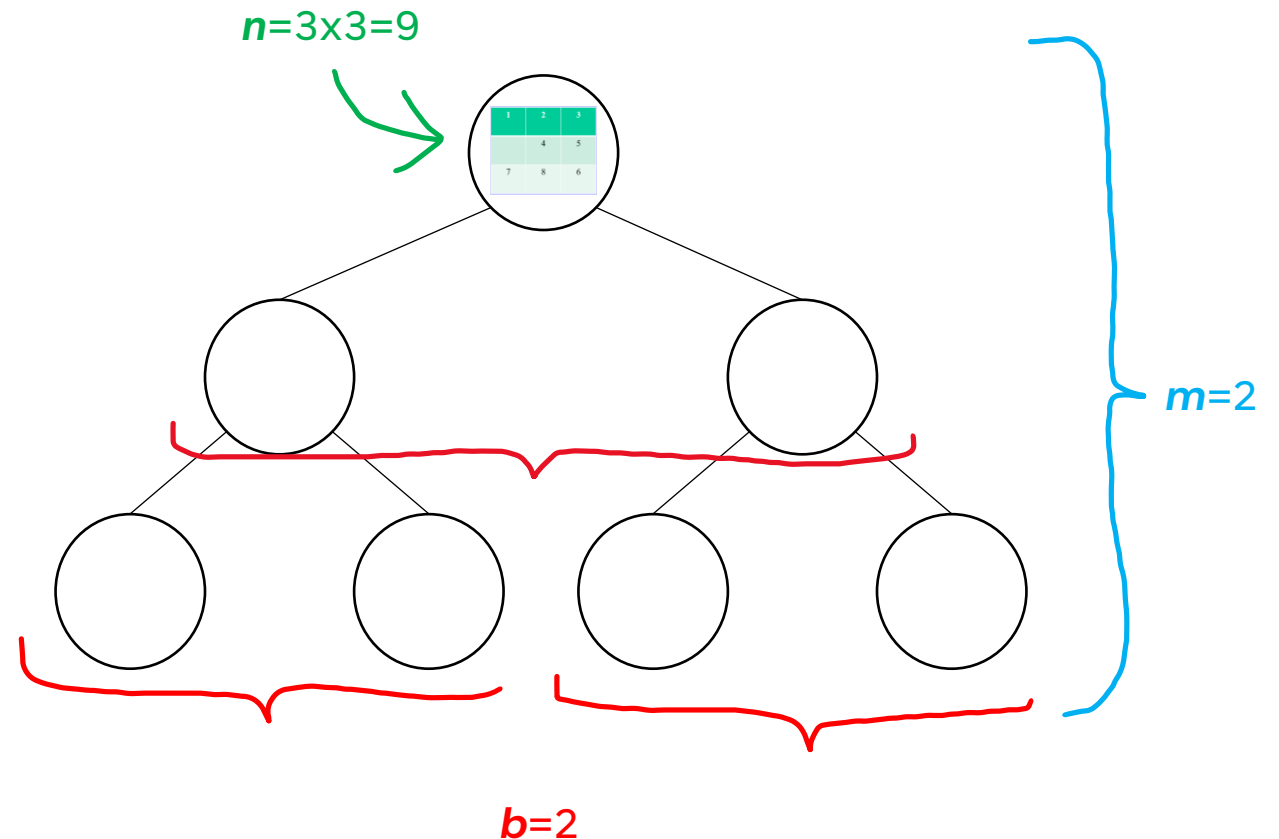
`ACTION-COST(Floor3Main,
GoToRoom303, Room303) =
Walking`

SHORTEST PATH (LAWS OF PHYSICS OPTIONAL)



TIME COMPLEXITY CALCULATION VARIABLES (THIS BAD BOY CAN FIT SO MANY BRANCHES IN IT)

- **Branching Factor (b)** – The average number of **actions applicable** for a given **state**.
- **Input Size (n)** – Typically, the number of variables within a given **state**. For instance, an 8-puzzle uses a 3x3 grid, $n = 9$.
- **Max Depth (m)** – The maximum number of **actions** that can be taken on a given **path** (depth of search tree).



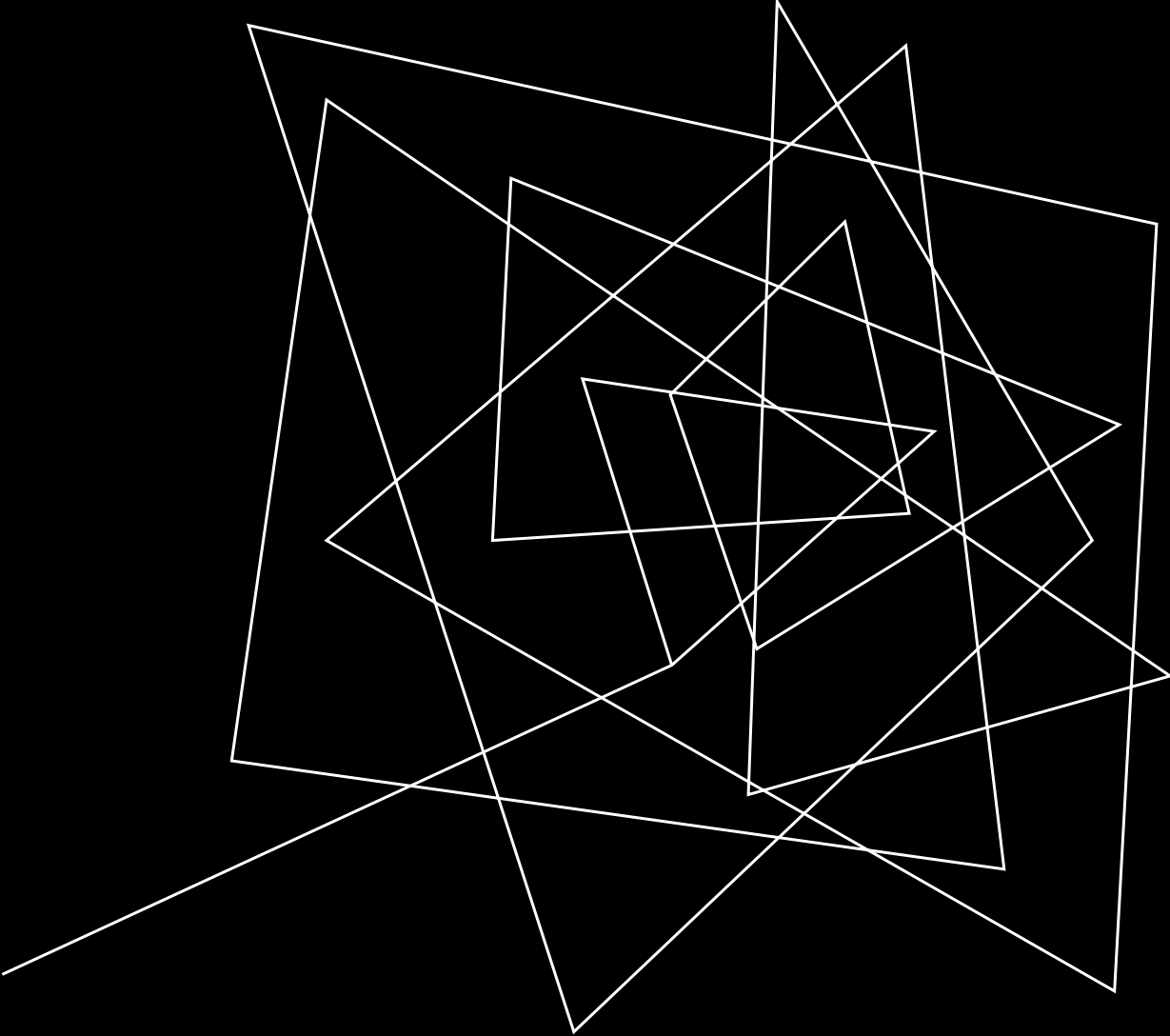
MEASURING PERFORMANCE

(SURE, IT'S $O(\infty)$ SPACE COMPLEXITY, BUT IT'S $O(1)$ EXPECTED TIME COMPLEXITY)

- **Completeness** – Is the algorithm guaranteed to find a solution if there is one, and to correctly report failure when there is not?
- **(Cost) Optimality** – Does it find a solution with the lowest path cost of all solutions?
- **Time Complexity** – How long does it take to find a solution? Typically measured with $O(n)$, with b , n , m , and d (depth of goal).
- **Space Complexity** – How much memory is needed to perform the search?

GRAPH SEARCH VS. TREE SEARCH (WHAT FOOT WOULD YOU LIKE TO SHOOT?)

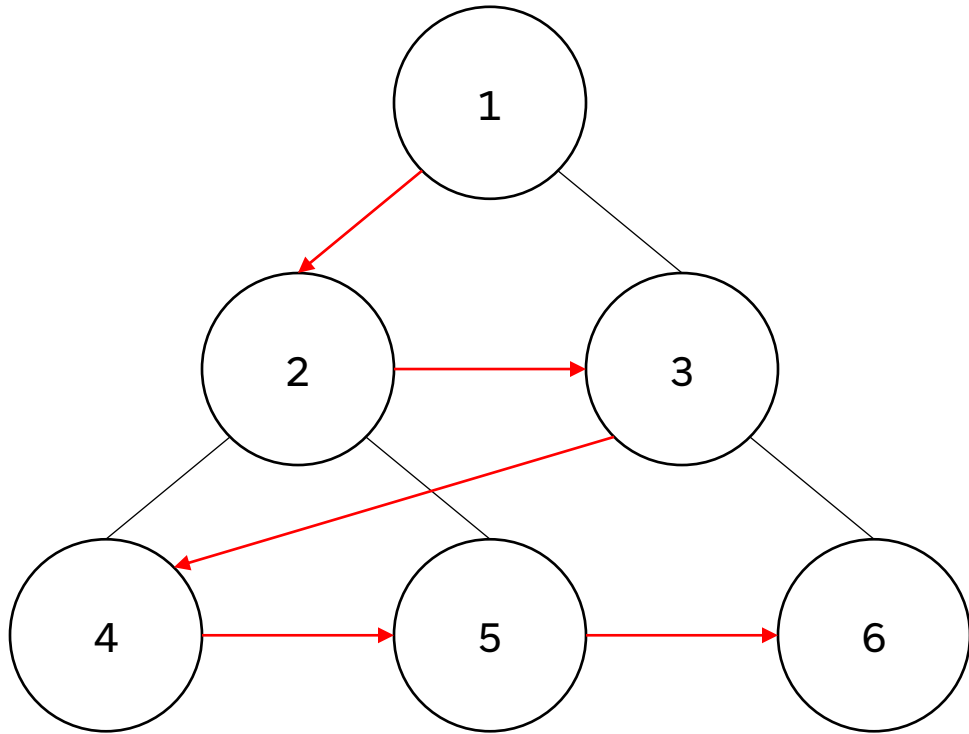
- Finally, we need to make an important distinction between **graph search** and **tree search**.
- A graph search:
 - Remembers visited **states** and skips redundant paths
 - High memory, low time
- A tree search:
 - Does not remember visited **states** and goes down redundant paths
 - Low memory, high time



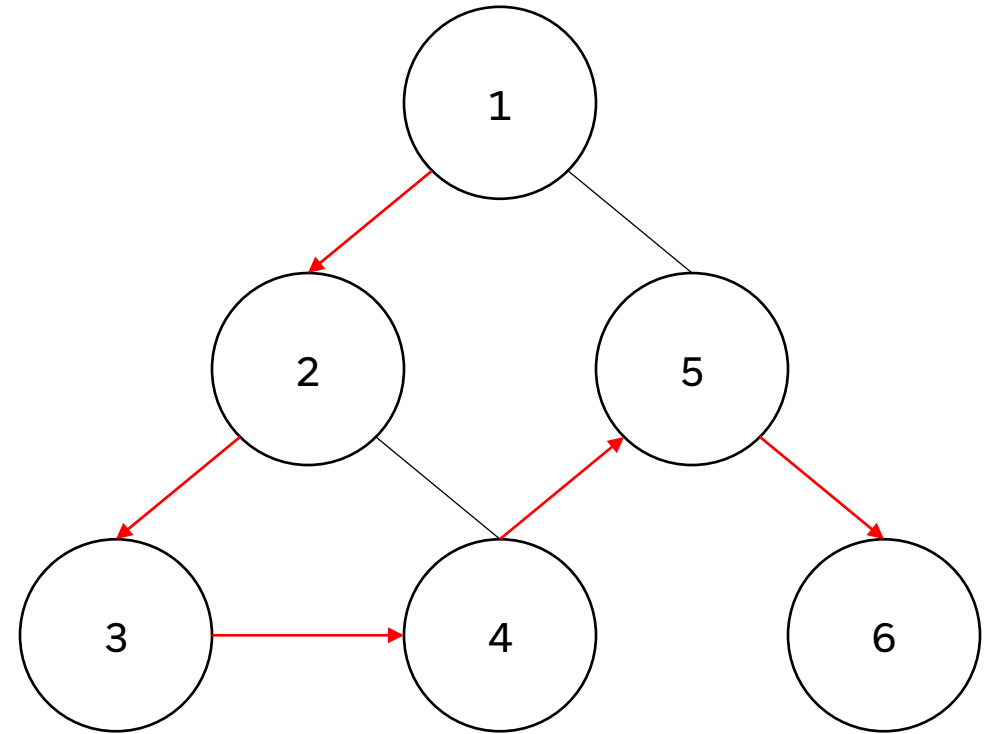
BLIND SEARCH ALGORITHMS

BFS/DFS/DLS/IDS

BREADTH-FIRST SEARCH



DEPTH-FIRST SEARCH



POP QUIZ!

