Artificial Intelligence

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

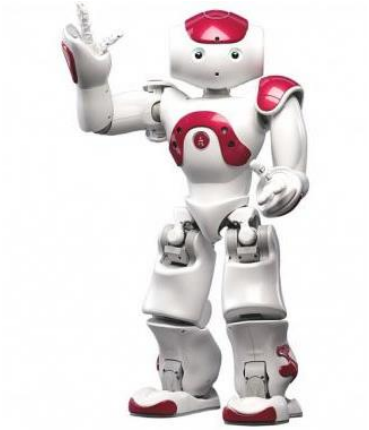
Examples of AI?

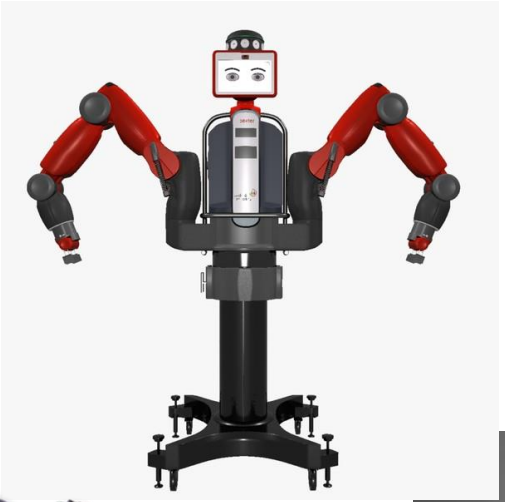
Examples of AI

Games

Examples of AI

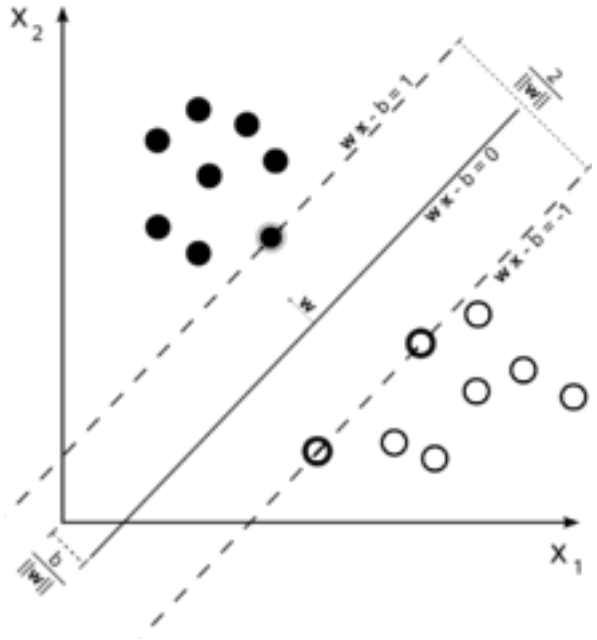
Robots

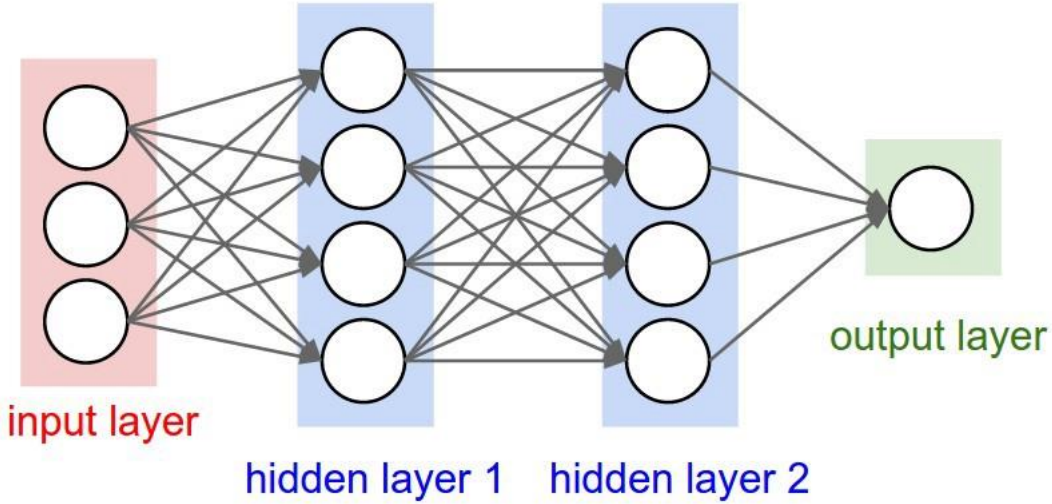




Examples of AI

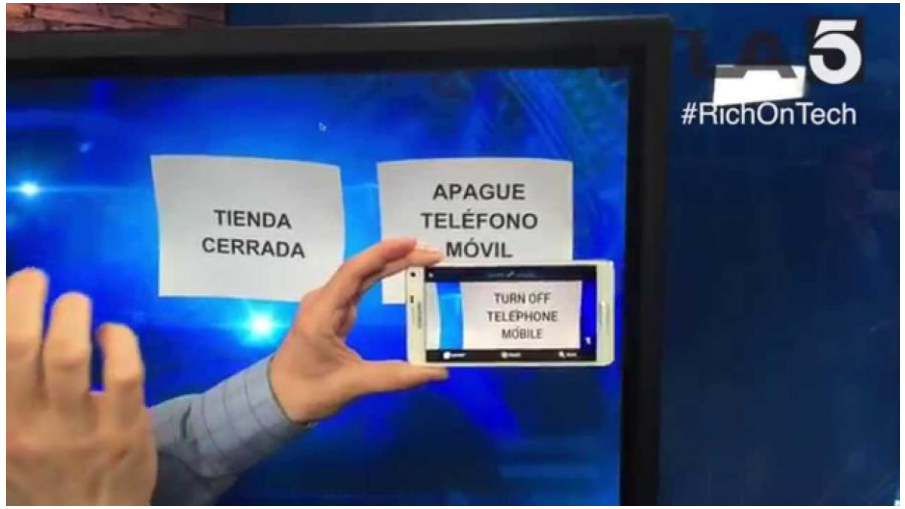
Machine Learning





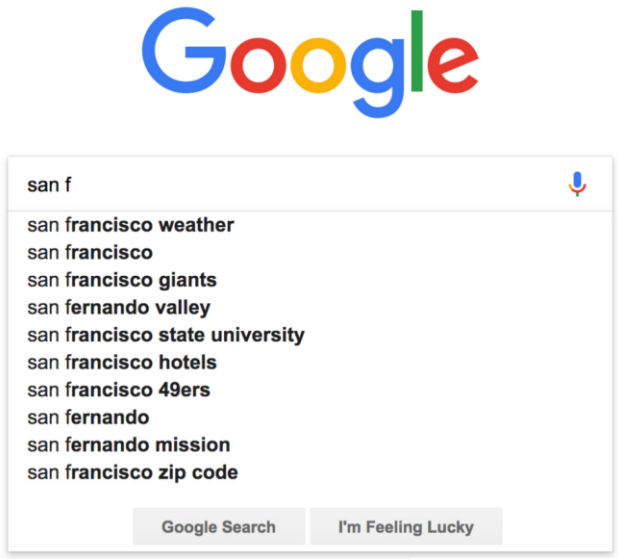
Examples of AI

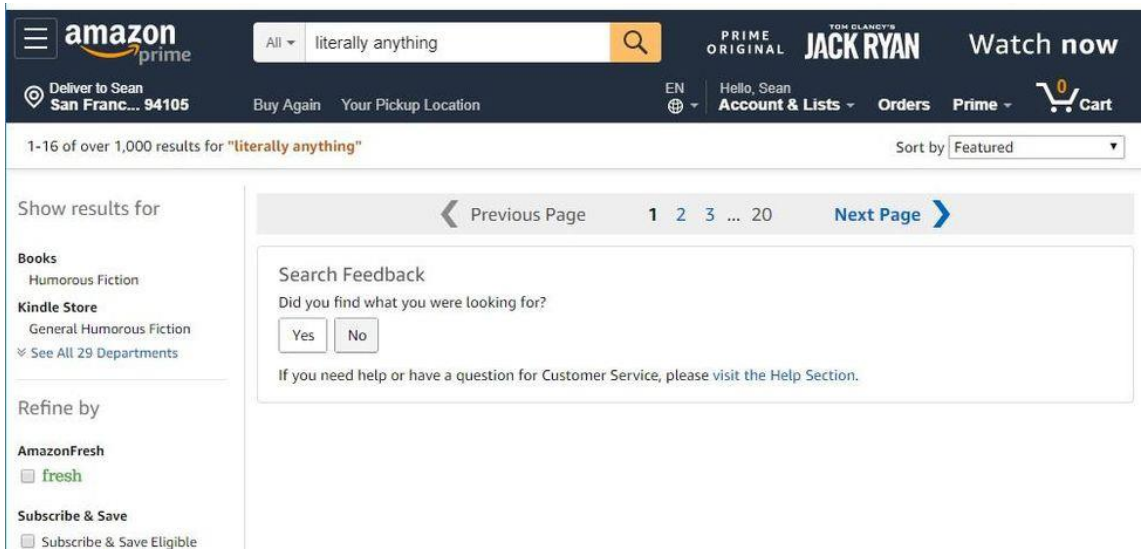
Language and Question Answering



Examples of AI

Search





Examples of AI

Finance, Fraud, Lending

What is AI?

What is AI?

Algorithms that perceive the environment and take actions to maximize the chance of achieving their goals

Wikipedia

What is AI?

Algorithms that perceive the environment and take actions to maximize the chance of achieving their goals

Wikipedia

What is AI?

Algorithms that perceive the environment and take actions to maximize the chance of achieving their goals

Two Goals:

AI Agents – can we recreate intelligence?

AI Tools – can we benefit people and society?