**What metrics will we use to
compare Breadth-First Search
with Depth-First Search?**

(Hint: There's 4)

BFS VS. DFS COMPARISON

	Breadth-First Search	Depth-First Search
Completeness	Yes, if goal exists, BFS will find it!	Yes, if the state space is finite and has no cycles.
(Cost) Optimality	Yes, if all actions have the same cost.	No, returns first solution found.
Time Complexity	$O(b^d)$, where d is the depth of the goal.	$O(b^d)$, where d is the depth of the goal.
Space Complexity	$O(b^d)$, where d is the depth of the goal (remembers all nodes).	$O(m)$, where m is the max depth of the search tree (remembers one set of children per level).
Type of Search	Graph Search	Tree Search
Data Structure	Queue	Stack



UPGRADES, PEOPLE, UPGRADES

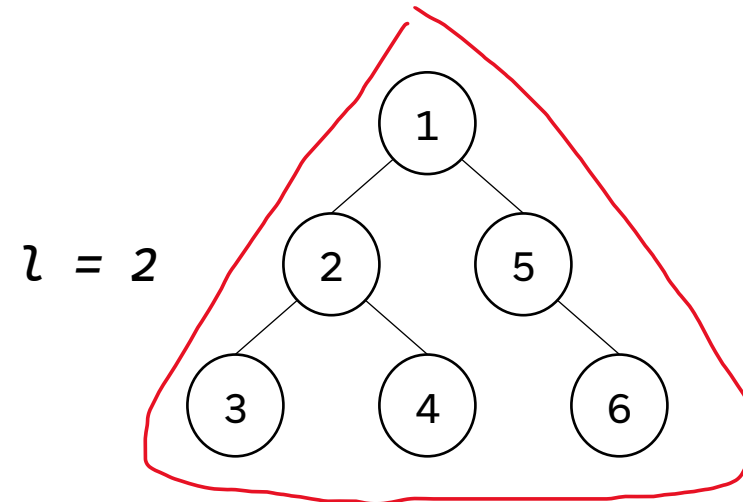
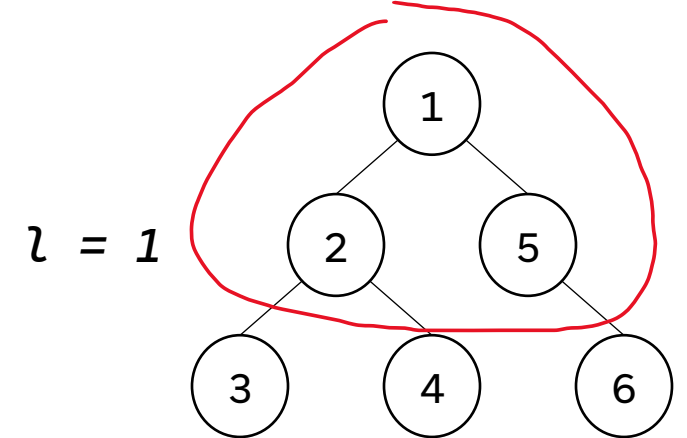
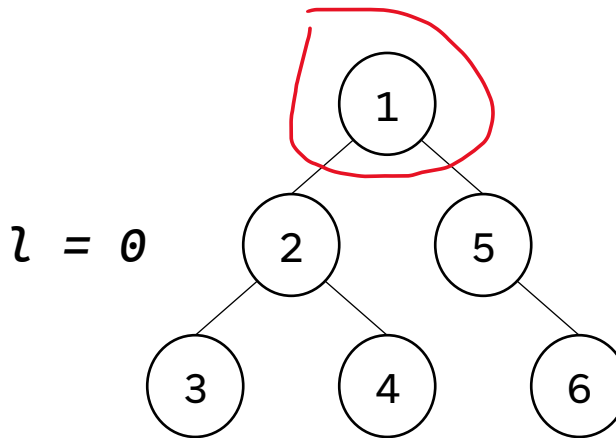
- Remember that in computation, **memory is a larger issue than time.**
 - This is because memory complexity is bound by time complexity.
 - This makes memory a bottleneck.
- Therefore, DFS is preferred (in general).
- However, DFS can get stuck at infinite (read: large) depth.
- So, let's limit the range of DFS!

DEPTH LIMITED SEARCH (DLS) (AN ALGORITHM'S GUIDE TO GIVING UP)

- Stop DFS at **fixed** length l , no matter what.
- **Incomplete** algorithm (if $l < d$, where d is the depth of the goal).
- Updated complexities:
 - Time: $O(b^l)$
 - Memory: $O(l)$

ITERATIVE DEEPENING SEARCH (BUT WAIT, THERE'S MORE!)

- Iterative executions of DLS, where l is incremented from $[0, cutoff]$.
- Complete, because goal will be found when $l = d$
- Cost complexity is $O(b^{d+1})$
 - The last layer (where goal is found) has b^d nodes and is explored once. The second to last layer has b^{d-1} nodes and is explored twice, continue...
 - $N(IDS) = (d)b^1 + (d-1)b^2 + \dots + (2)b^{d-1} + (1)b^d$
- Memory complexity still $O(d)$



SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

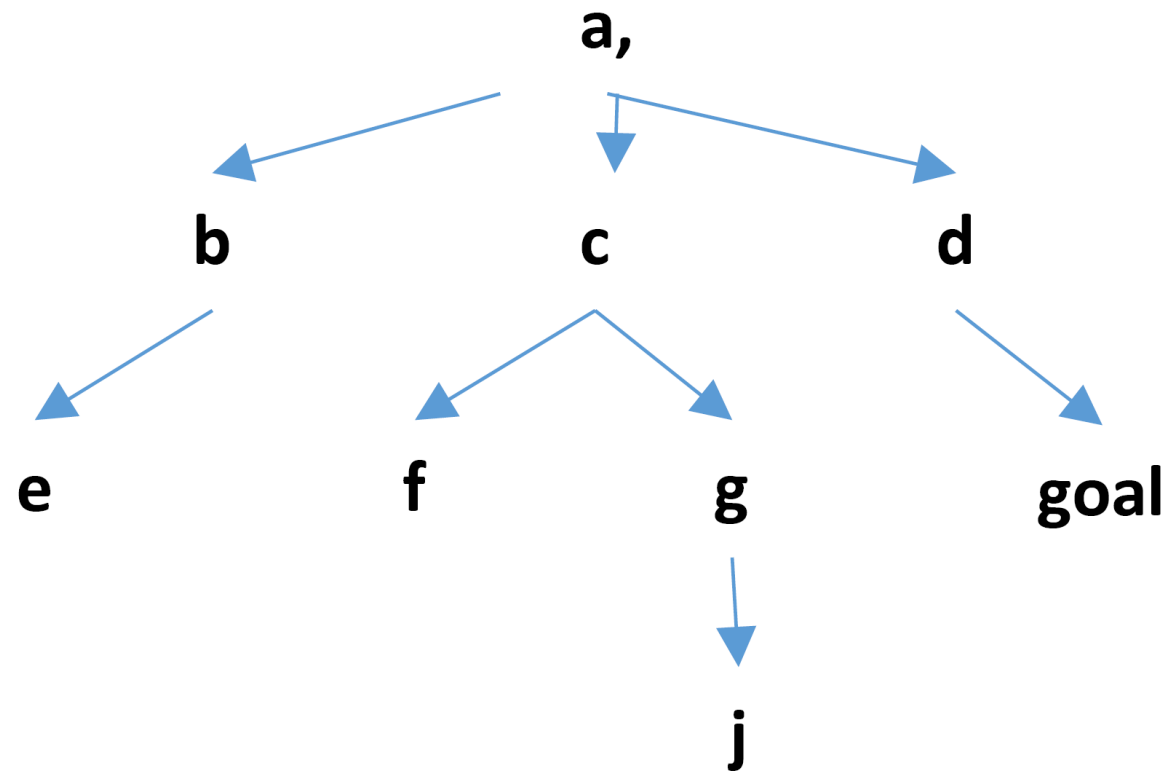
Variables

$l = 0$

visited = []

Stack = []

IN MOST
IMPLEMENTATIONS, A
WOULD BE VISITED
HERE, BUT THAT IS NOT
THE CASE IN THIS
IMPLEMENTATION



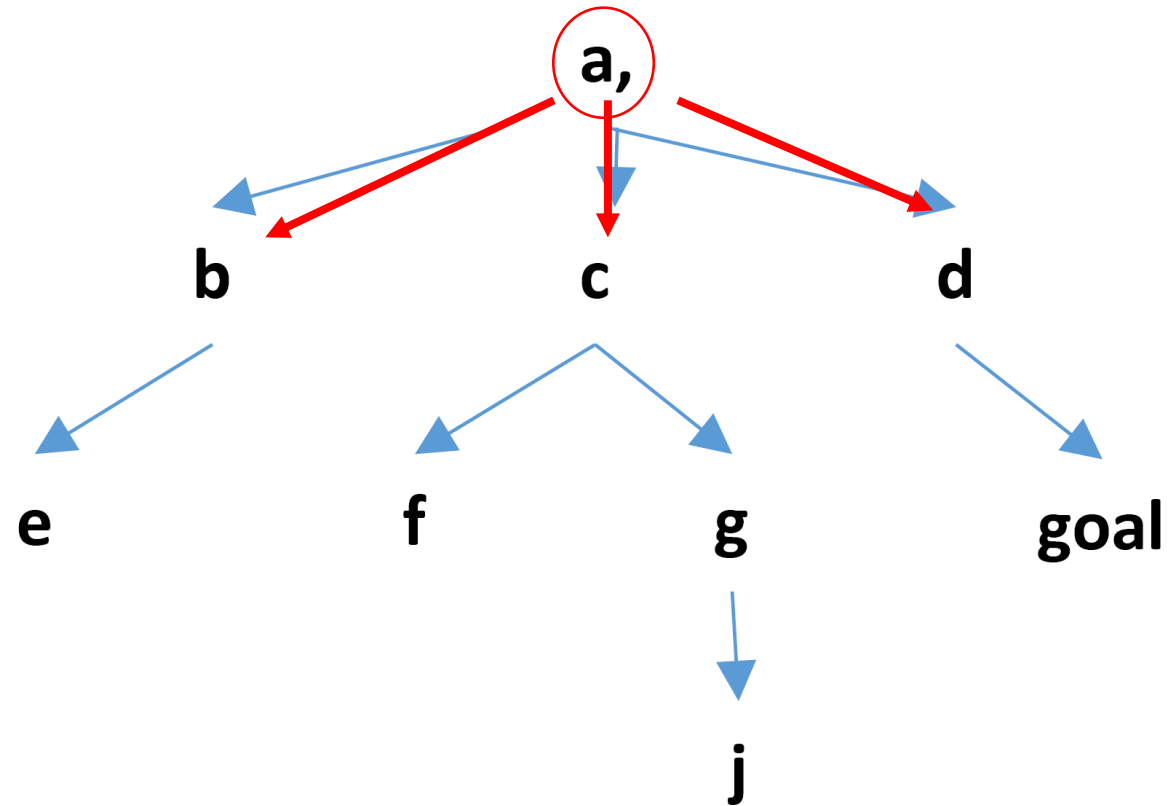
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 1$

visited = [a]

Stack = [b, c, d]



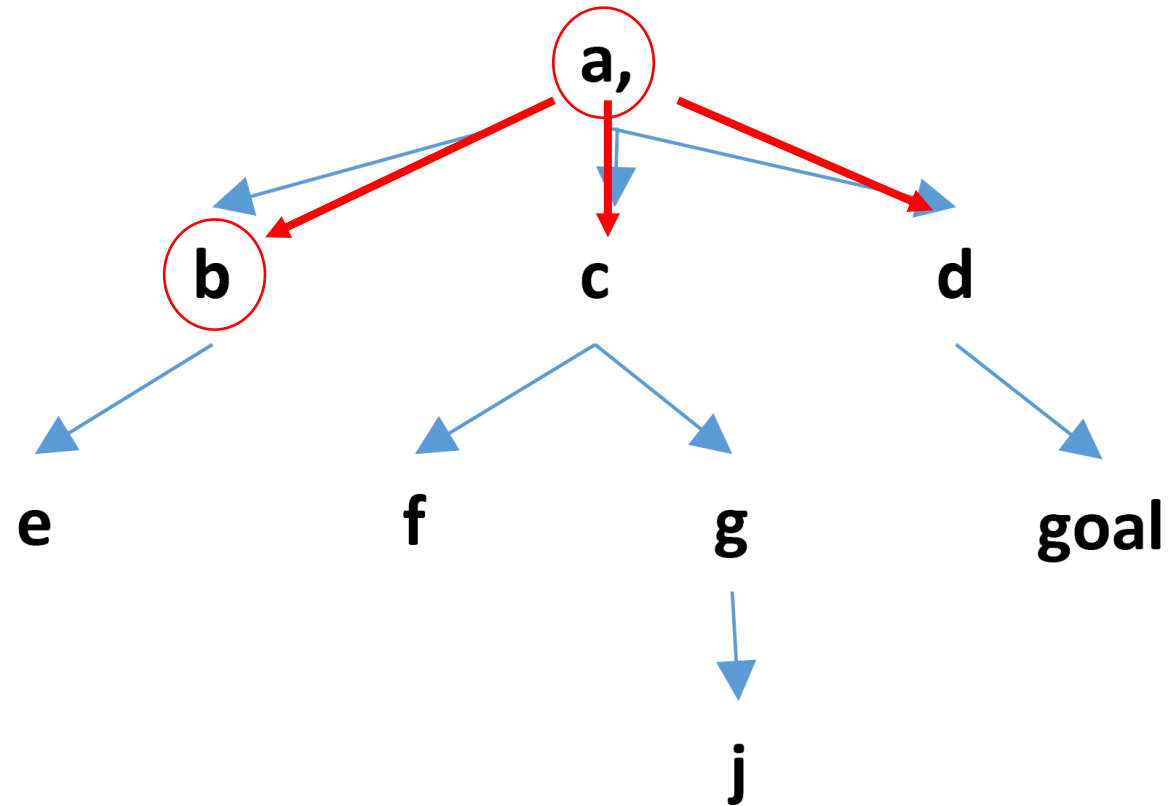
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 1$

visited = [a, b]

Stack = [c, d]



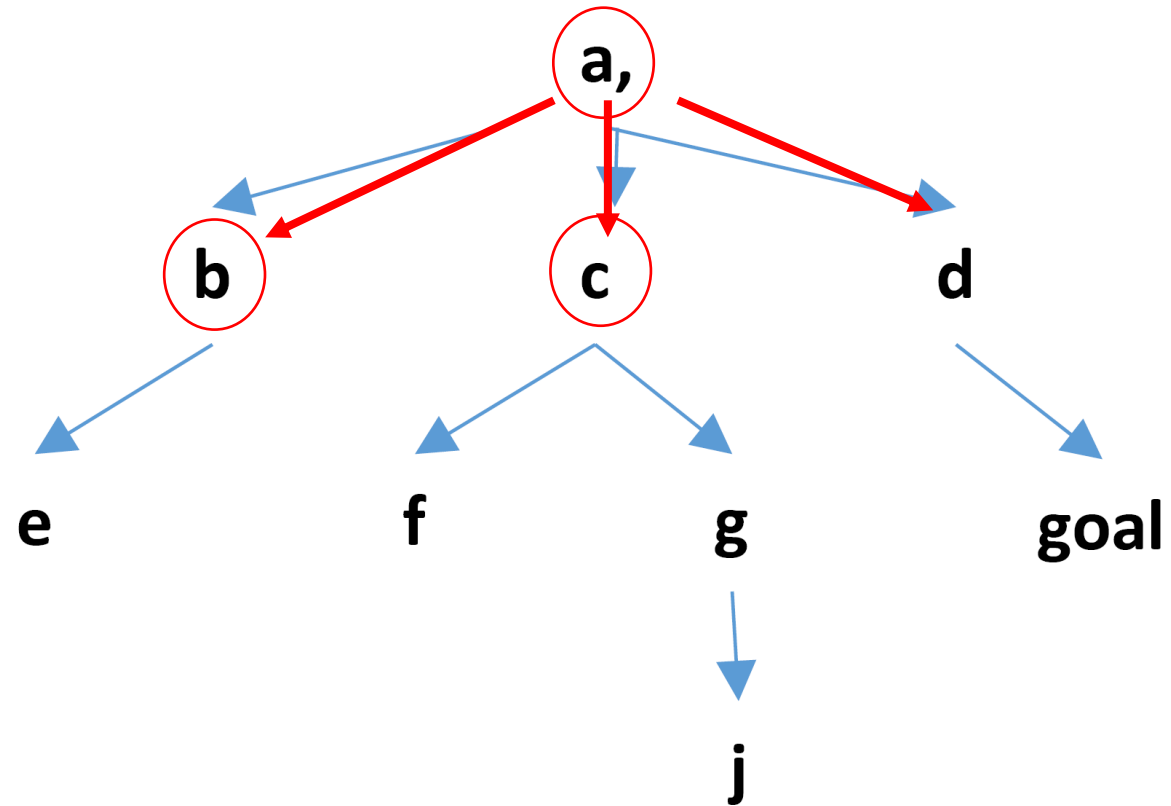
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 1$

visited = [a, b, c]

Stack = [d]



SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

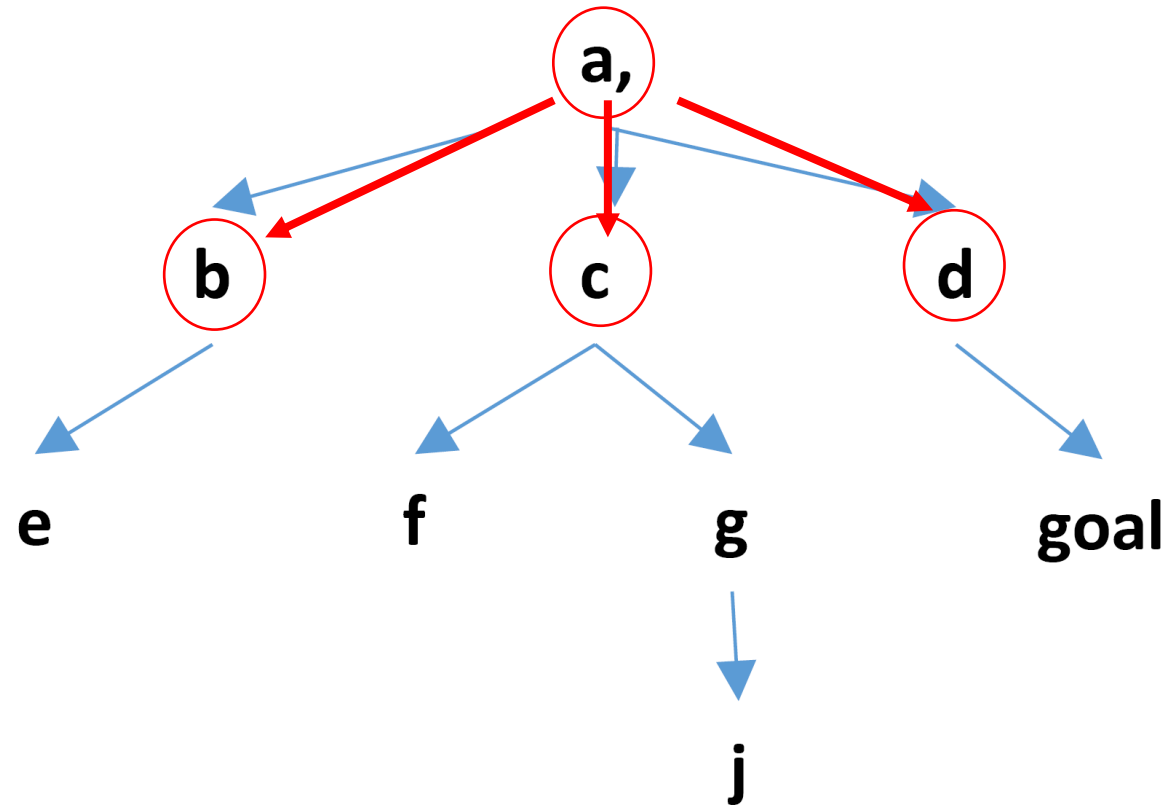
Variables

$l = 1$

visited = [a, b, c,
d]

Stack = []

MAXIMUM DEPTH REACHED



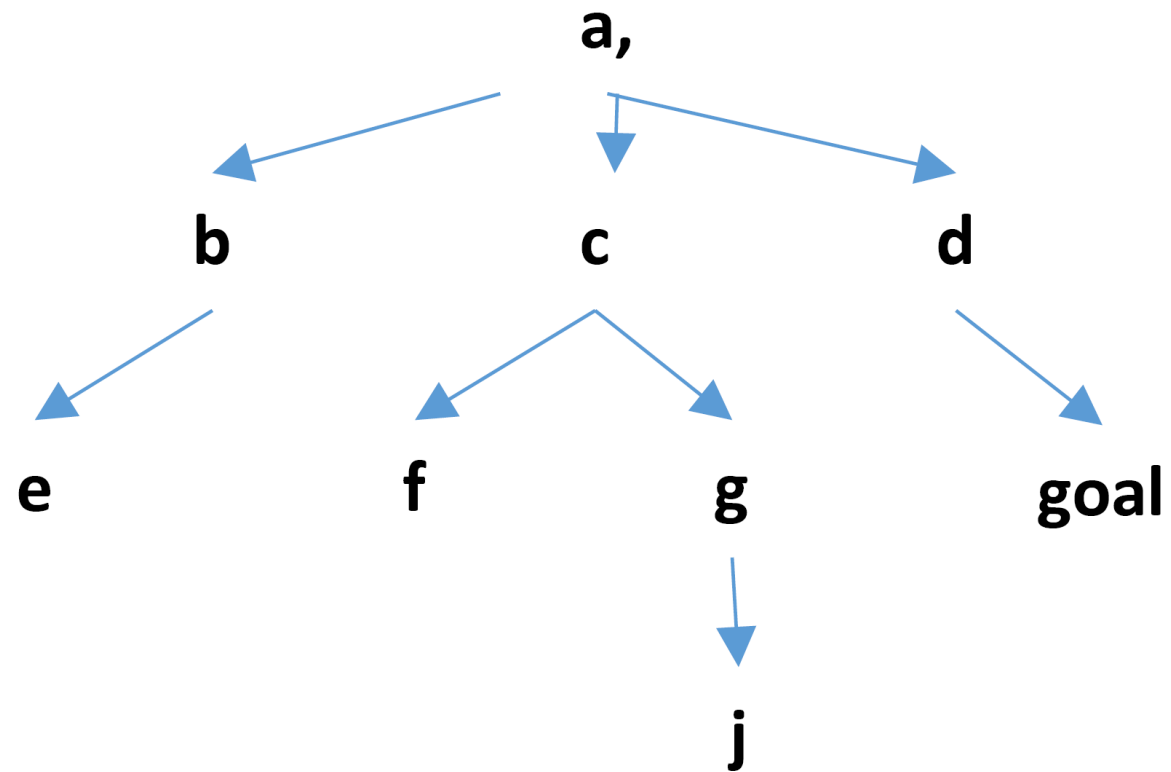
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d]

Stack = [a]



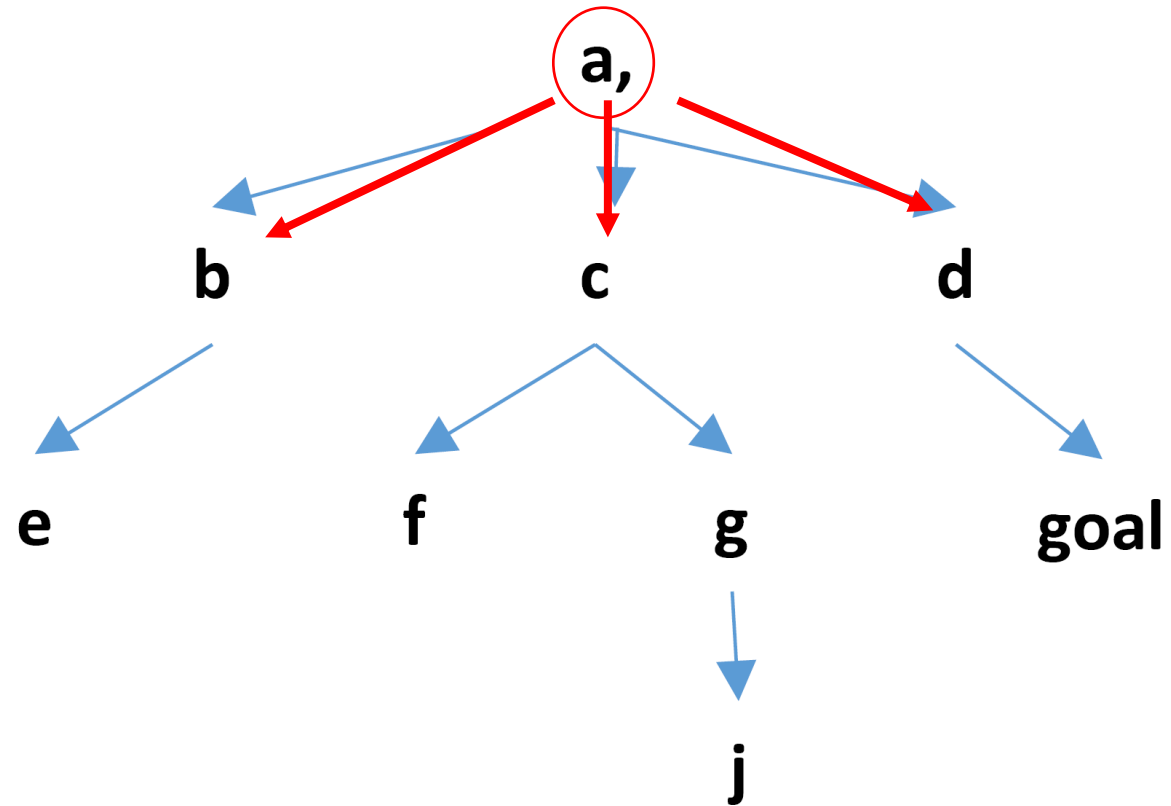
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a]

Stack = [b, c, d]



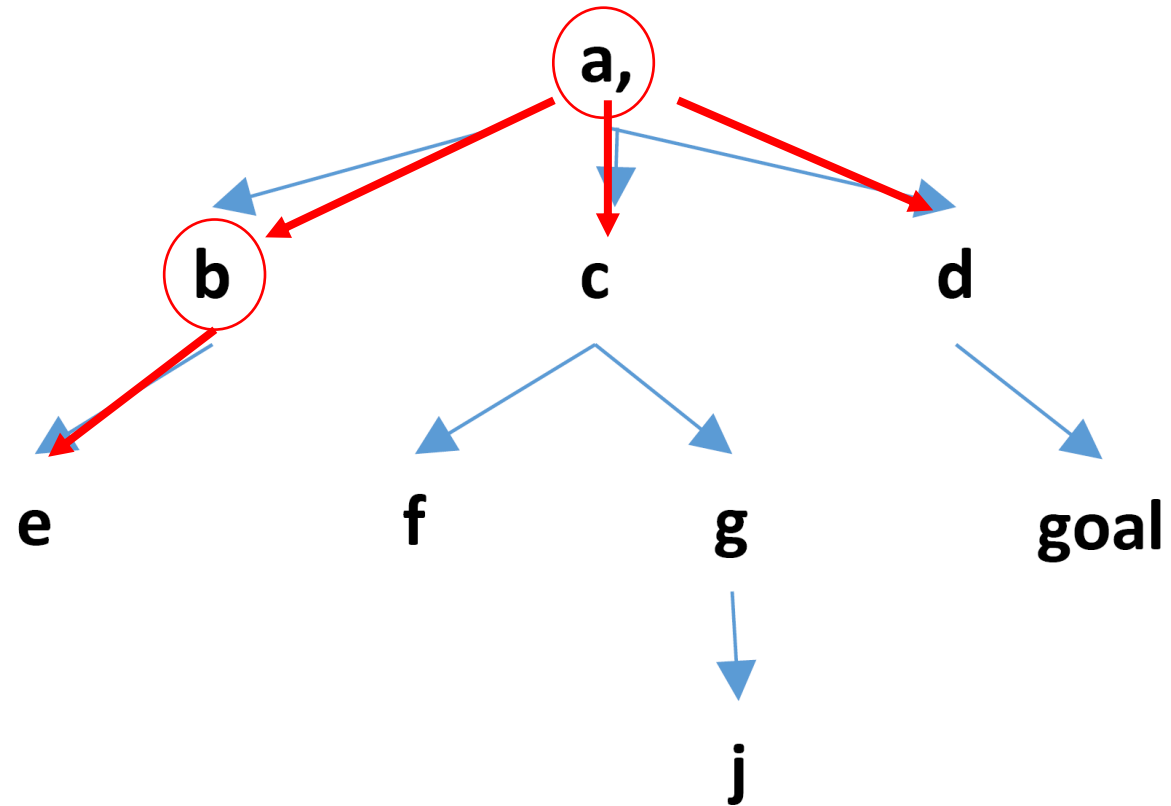
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b]

Stack = [e, c, d]