AI Agents

Perception

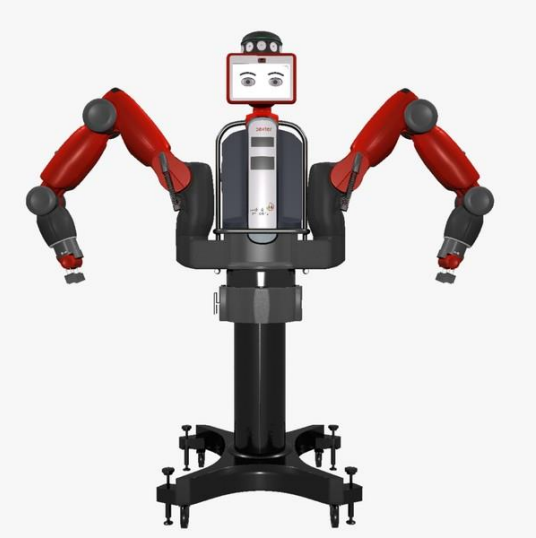
Robotics

Language

Knowledge

Reasoning

Learning



Example: Amazon Alexa

Microphones listen for speech
The audio file is sent to the cloud
The server parses the speech
and finds meaning in it
It determines a good response
that matches the question
The server sends the response back
to Alexa to speak

Alexa is trying to respond in a HUMAN way

AI Tools


Google
san f
san francisco weather
san francisco
san francisco giants
san fernando valley
san francisco state university
san francisco hotels
san francisco 49ers
san fernando
san fernando mission
san francisco zip code
http://www.amazon.com/
Favorites
Suggested Sites
Amazon.com: Online Shopping for Electronic
amazon.com
Hello. Sign
Your Ama
Shop All Departments
Search

Example: Netflix Recommendations

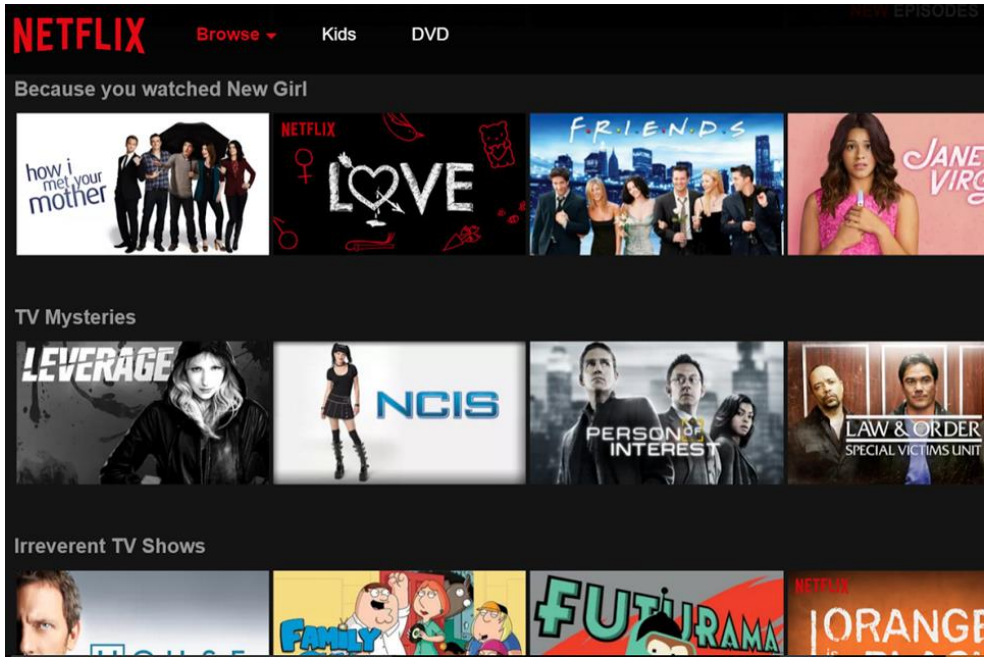
The website records every show you've watched and for how long

It records everyone else's as well

Netflix also knows about movies/shows and what they're about

It tries to look through all users who are similar to you

Whatever those users watched that you haven't may be something you like



Netflix is trying to do something super-human by comparing you to so many others

Example: Self-Driving Car

It uses sensors that aren't particularly human-like

It drives safer than a human

It must understand the driving conventions that people do

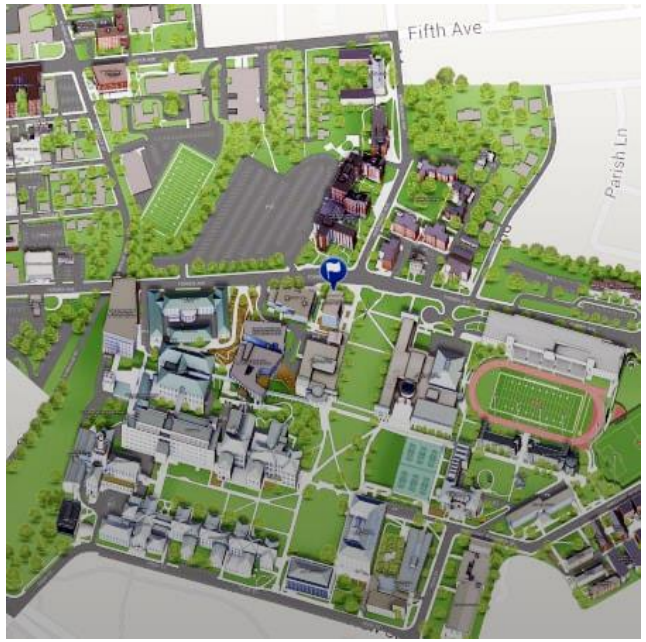
It must follow the same conventions to avoid confusion

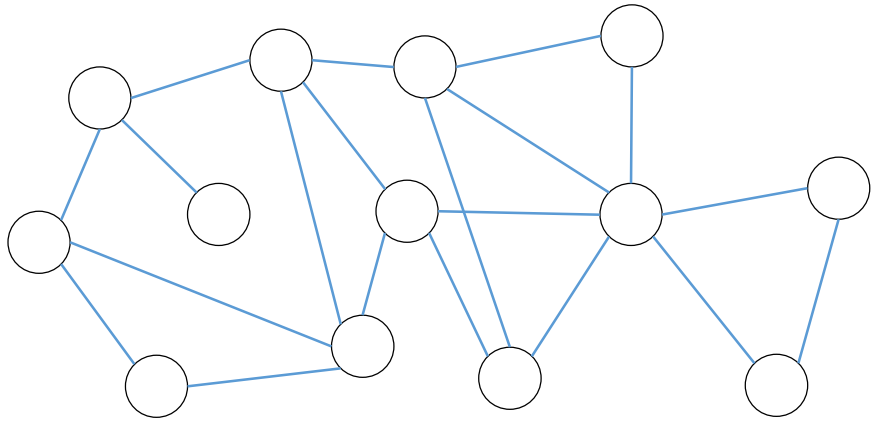
Common Research/Development Paradigm

Modeling

Common Research/Development Paradigm

Modeling



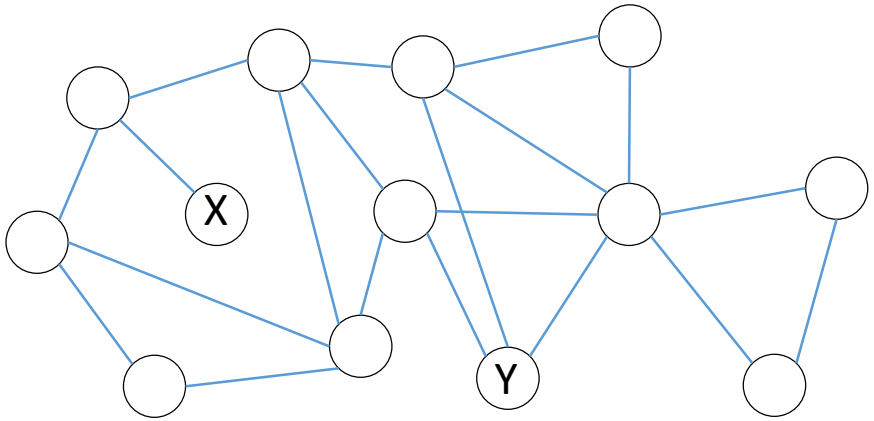


Common Research/Development Paradigm

Common Research/Development Paradigm

Modeling → Inference

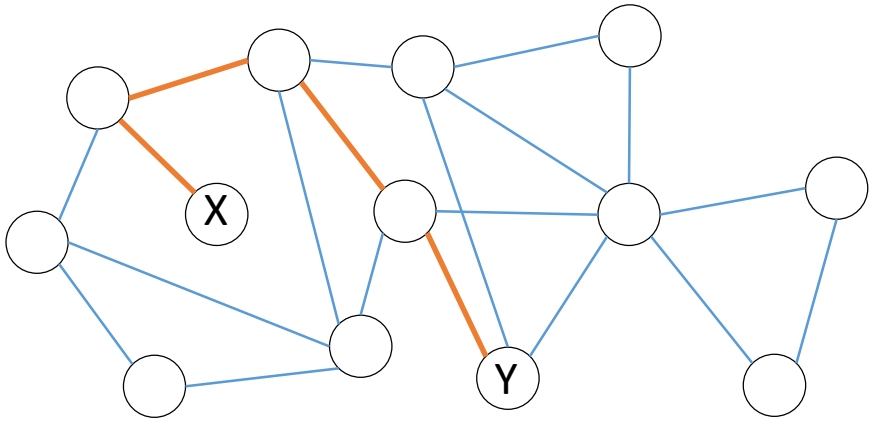
What is the shortest path from X to Y?



Common Research/Development Paradigm

Modeling → Inference

What is the shortest path from X to Y?

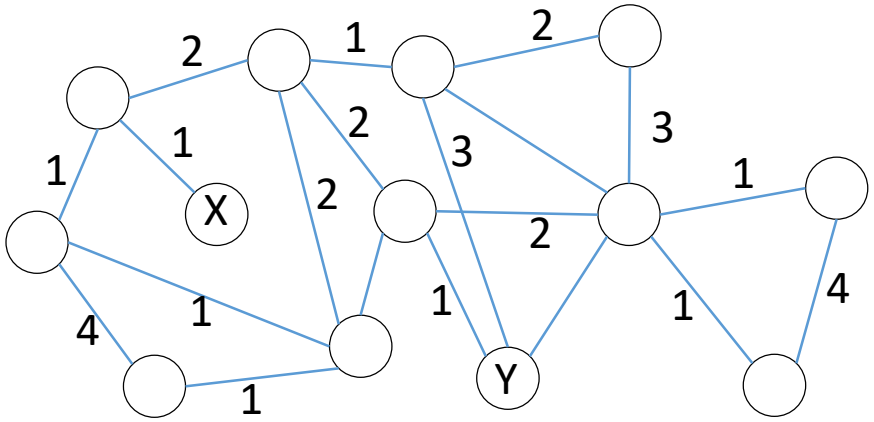


Common Research/Development Paradigm

Common Research/Development Paradigm

Modeling ⇄ Inference

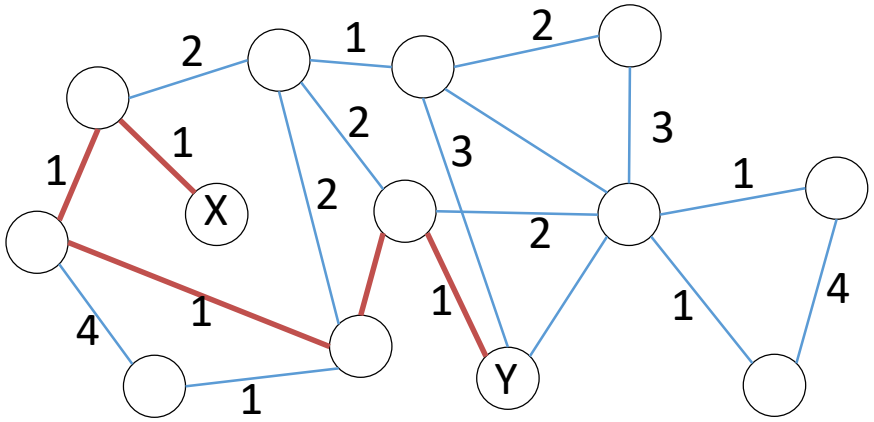
Add Lengths



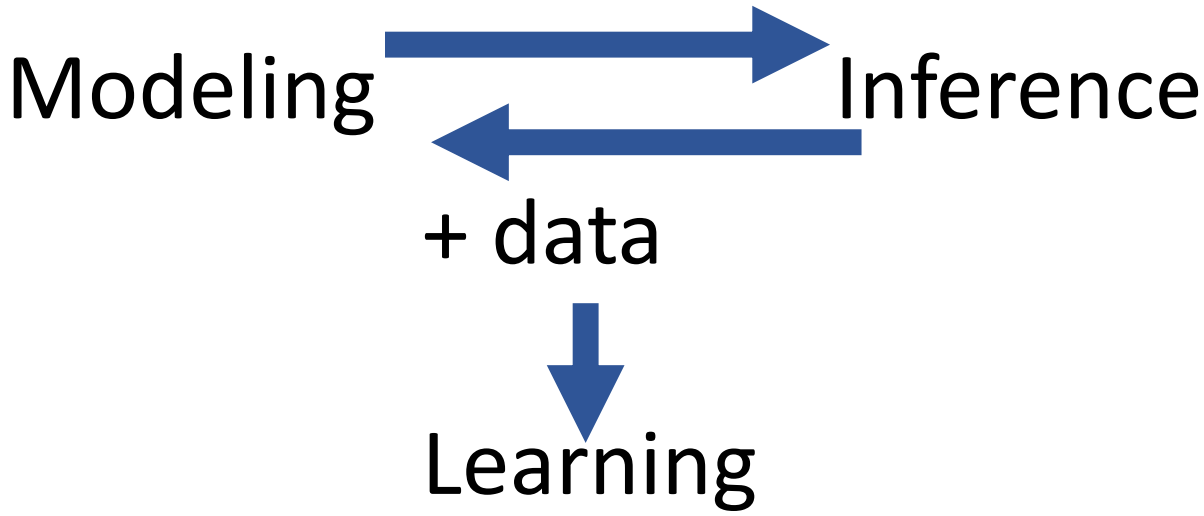
Common Research/Development Paradigm

Modeling ⇄ Inference

What is the shortest path from X to Y?