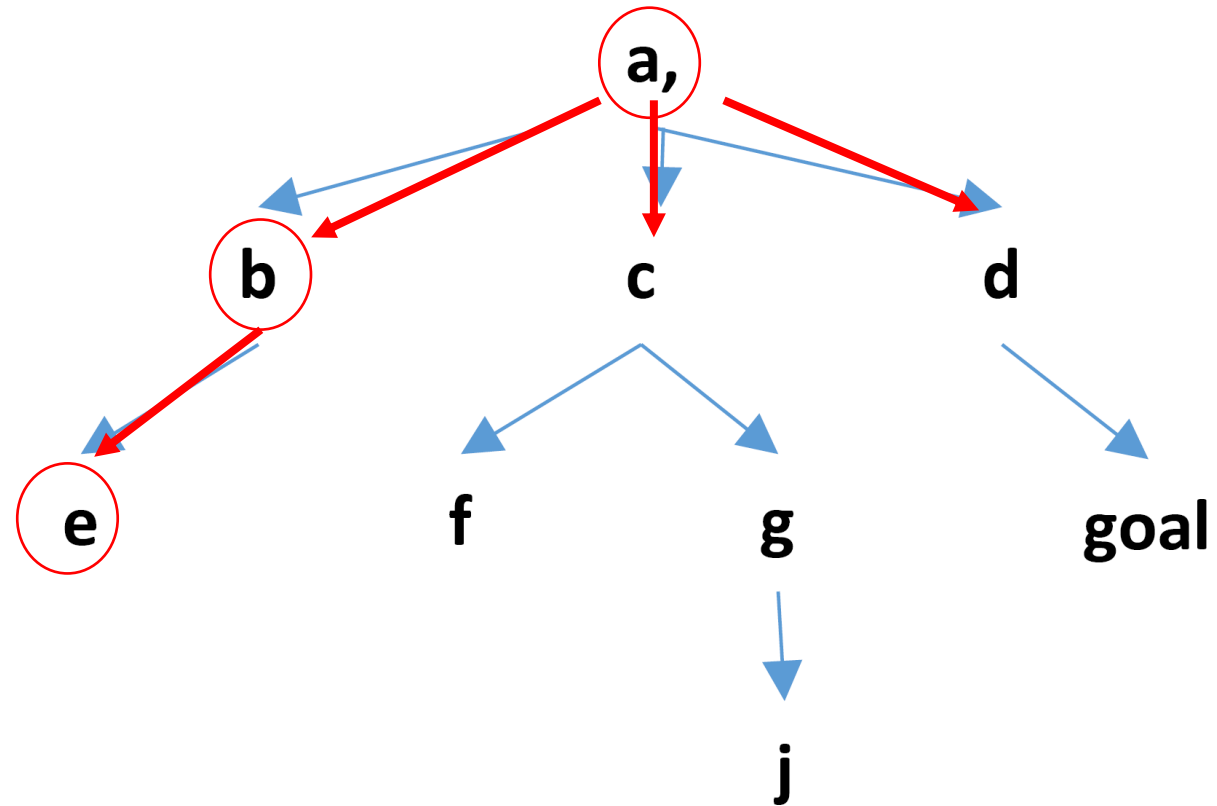
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b, e]

Stack = [c, d]



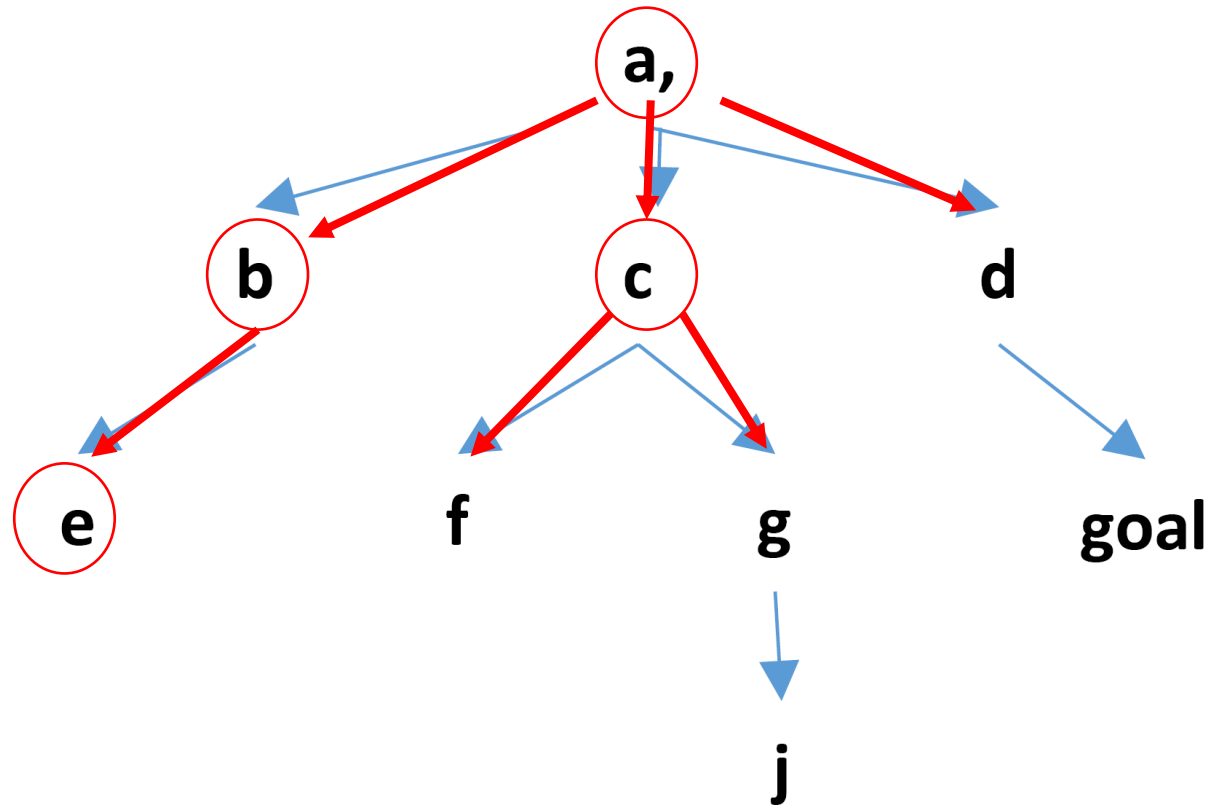
SEARCH QUIZ QUESTION 1 (TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b, e, c]

Stack = [f, g, d]



SEARCH QUIZ QUESTION 1

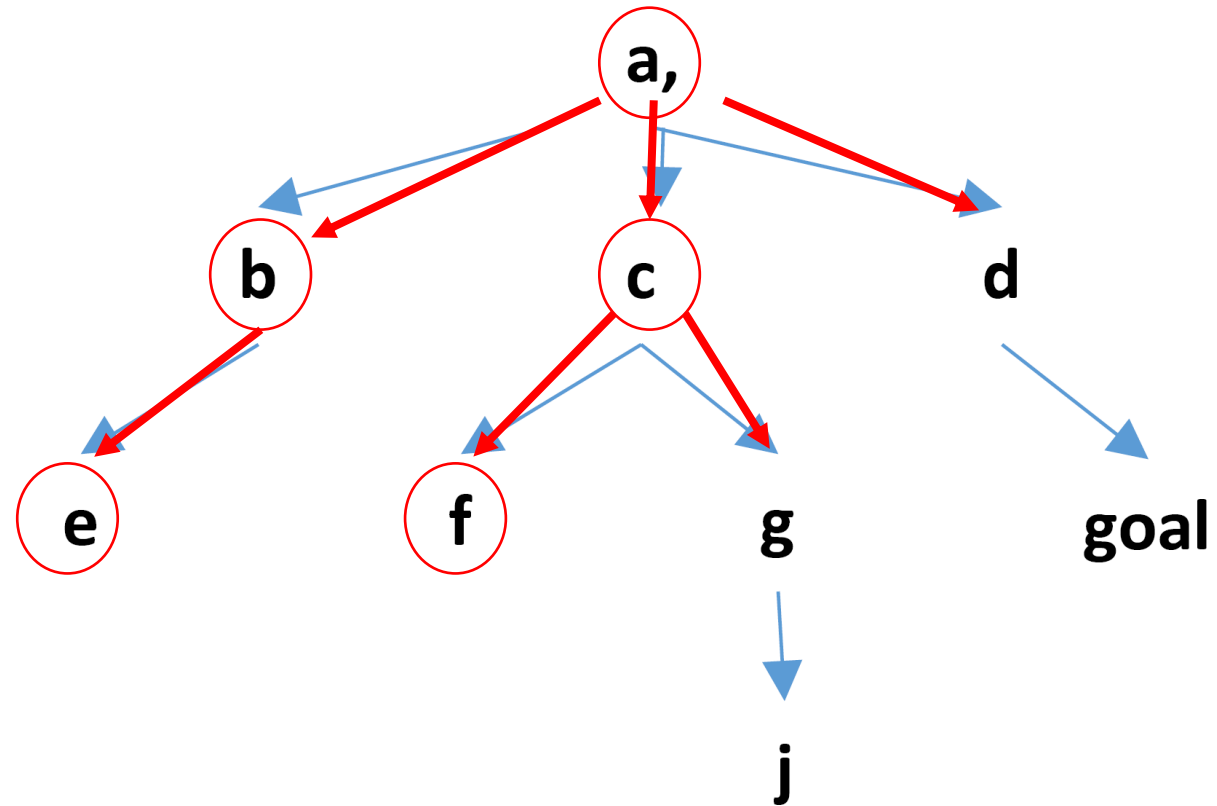
(TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b, e, c, f]

Stack = [g, d]



SEARCH QUIZ QUESTION 1

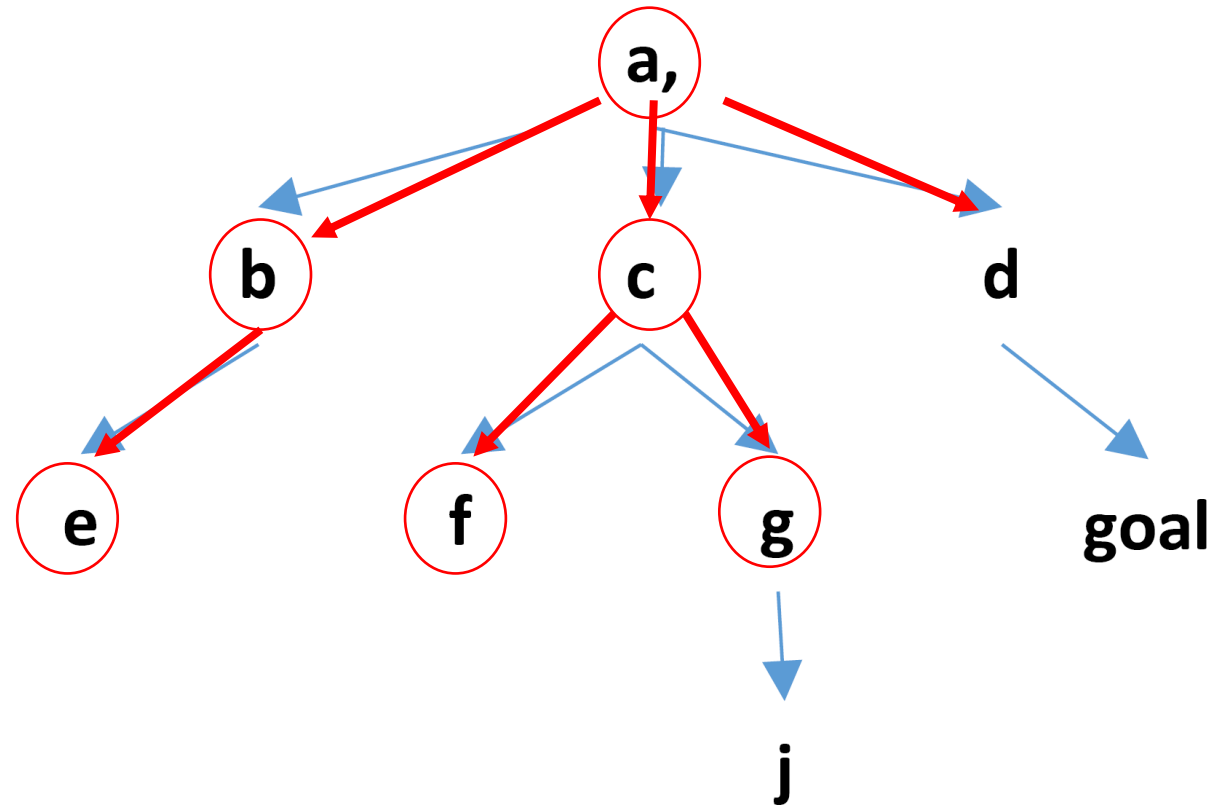
(TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b, e, c, f, g]

Stack = [d]