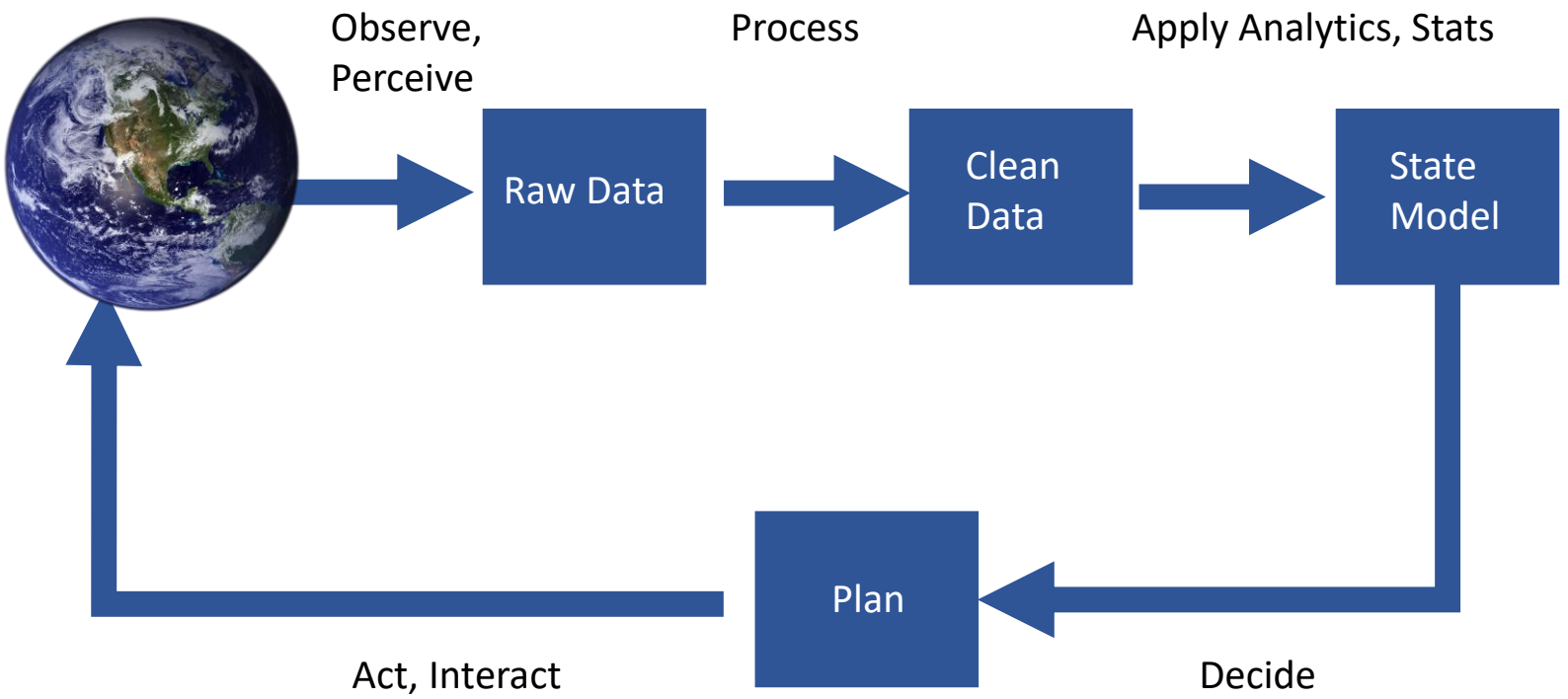


Common Research/Development Paradigm

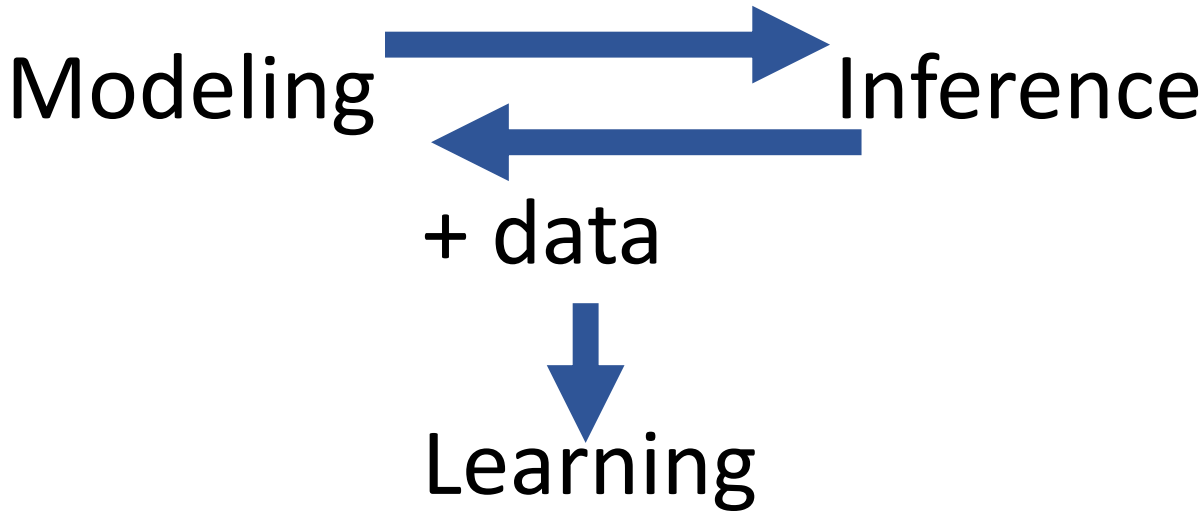


Common Research/Development Paradigm

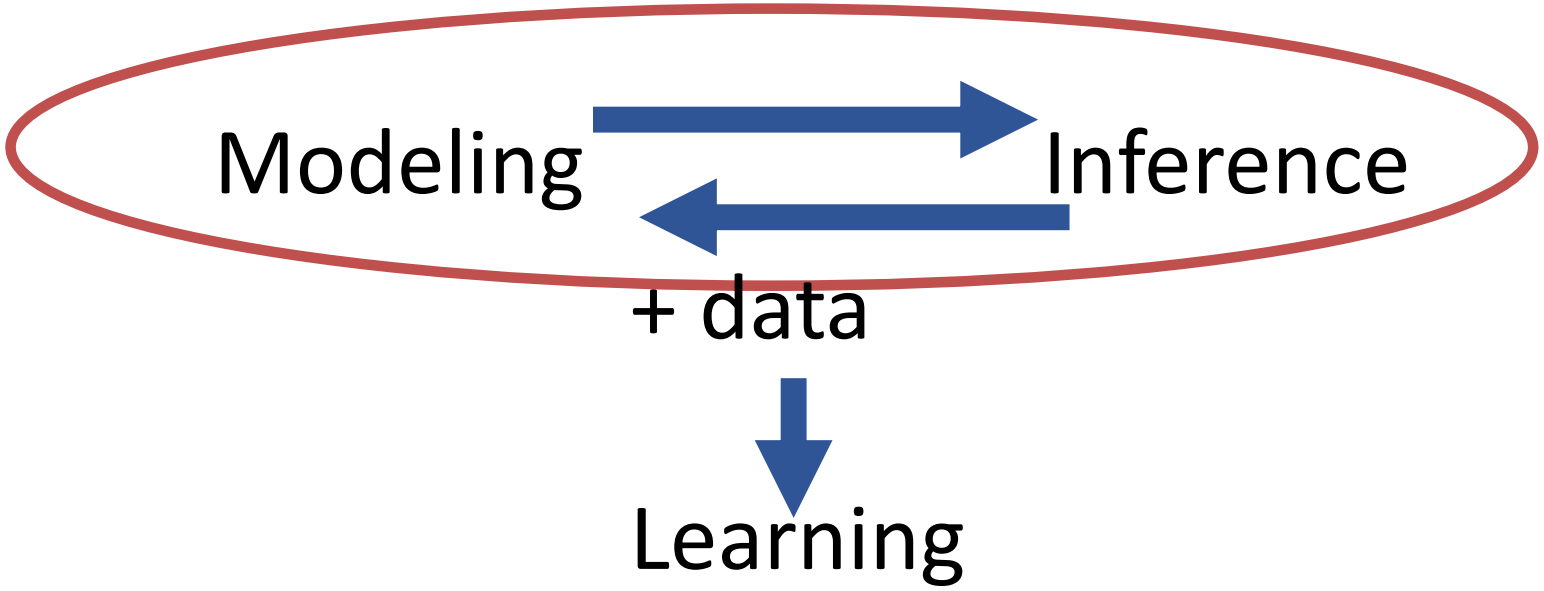
Sensing, Cognition, Action Loop



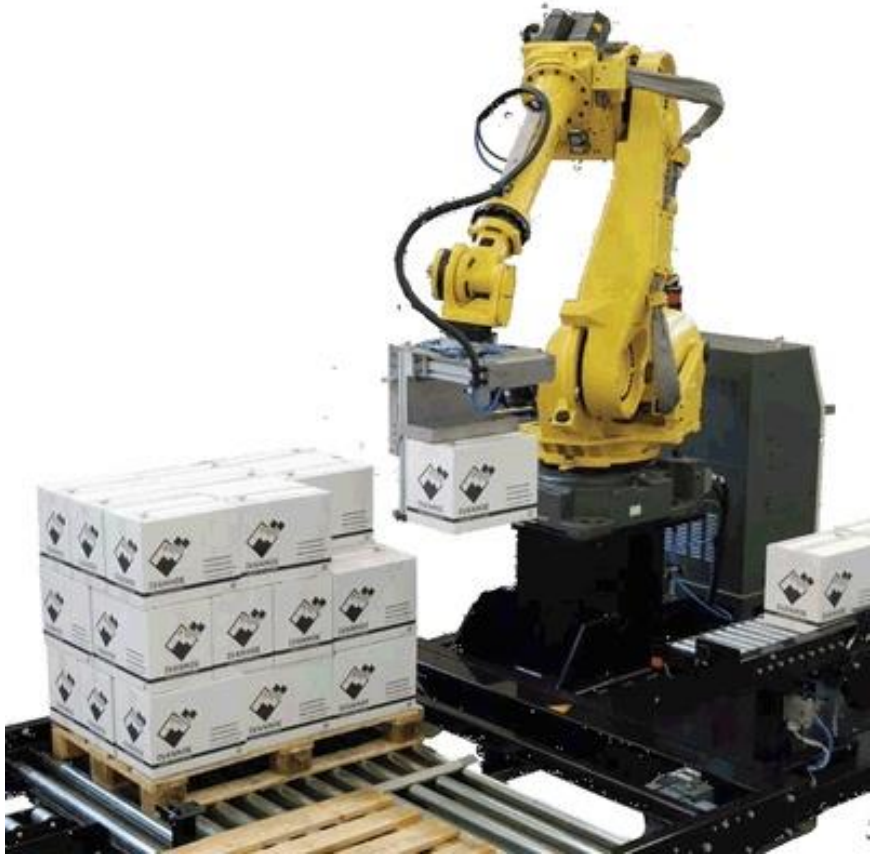
Common Research/Development Paradigm



Common Research/Development Paradigm



Robot Block Stacking



Robot Block Stacking

How can we tell the robot to stack and unstack blocks from/to any configuration?

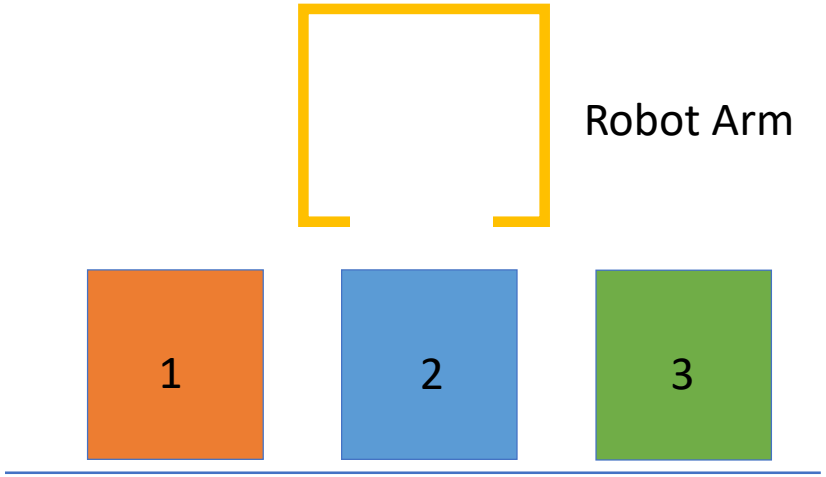
Planning

Given an initial state, generate a sequence of actions that transforms the world into the goal state

Planning

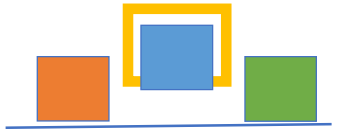
Given an initial state, generate a sequence of actions that transforms the world into the goal state

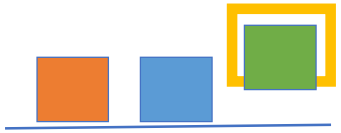
Modeling Block Stacking



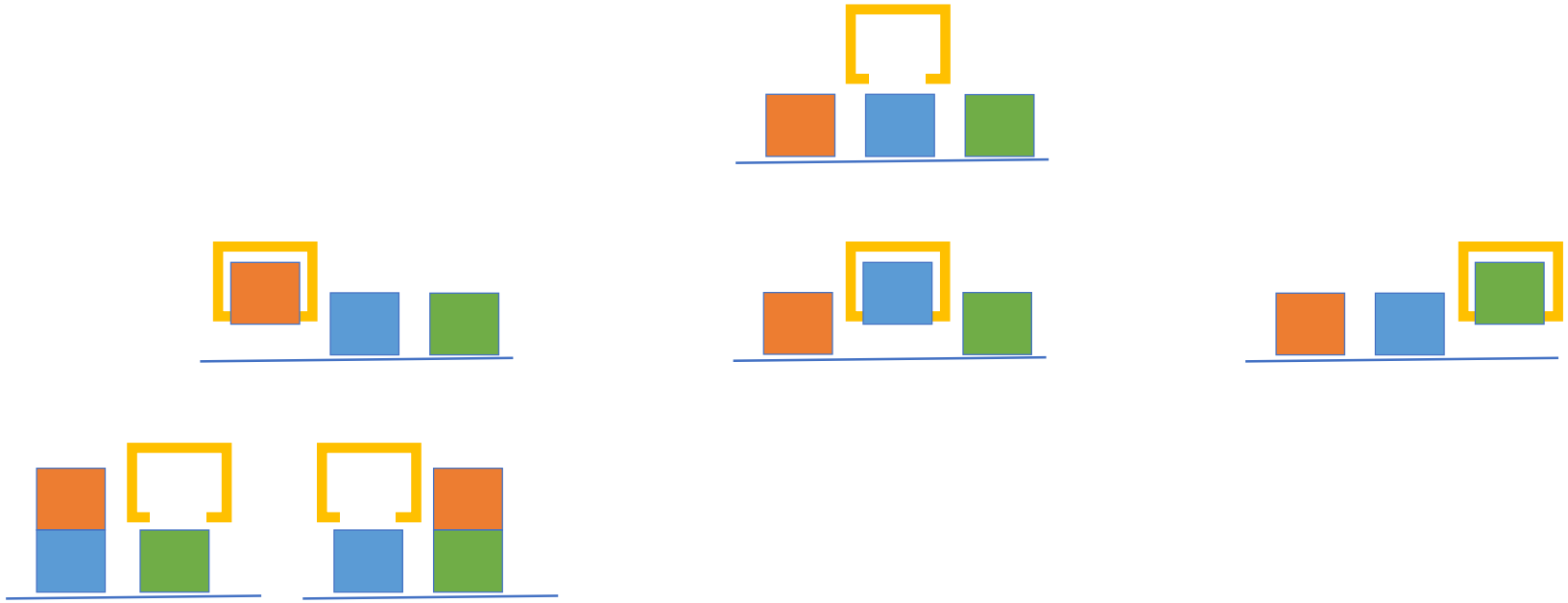
Block Stacking States

Block Stacking States



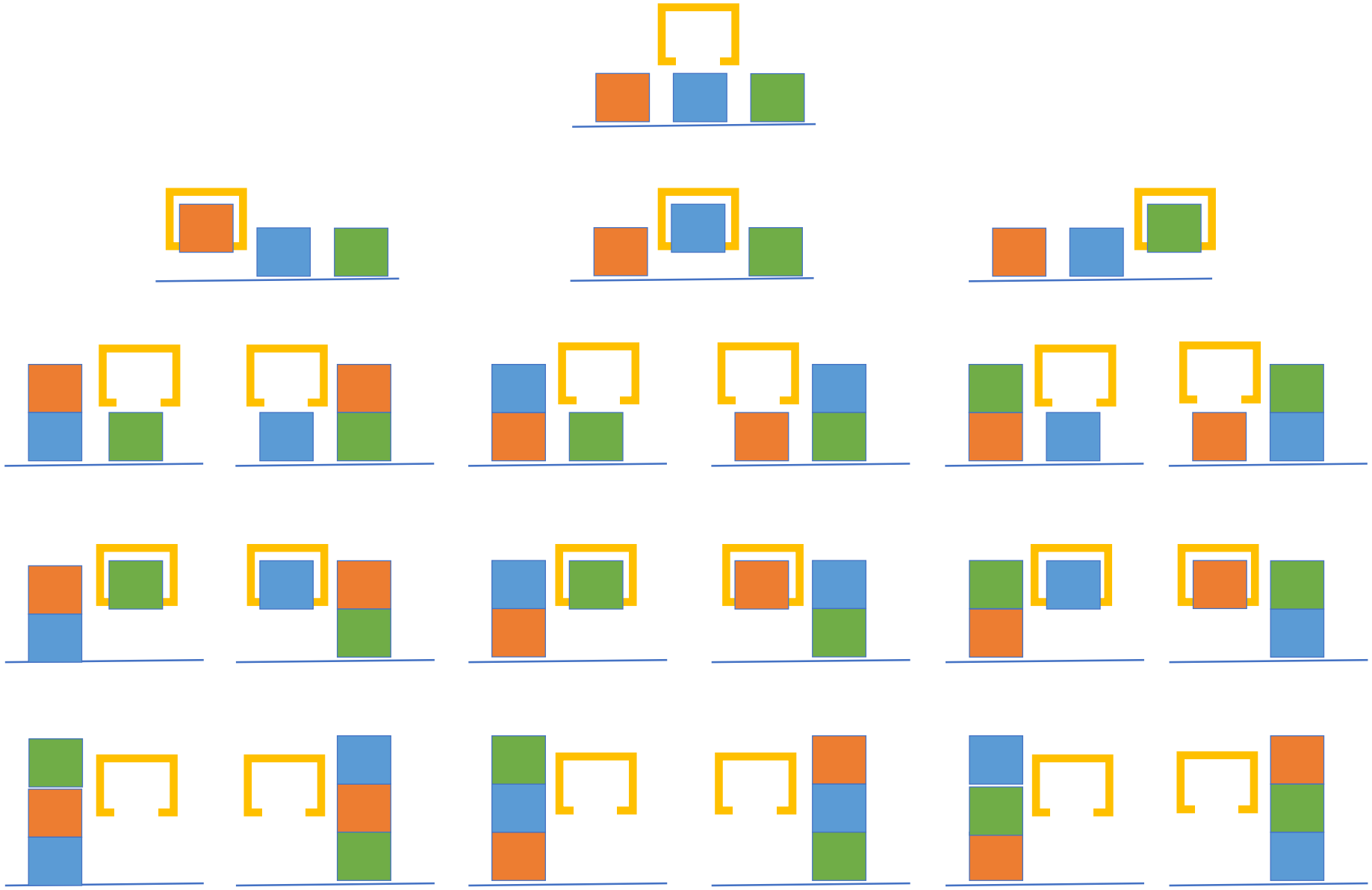


Block Stacking States

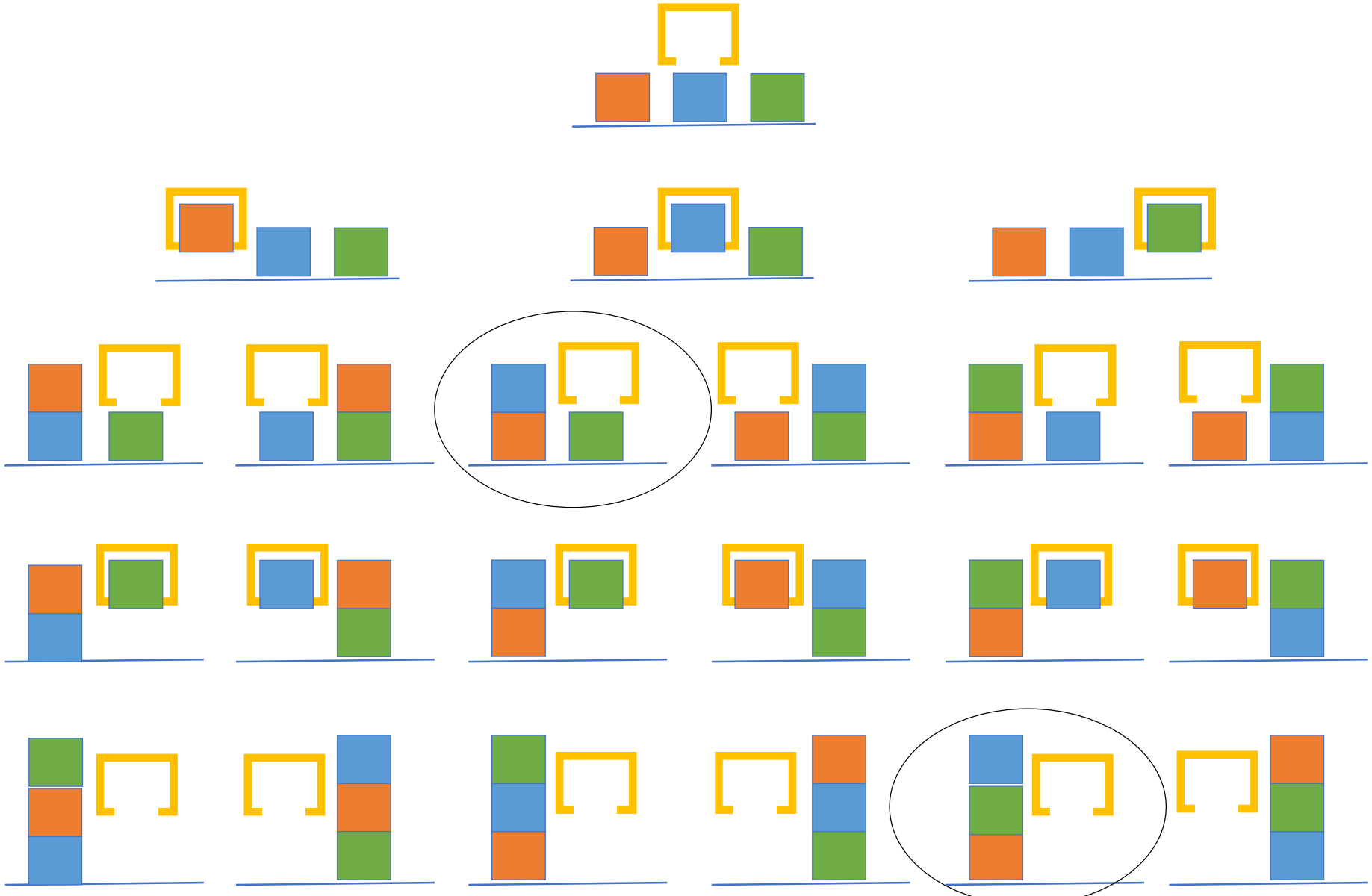


Block Stacking States

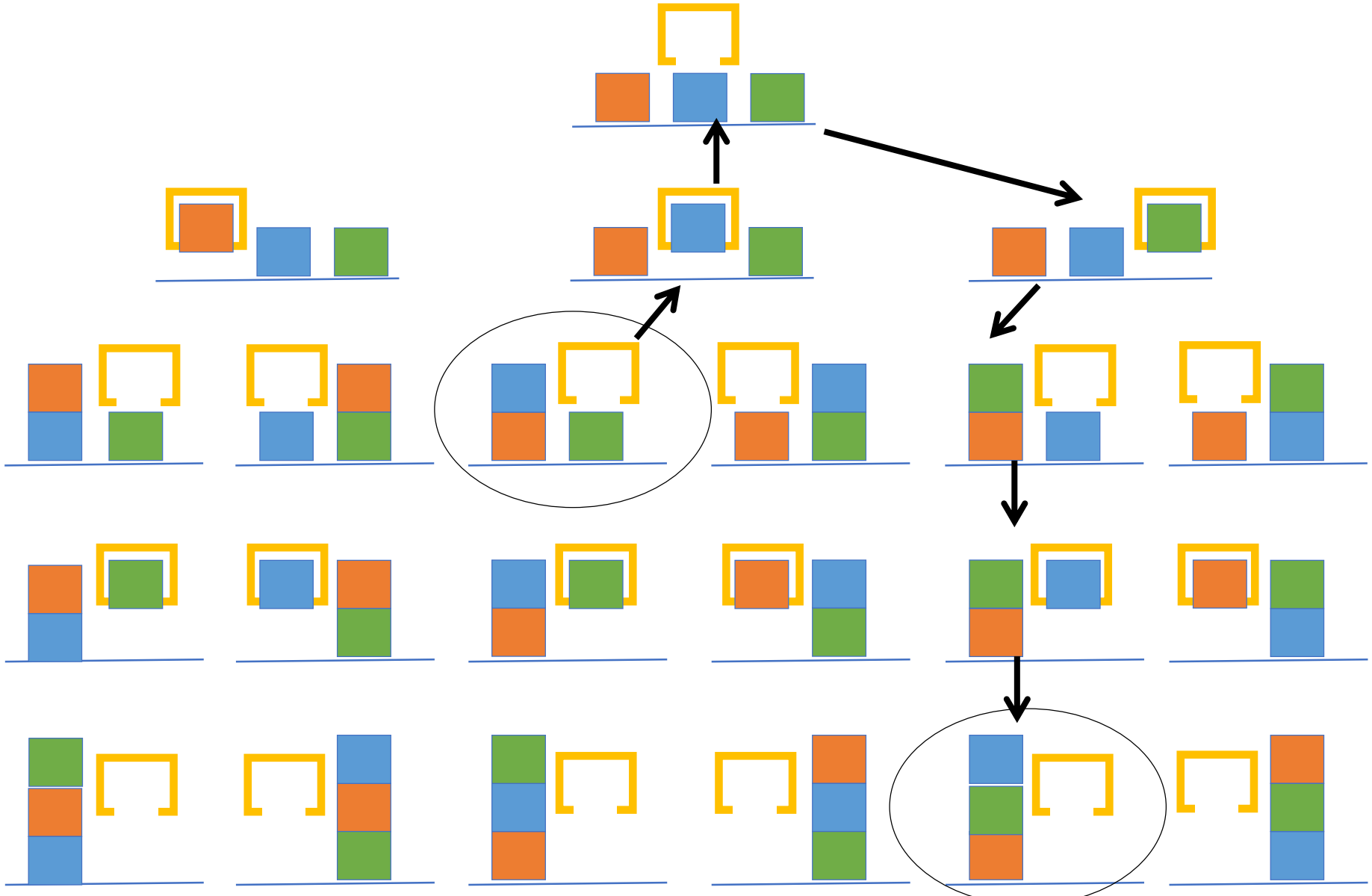