SEARCH QUIZ QUESTION 1

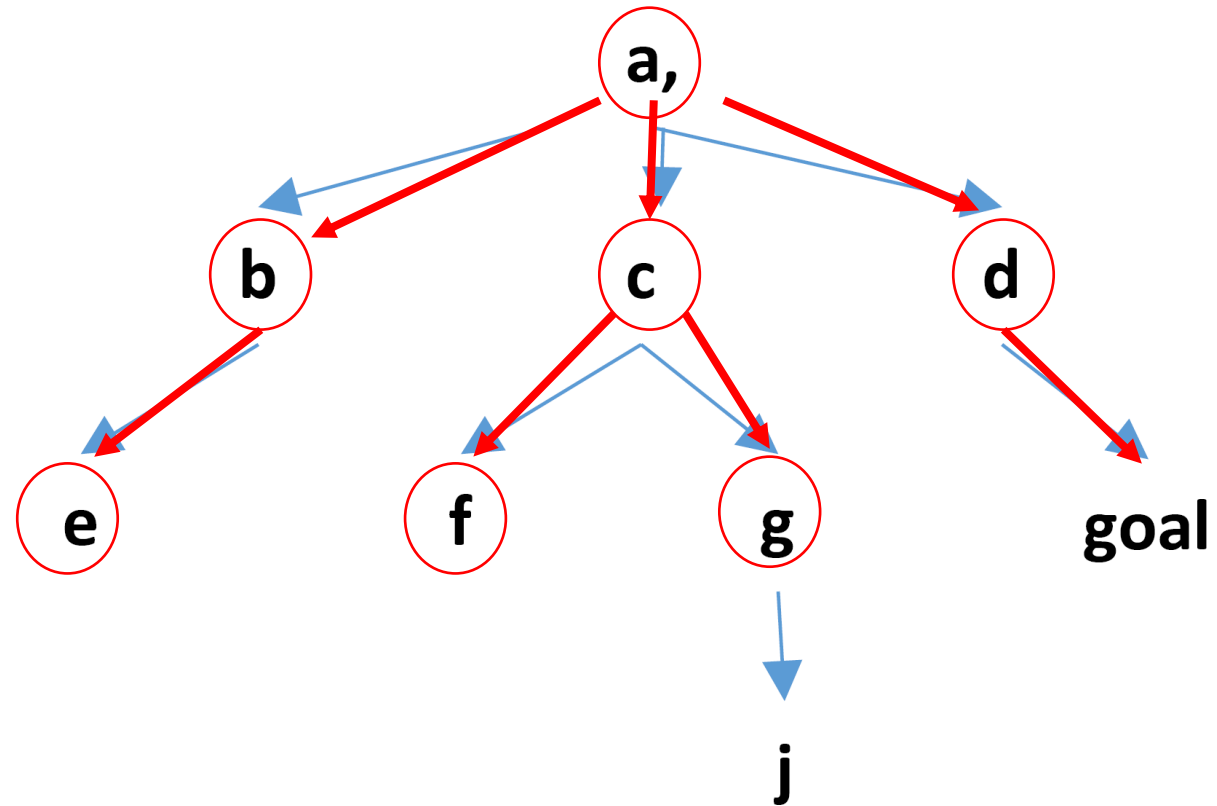
(TASK: APPLY IDS)

Variables

$l = 2$

visited = [a, b, c,
d, a, b, e, c, f, g,
d]

Stack = [goal]



SEARCH QUIZ QUESTION 1

(TASK: APPLY IDS)

Variables

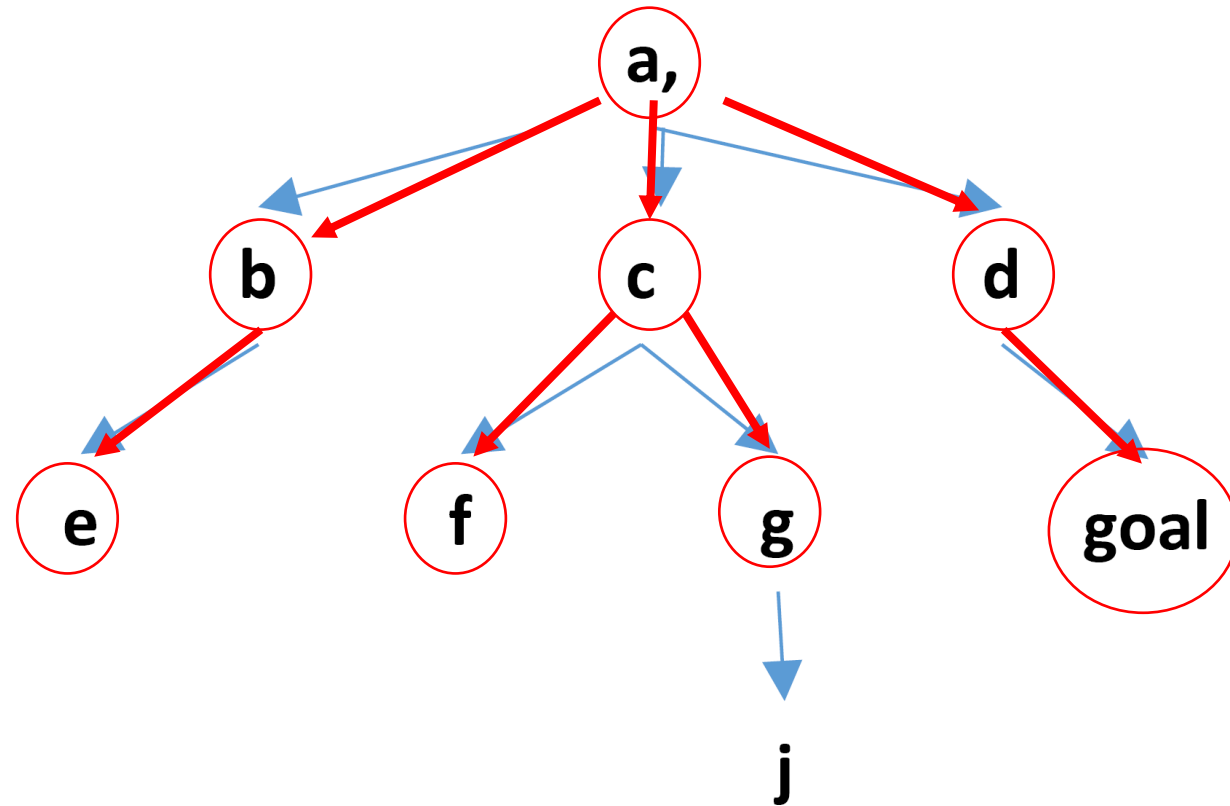
$l = 2$

visited = [a, b, c,
d, a, b, e, c, f, g,
d, goal]

Stack = []

SOLUTION FOUND

abcdabecfgd-goal





INFORMED SEARCH ALGORITHMS

GBFS/A*/IDA*



HEURISTICS (ADDING HUNCHES TO YOUR SEARCH)

- Informed search algorithms use **heuristic functions** to make educated guesses about what **state** to go to next.
- $f(n)$ returns the heuristic's best guess on “how far” the **state** n is from the goal (lower is better, because n is closer).
- Informed search algorithms run $f(n)$ on **states** made available by **actions**, and pick the **state** with the minimum $f(n)$

HEURISTIC EXAMPLE (BIENVENUE DANS L'HEURISTIQUE)



- Consider you are in Paris, and you are trying to walk to the Eiffel Tower.
- You are a couple streets away, but you can see it in the sky.
- You don't know the **optimal solution** to get the Eiffel Tower.
- However, your vision is a **heuristic** that tells you roughly where to go (the bigger/closer it gets, the better)

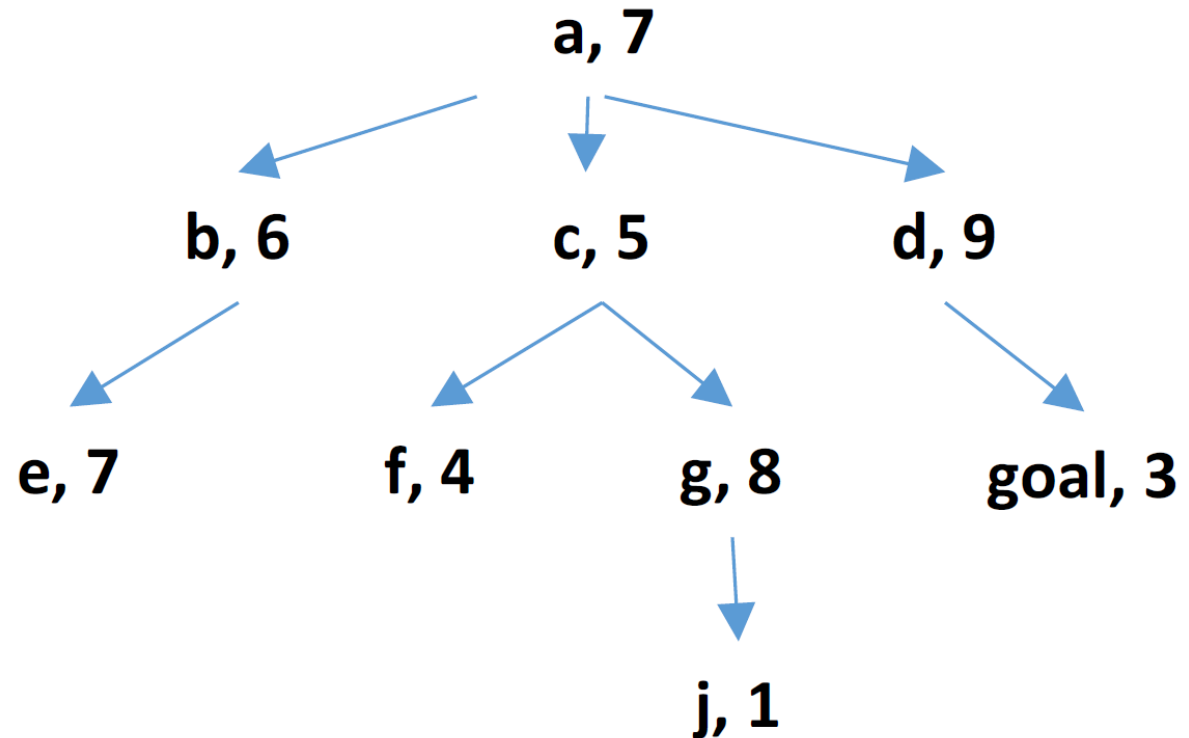
GREEDY BEST-FIRST SEARCH (GBFS) (LIKE NTFS, BUT NOT AT ALL)

- Breadth-First Search, except the queue is a priority queue, and a **state**, n , has the key $f(n)$. In this case, the queue is called the **fringe**.
- Not (cost) optimal, **path** is found using heuristic only.
- Assuming **graph search**:
 - Complete with finite space.
 - $O(b^m)$ time & space complexity.
- Note that GBFS is **NOT complete** without repeated-state checking.

SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables
visited = []
Queue = [a]



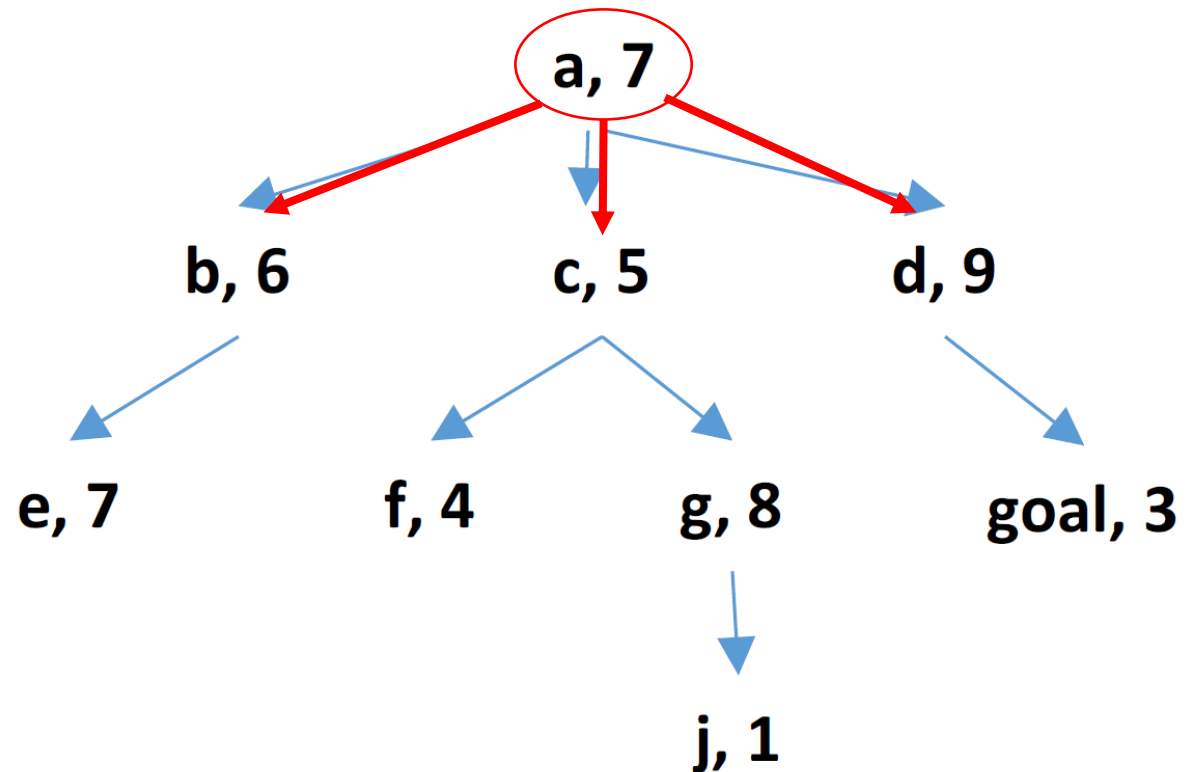
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a]

Queue = [c, b, d]



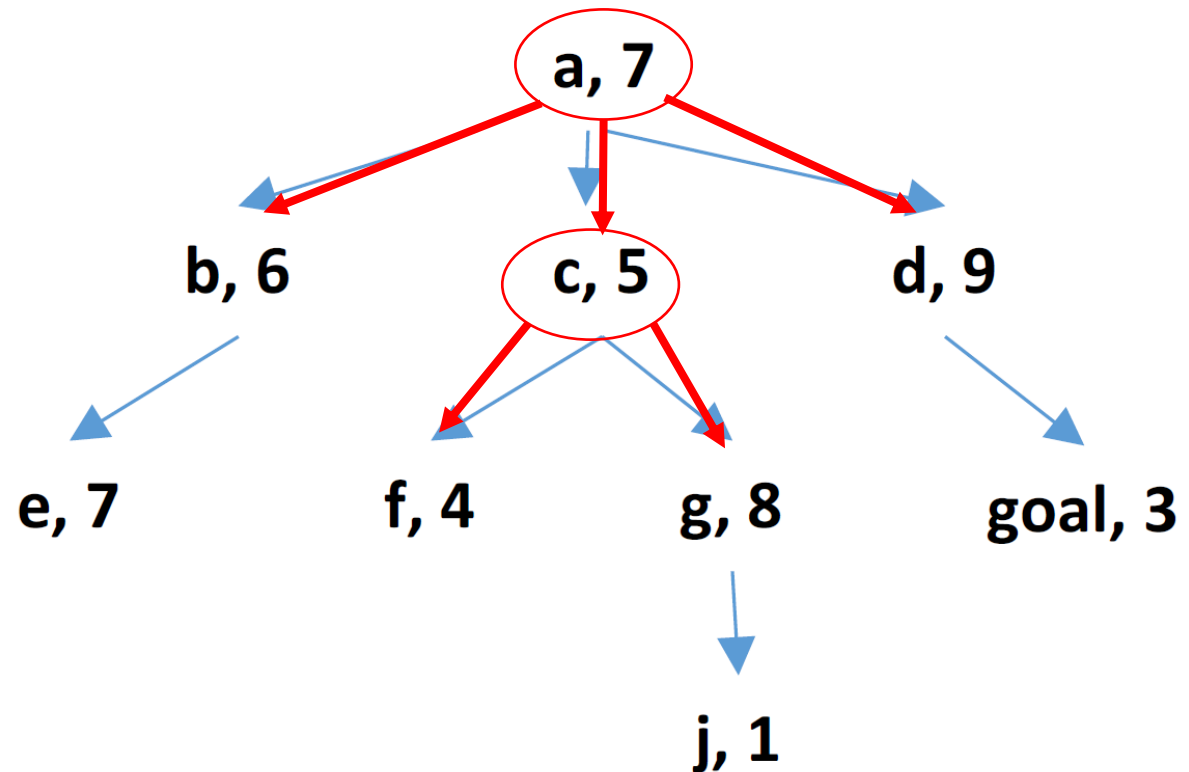
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c]

Queue = [f, b, g, d]



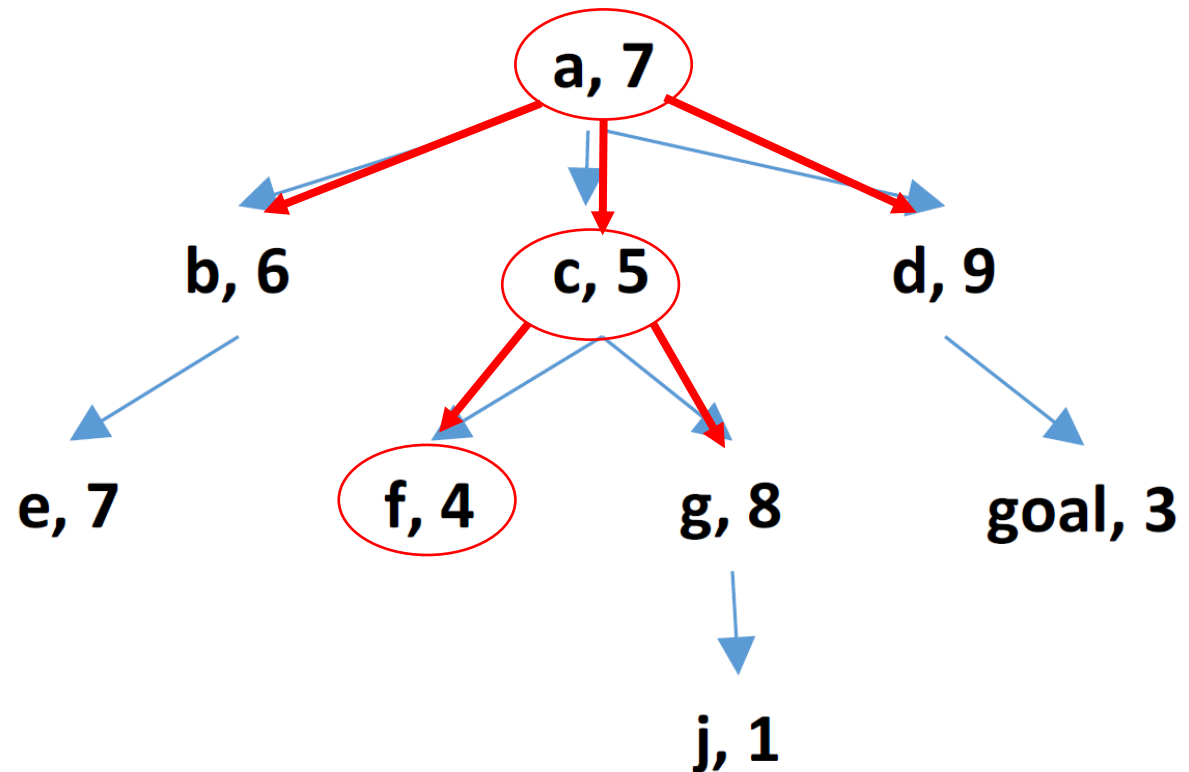
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f]

Queue = [b, g, d]



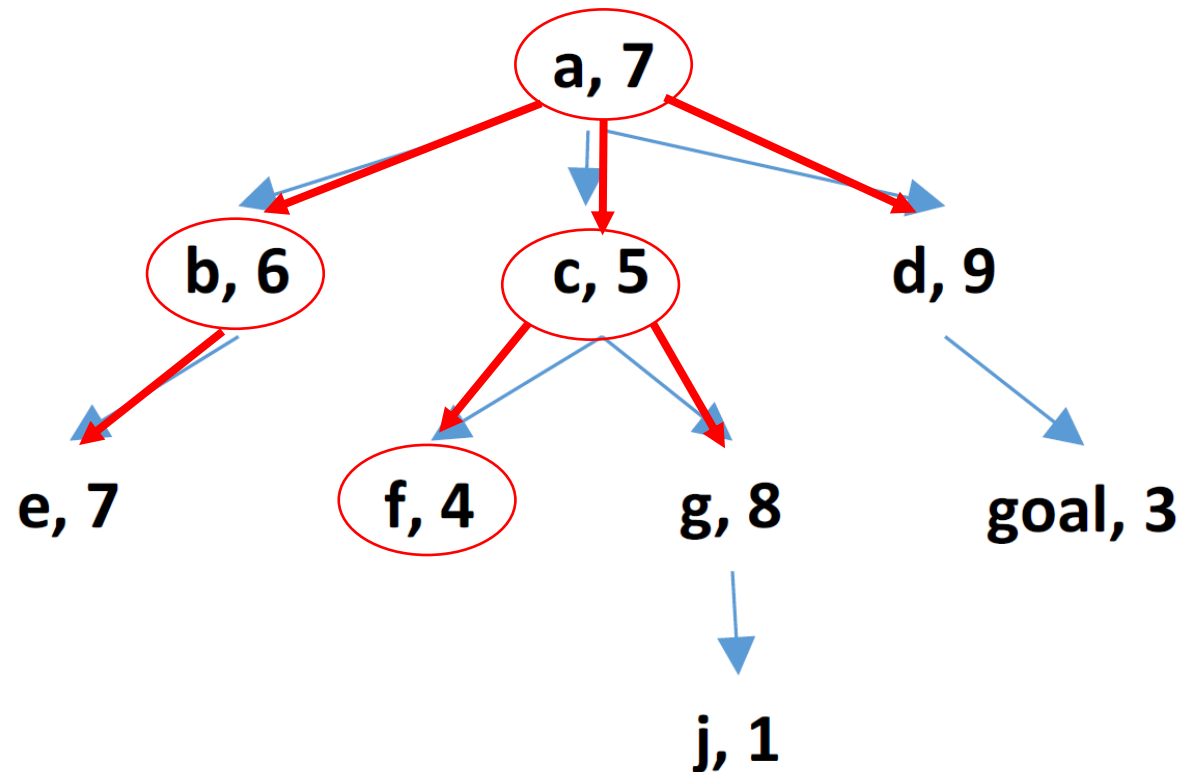
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f, b]

Queue = [e, g, d]



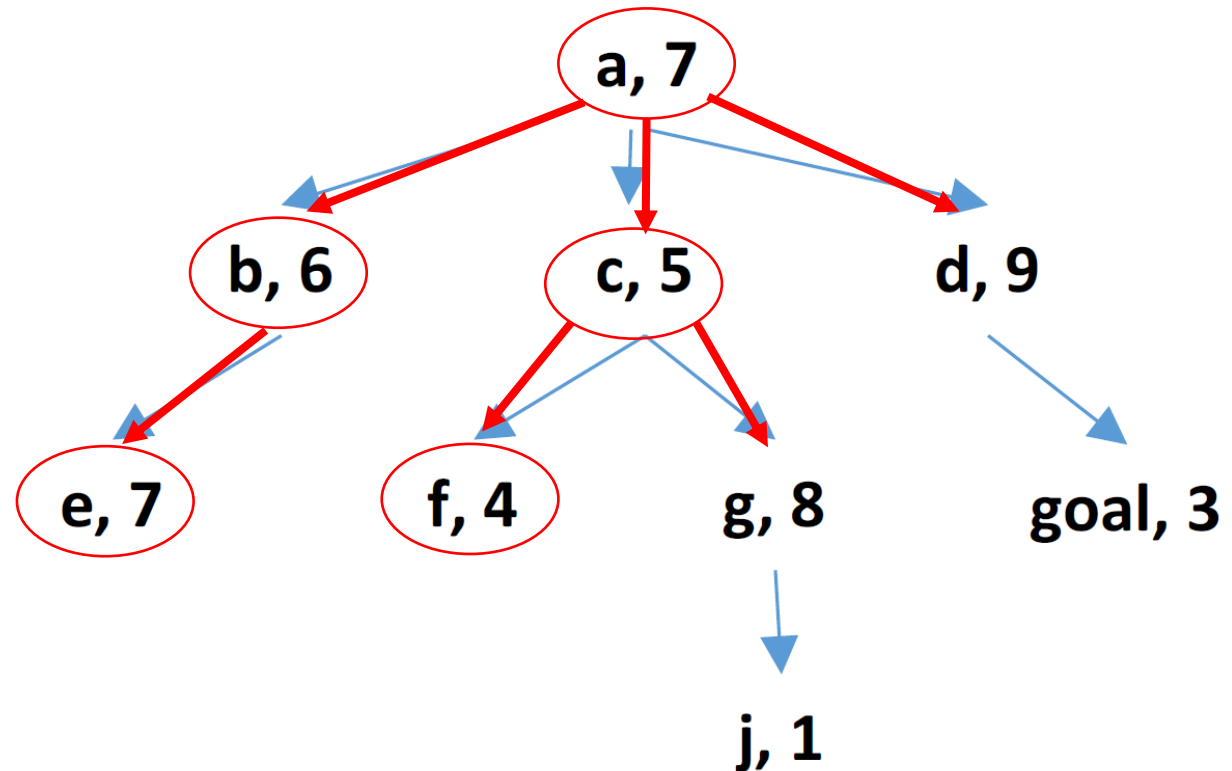
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f,
b, e]

Queue = [g, d]



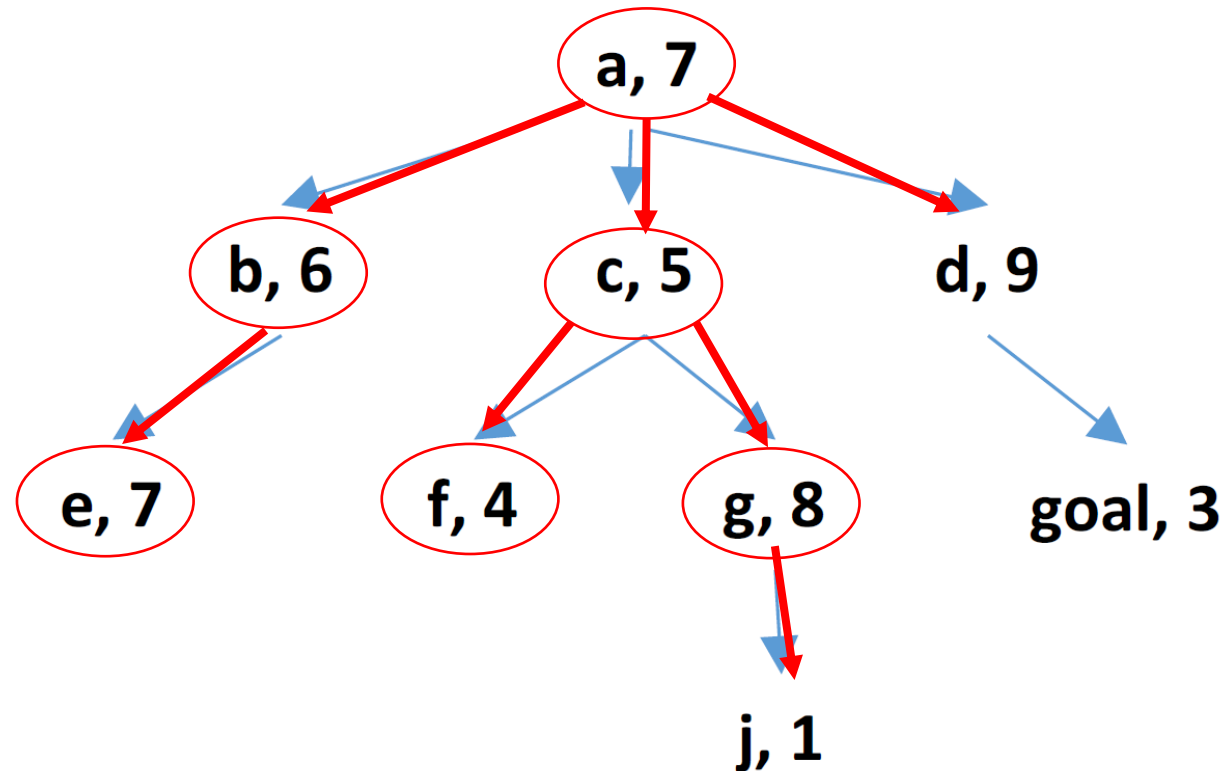
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f,
b, e, g]

Queue = [j, d]