Block Stacking States



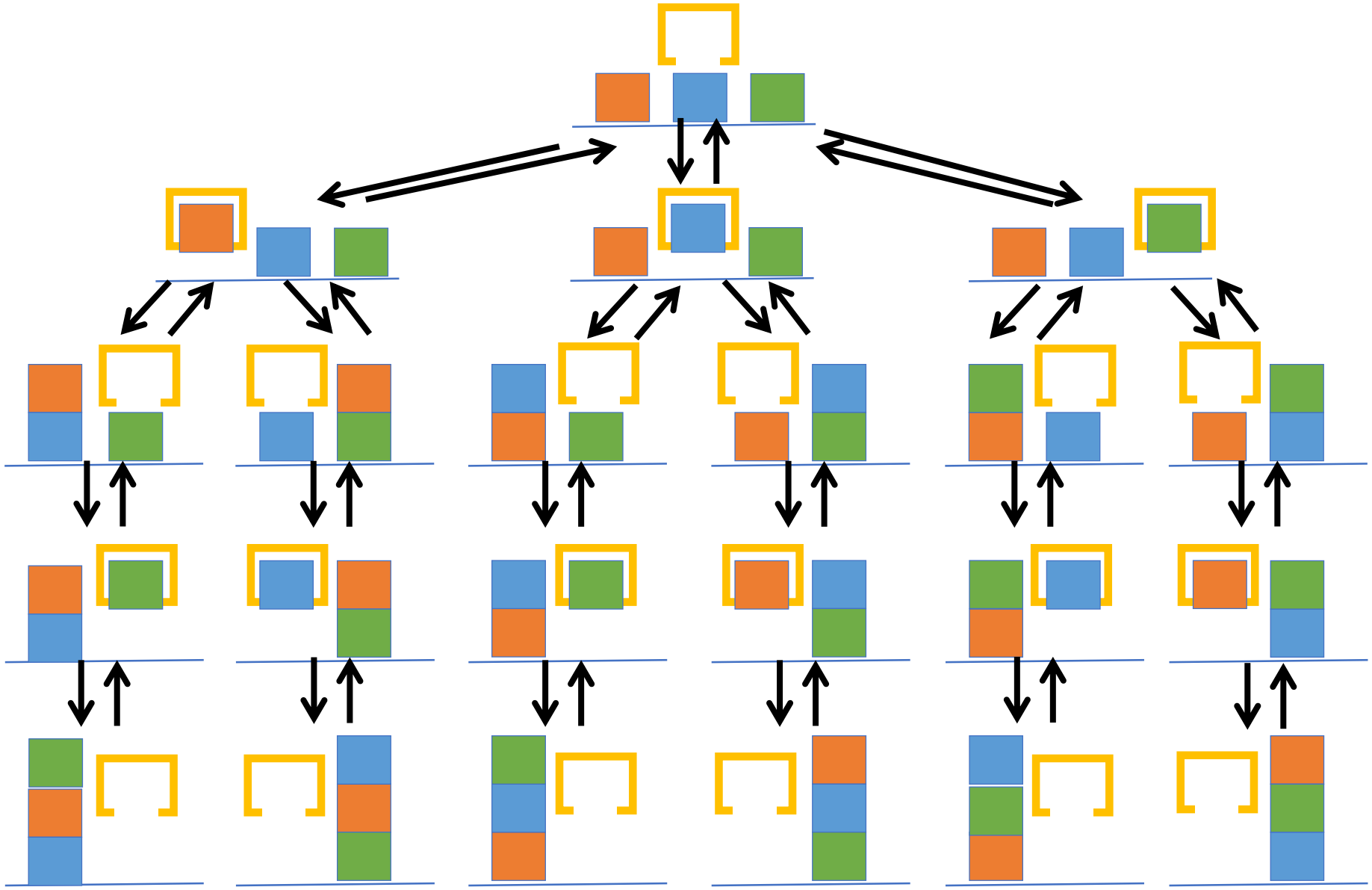
Initial and Goal States



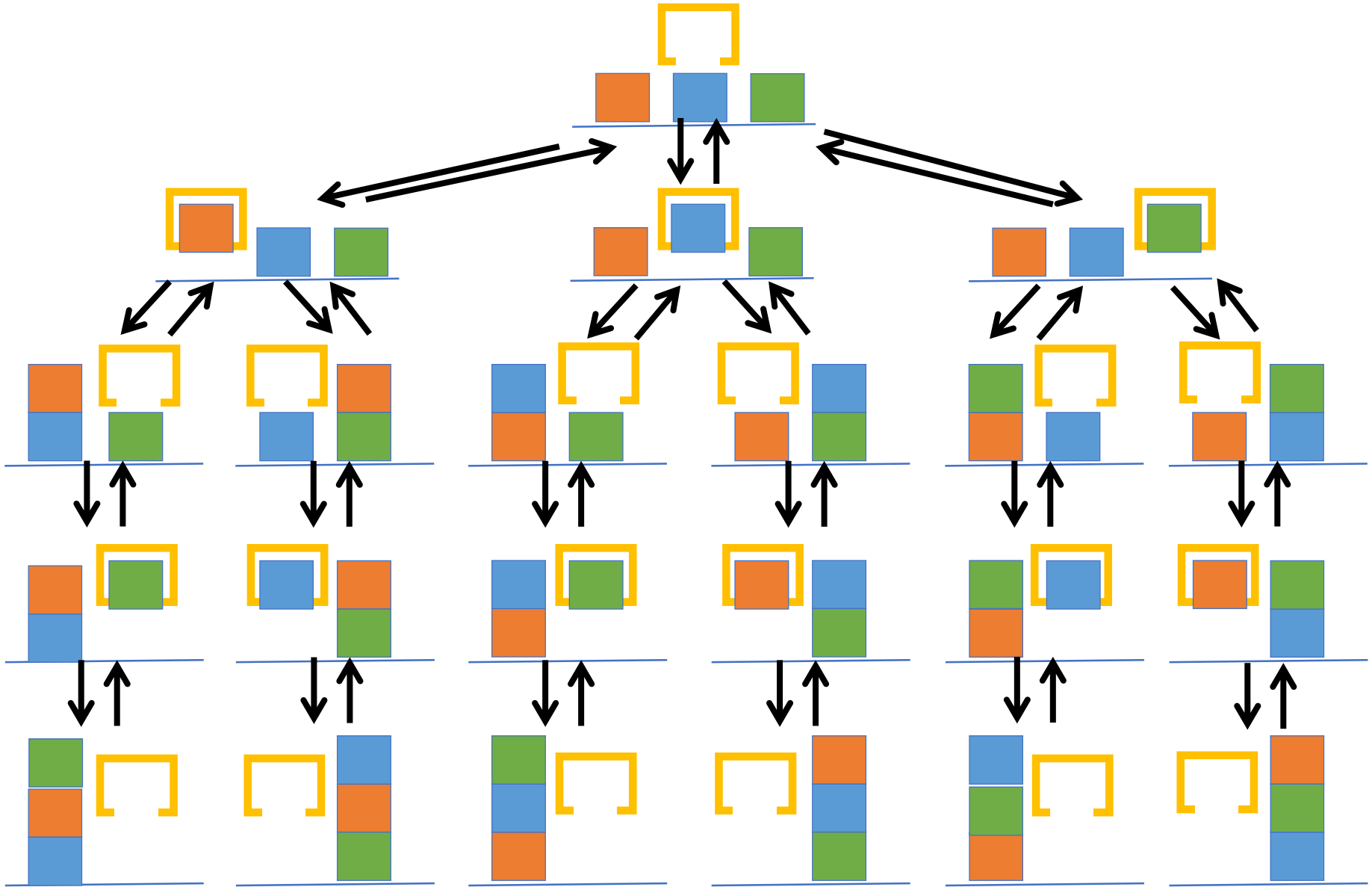
Plan from Initial to Goal State



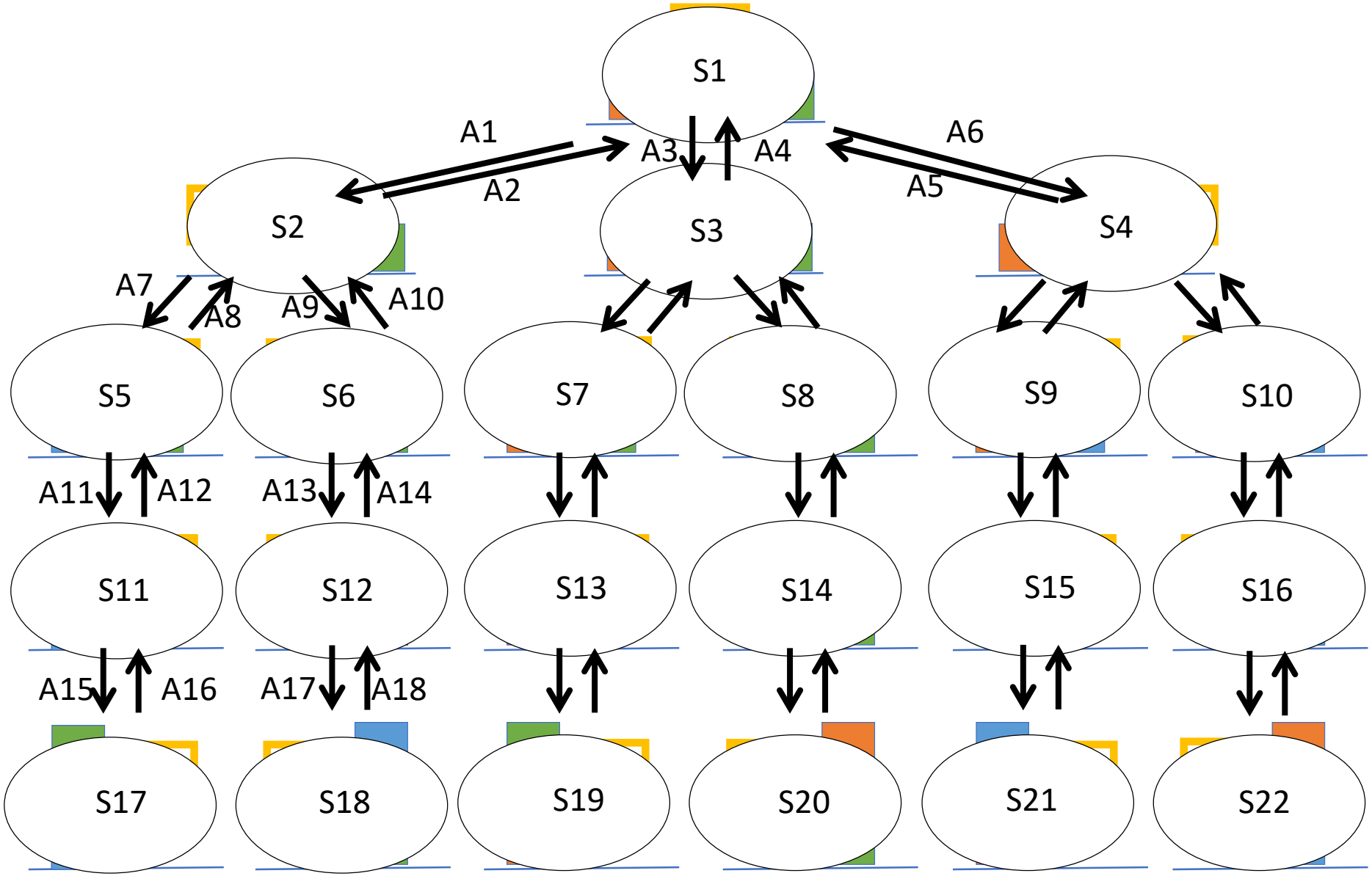
Block Stacking Actions



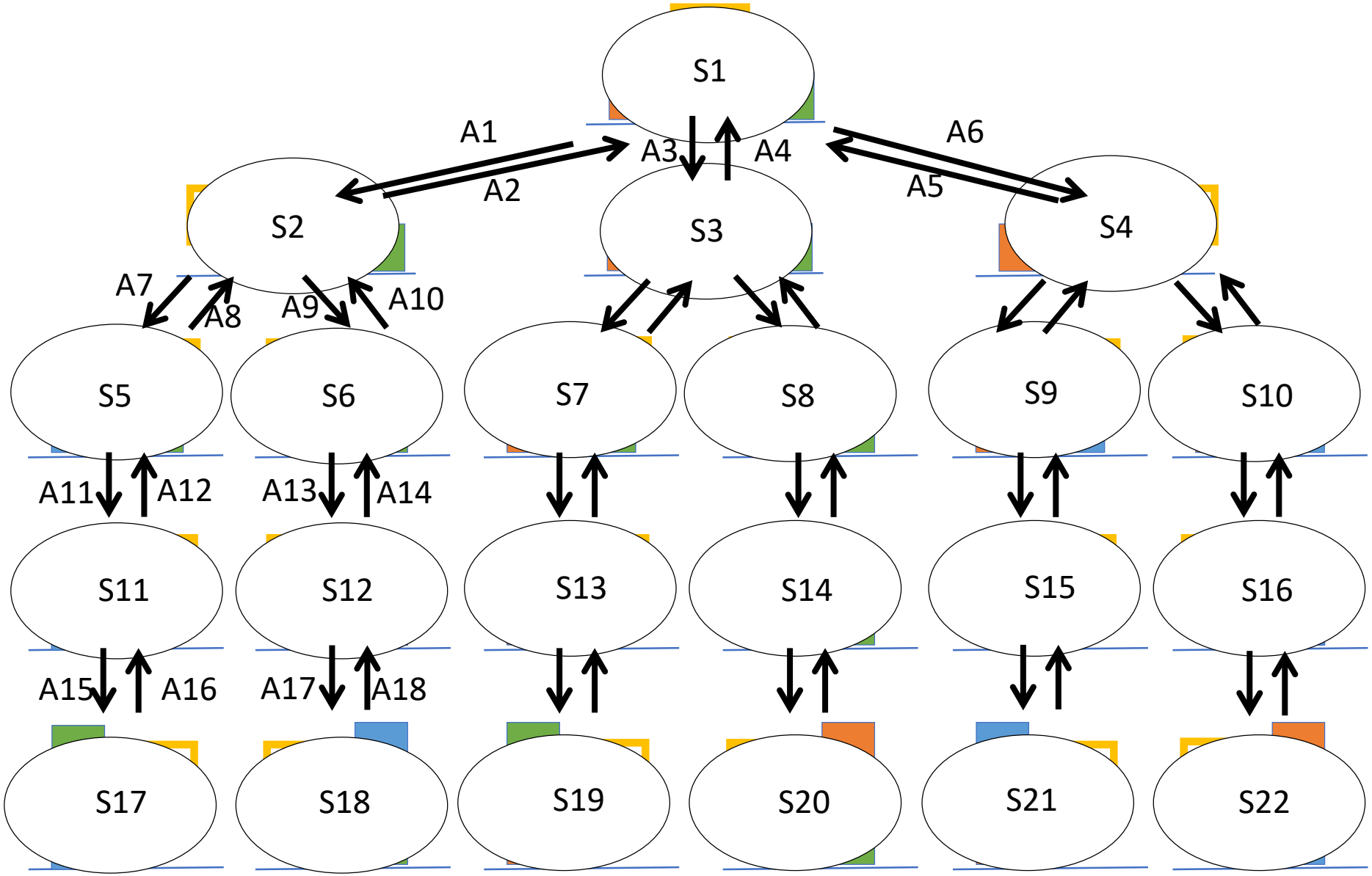
Representation?