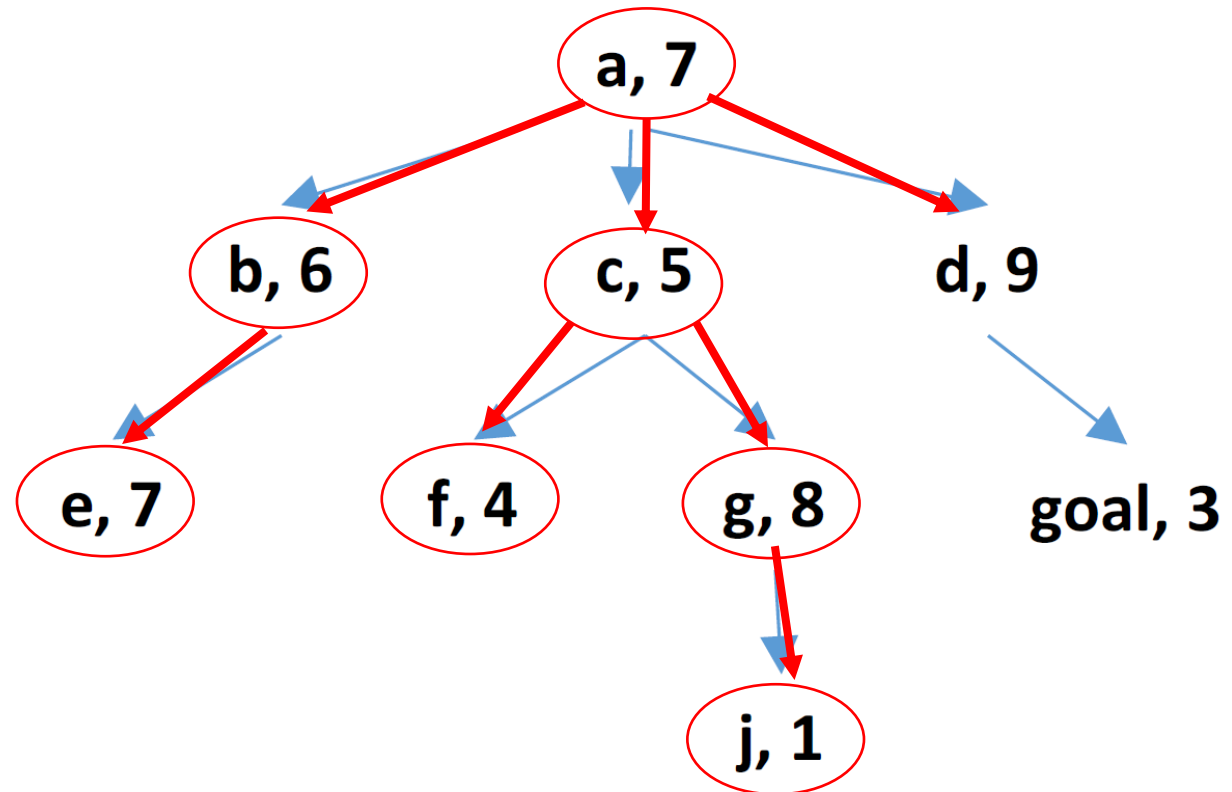
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f,
b, e, g, j]

Queue = [d]



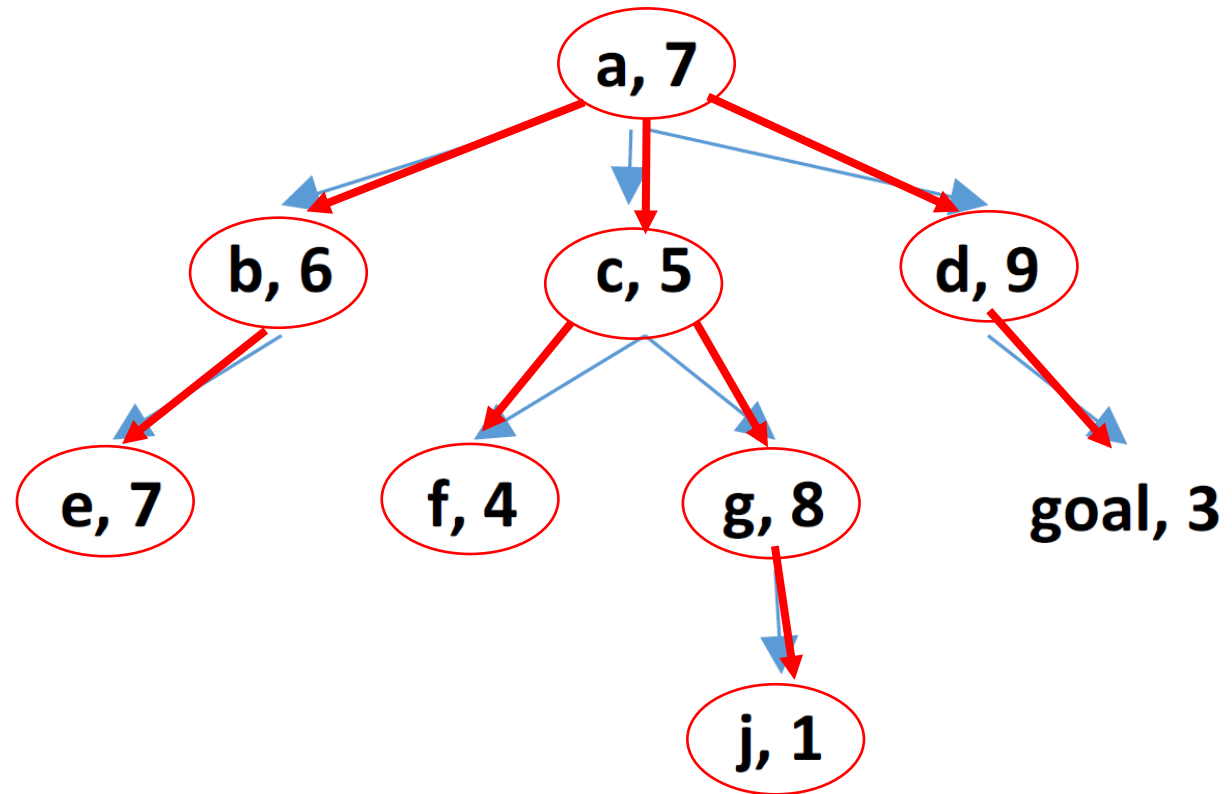
SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

Variables

visited = [a, c, f,
b, e, g, j, d]

Queue = [goal]



SEARCH QUIZ QUESTIONS 2 & 3

(TASK: APPLY GUIDED/INFORMED SEARCH)

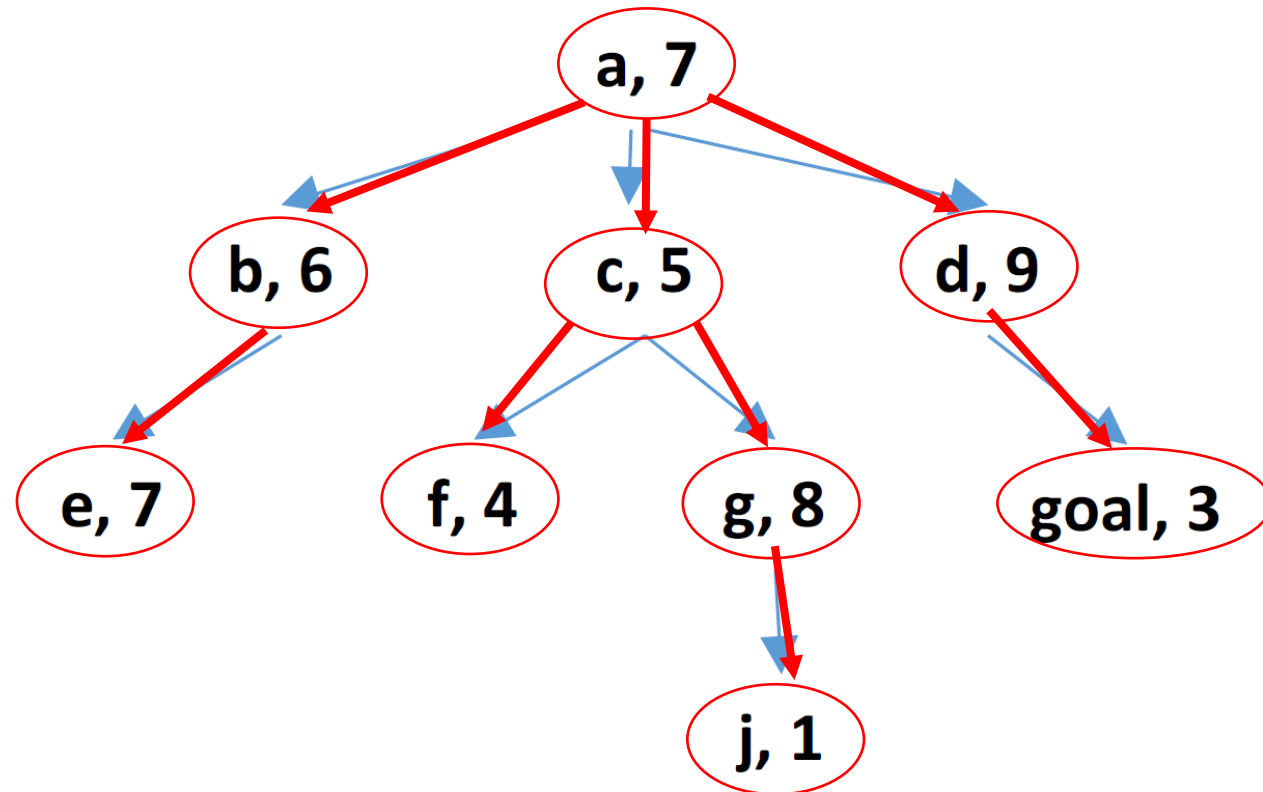
Variables

visited = [a, c, f,
b, e, g, j, d, goal]

Queue = []

SOLUTION FOUND:

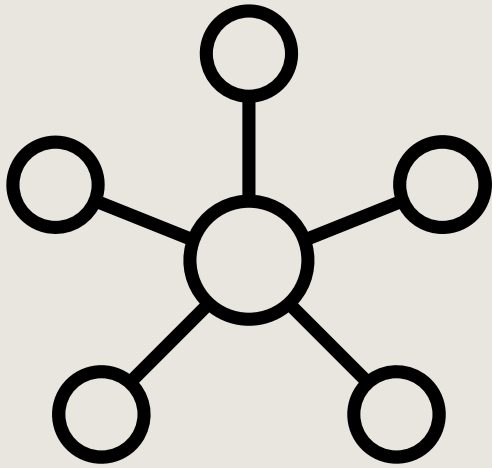
acfbegjd-goal





PROPERTIES OF QUALITY HEURISTICS

- Notice that with the given **heuristic**, GBFS performed **worse** than regular breadth-first search!
- We want quality **heuristics**, so we have three categories of **heuristics**, based on the properties of the **heuristic**:
 - Inconsistent/Inadmissible (Good)
 - Admissible (Better)
 - Consistent (inherently Admissible) (Best)



ADMISSIBLE HEURISTICS (HOW DO I ADMIT A FUNCTION?)

- A heuristic with **admissibility** is an **admissible heuristic**.
 - Condition for **admissibility**: $f(n) \leq \text{distance}(n, \text{goal})$, where n is a **state**.
 - In plain English: the heuristic never **overestimates** the distance from the **state** to the **goal**.
 - If the heuristic **over-estimates**, a minimizer may be **wrongly** guided!
- For an informed search to find the **optimal solution**, its heuristic must be **admissible**.

Step 1 — Create a **relaxed** version of the problem you're developing a heuristic for.

Step 2 — Create a formula that gives the **exact solution cost** of your **relaxed problem**.

Step 3 — Use your formula as a **heuristic** for the original problem.

Note — This **heuristic** will always be **admissible**.

FINDING ADMISSIBLE HEURISTICS

CONSISTENT HEURISTICS (THE OL' RELIABLE OF HEURISTICS)

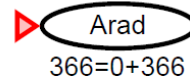
- A heuristic with **consistency** is a **consistent heuristic**.
 - Condition for **consistency**:
 $dist(x, goal) \leq ACTION - COST(x, a, y) + dist(y, goal)$, where *goal* is the **goal state**, *x* and *y* are **states** and *a* is an **action** that **transitions** *x* to *y*.
 - A consistent heuristic does not revisit nodes and makes the search **optimally efficient** (only visiting as many nodes as you need to find the optimal solution)!
- All **consistent** heuristics are **admissible** (does **NOT** apply vice-versa).

A* SEARCH

(REACHING FOR THE *S)

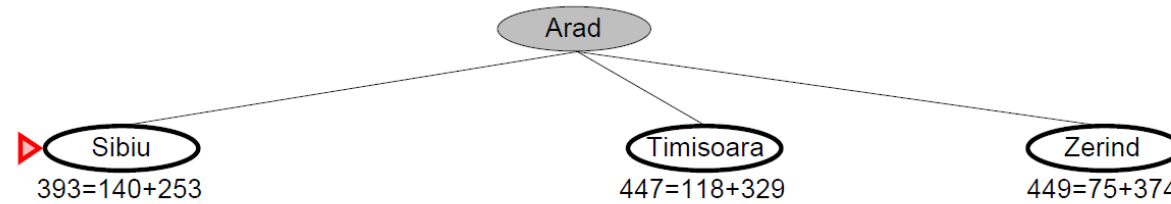
- Modified best-first search that uses **admissible heuristic** function
 $f(n) = g(n) + h(n)$
 - $g(n)$ = actual distance from **initial state** to n .
 - $h(n)$ = estimated distance from n to *goal*.
 - $h(n) \geq 0$ for all n
- A* is **complete** and **(cost) optimal**.
- When A* uses a **consistent heuristic**, we say it is **optimally efficient**, meaning it visits only as many **states** as are necessary to find the **optimal solution**.
- However, A* keeps all nodes in memory, meaning it has exponential **time and space complexity**.
- Additionally, A* may get stuck if it has an **inadmissible heuristic**.
- Therefore, some problems may have too large of a **state space** or too weak of a **heuristic** to use A*.
- Solution? **IDA***

A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)

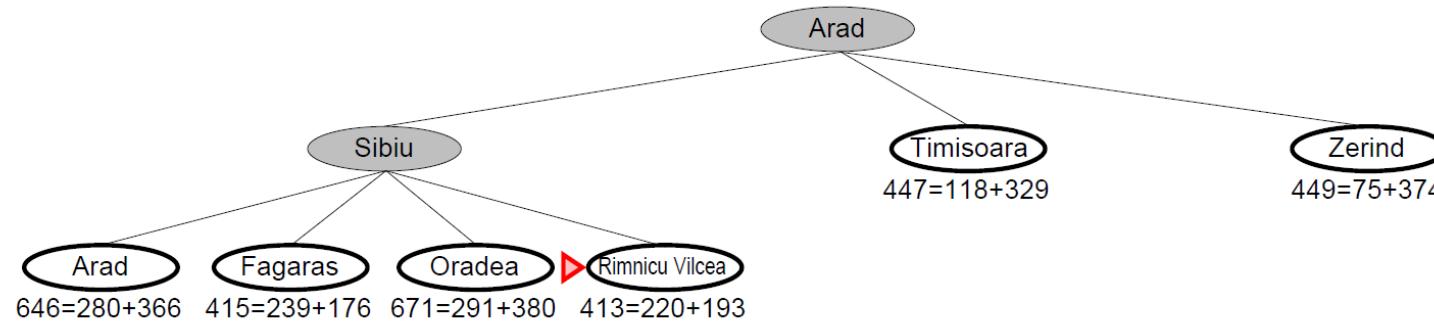


Arad
 $366=0+366$

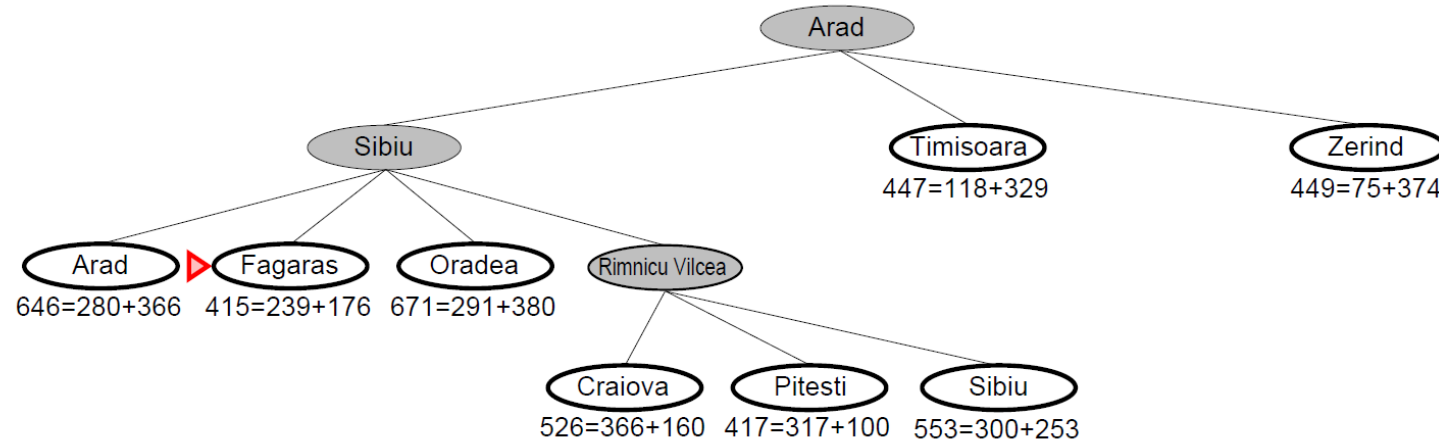
A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)



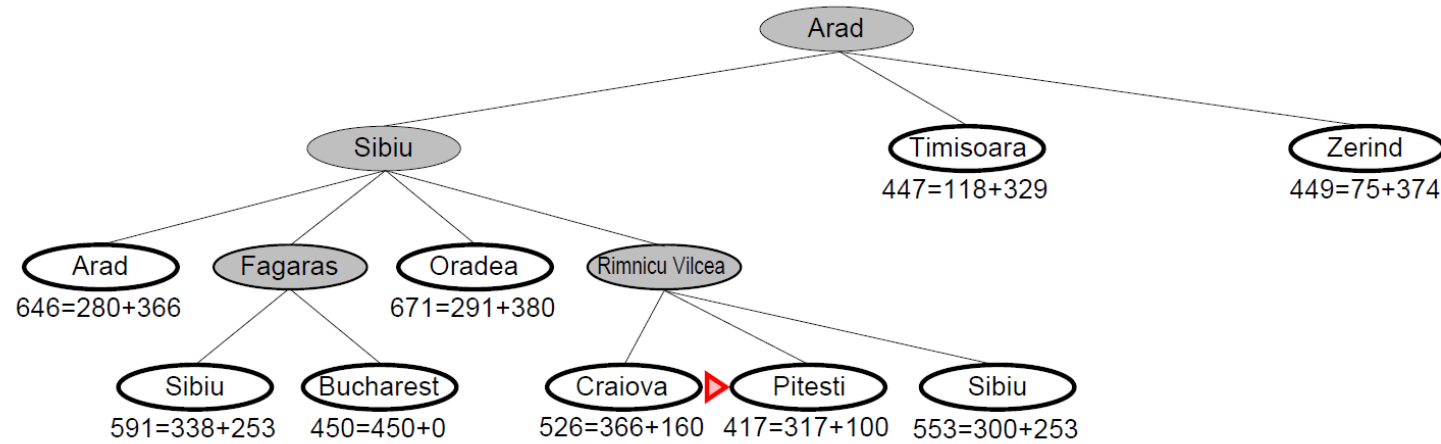
A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)



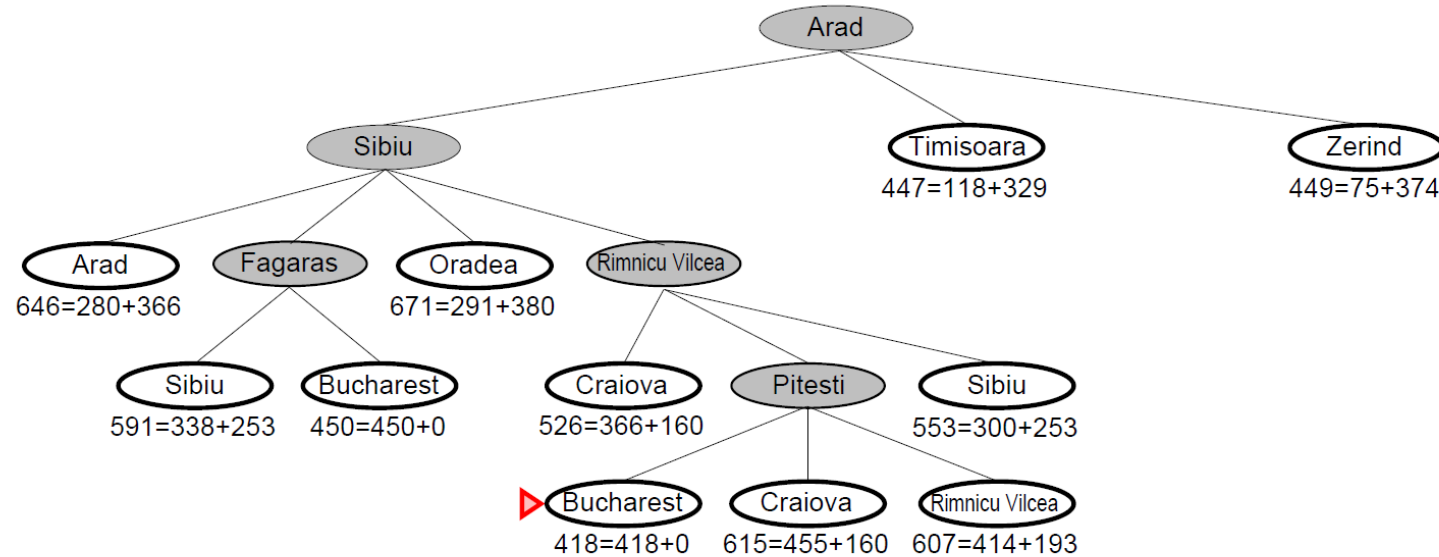
A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)



A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)



A* SEARCH EXAMPLE (ROMANIAN FIELD TRIP, ANYONE?)

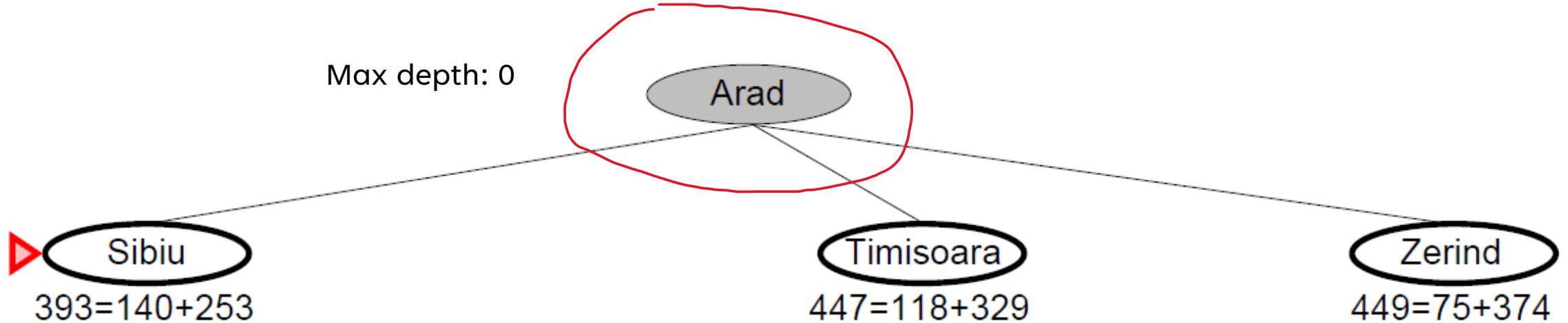


ITERATIVE DEEPENING A* (IDA*) (I MEAN, IT WORKED FOR DFS?)

- Iterative executions of A*, where a maximum depth is set and incremented by the smallest $f(n)$ that exceeds the maximum depth each run.
- IDS + A* provides guarantee of finding the solution for an additional cost.
- **Book** suggests IDA* is typically implemented as a **tree search**, contrasting A* which is typically implemented as a **graph search**.

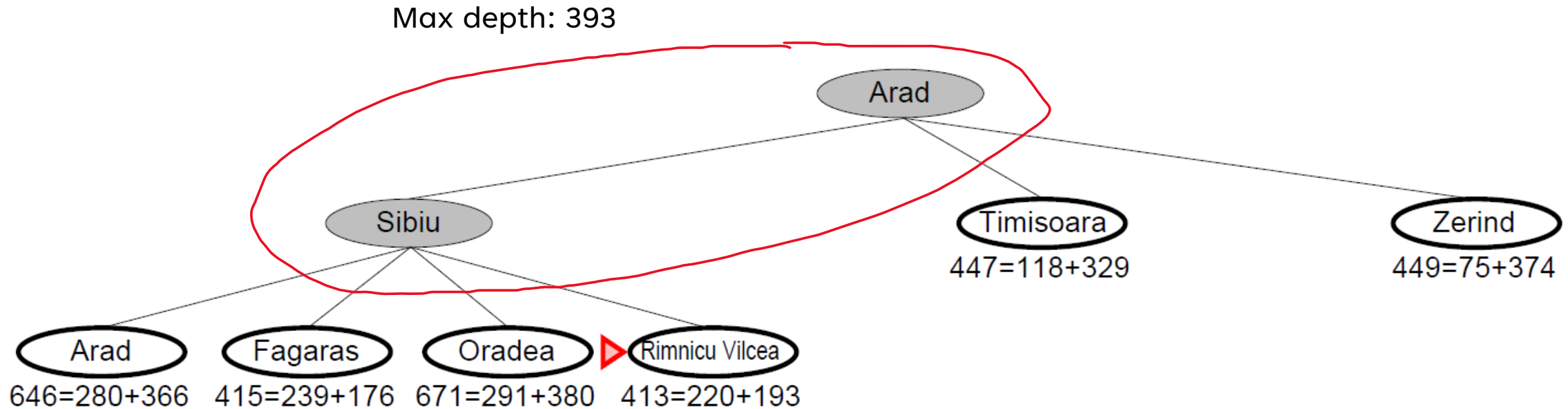
IDA* SEARCH EXAMPLE

(WHILE(!GOALFOUND) {ROMANIAN FIELD TRIP, ANYONE?})



IDA* SEARCH EXAMPLE

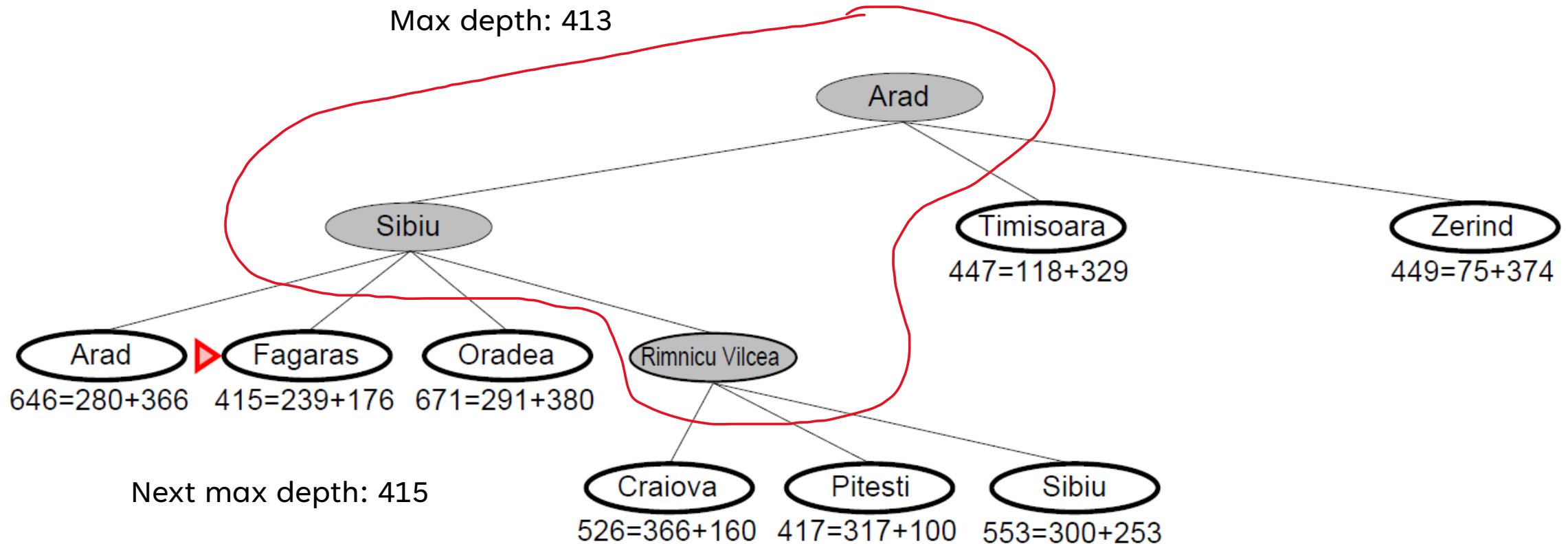
(WHILE(!GOALFOUND) {ROMANIAN FIELD TRIP, ANYONE?})



Next max depth: 413

IDA* SEARCH EXAMPLE

(WHILE(!GOALFOUND) {ROMANIAN FIELD TRIP, ANYONE?})





THANK YOU

Joshua Sheldon

jsheldon2022@my.fit.edu