States and Actions

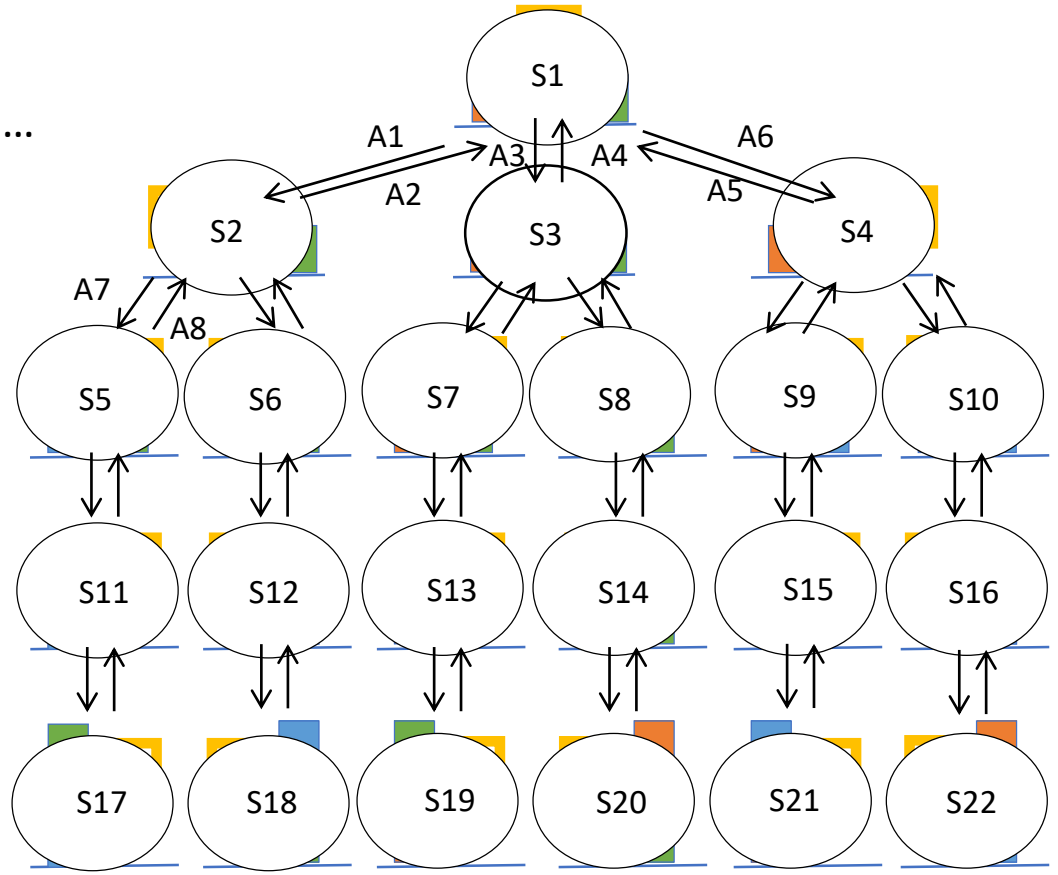


How do we represent this in code?



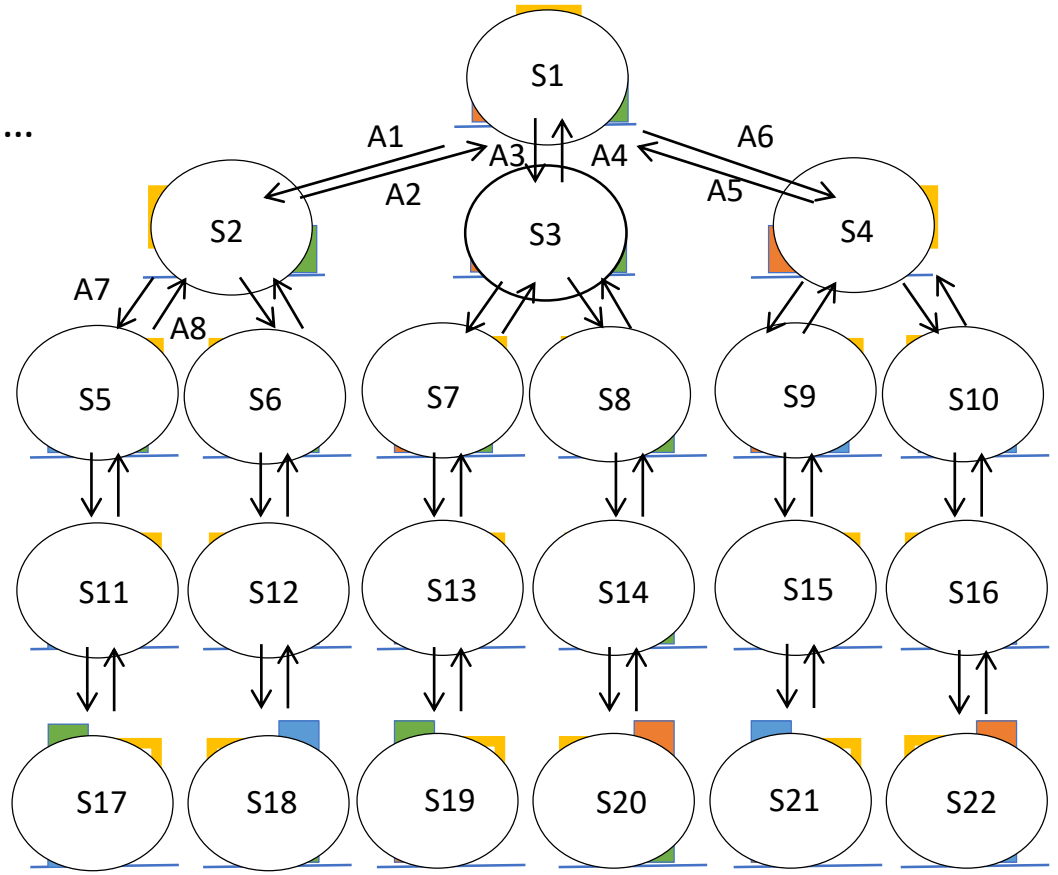
Representing a Graph

	S1	S2	S3	S4	S5	…
A1						
A2						
A3						
A4						
A5						
A6						
A7						
A8						
…						



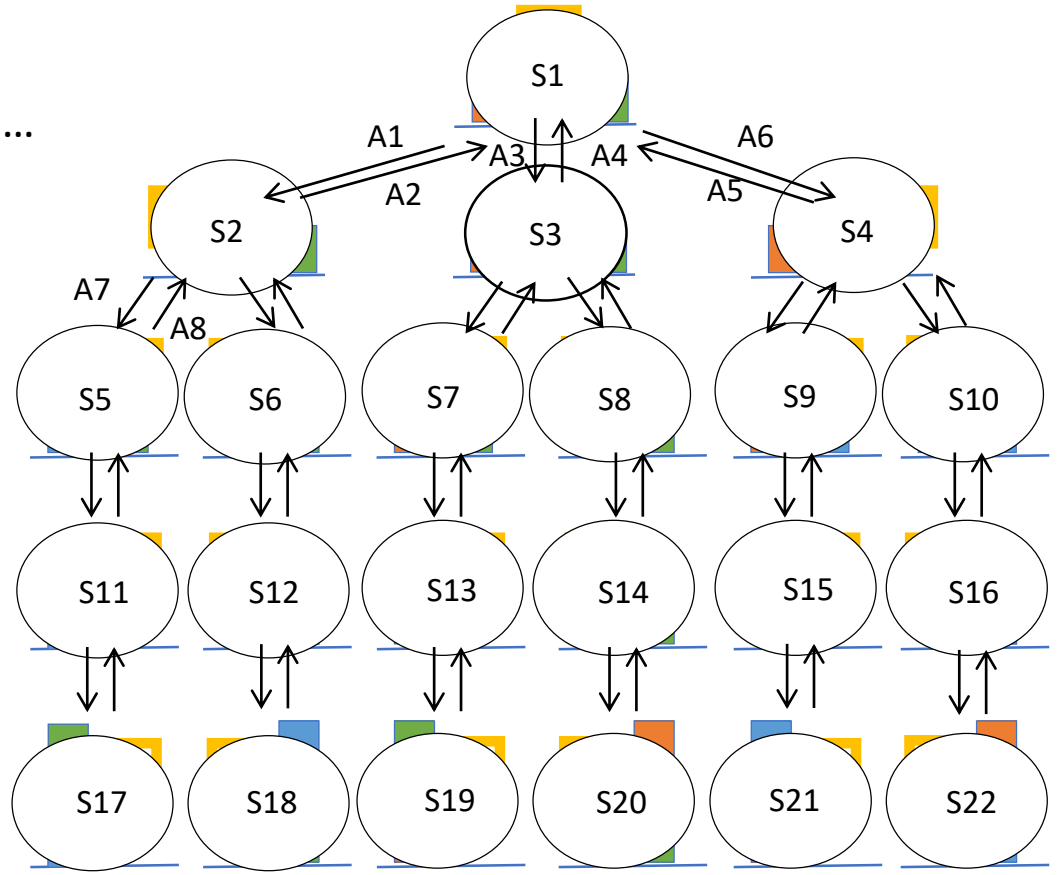
Representing a Graph

	S1	S2	S3	S4	S5	…
A1	S2					
A2	None					
A3	S3					
A4	None					
A5	S5					
A6	None					
A7	None					
A8	None					
…	…					



Representing a Graph

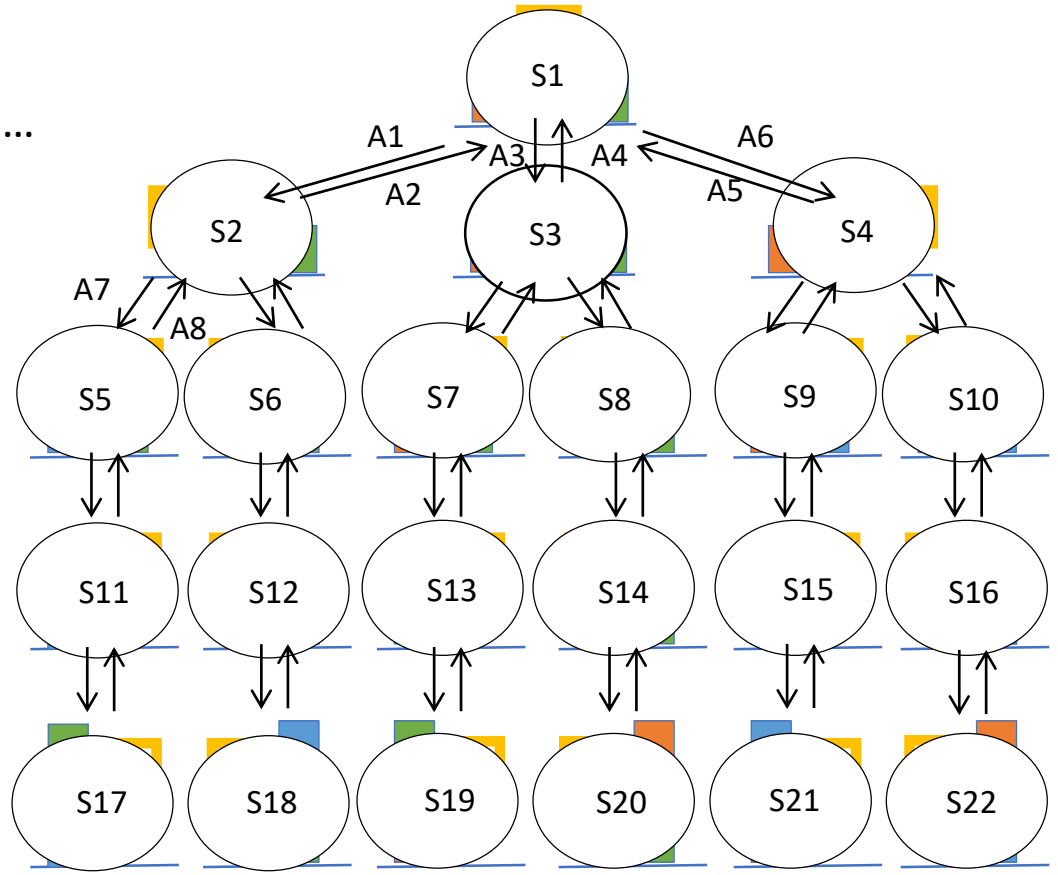
	S1	S2	S3	S4	S5	…
A1	S2	None	None	None	None	
A2	None	S1	None	None	None	
A3	S3	None	None	None	None	
A4	None	None	S1	None	None	
A5	S5	None	None	S1	None	
A6	None	None	None	None	None	
A7	None	S5	None	None	None	
A8	None	None	None	None	S2	
…	…	…	…	…	…	



Representing a Graph

	S1	S2	S3	S4	S5	…
A1	S2	None	None	None	None	
A2	None	S1	None	None	None	
A3	S3	None	None	None	None	
A4	None	None	S1	None	None	
A5	S5	None	None	S1	None	
A6	None	None	None	None	None	
A7	None	S5	None	None	None	
A8	None	None	None	None	S2	
…	…	…	…	…	…	

2D List



Representing a Graph

	S1	S2	S3	S4	S5	…
A1	S2	None	None	None	None	
A2	None	S1	None	None	None	
A3	S3	None	None	None	None	
A4	None	None	S1	None	None	
A5	S5	None	None	S1	None	
A6	None	None	None	None	None	
A7	None	S5	None	None	None	
A8	None	None	None	None	S2	
…	…	…	…	…	…	

2D List

S1: [(A1,S2),(A3,S3),(A5,S4)]

Dictionary

Representing a Graph

	S1	S2	S3	S4	S5	…
A1	S2	None	None	None	None	
A2	None	S1	None	None	None	
A3	S3	None	None	None	None	
A4	None	None	S1	None	None	
A5	S4	None	None	S1	None	
A6	None	None	None	None	None	
A7	None	S5	None	None	None	
A8	None	None	None	None	S2	
…	…	…	…	…	…	

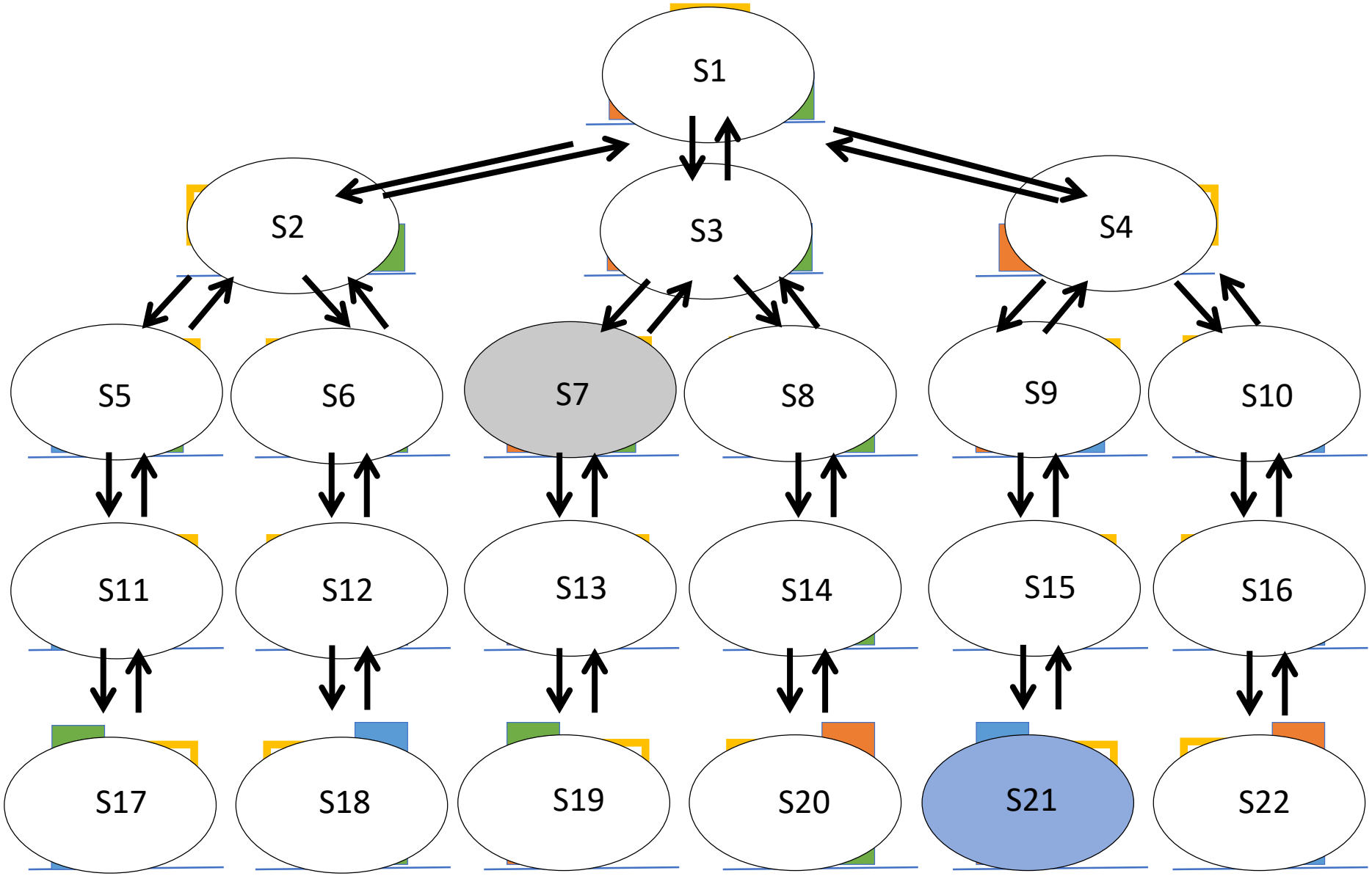
2D List

```
S1: [(A1,S2),(A3,S3),(A5,S4)]
S2: [(A2,S1),(A7,S5)]
S3: [(A4,S1),(A19,S7)]
S4: [(A5,S1),(A31,S9)]
S5: [(A8,S2),(A11,S11)]
…
```

Dictionary

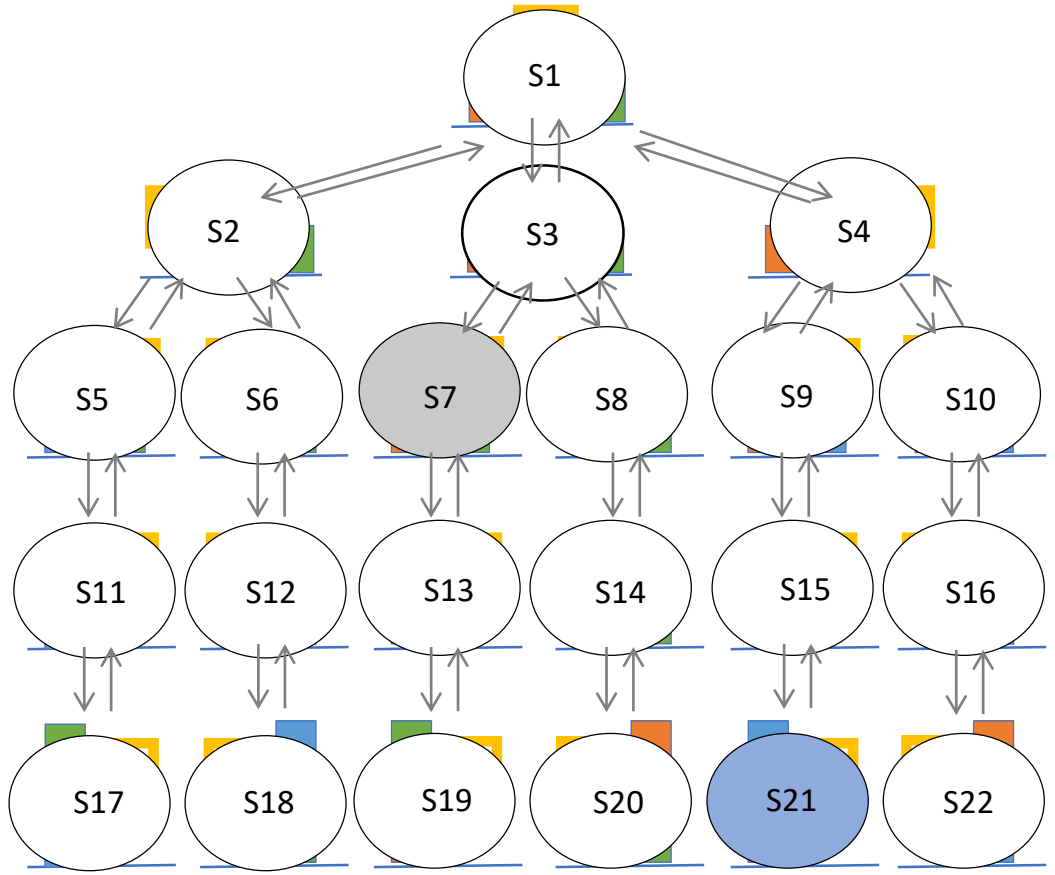
Common Research/Development Paradigm

How do we find the action plan?



Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

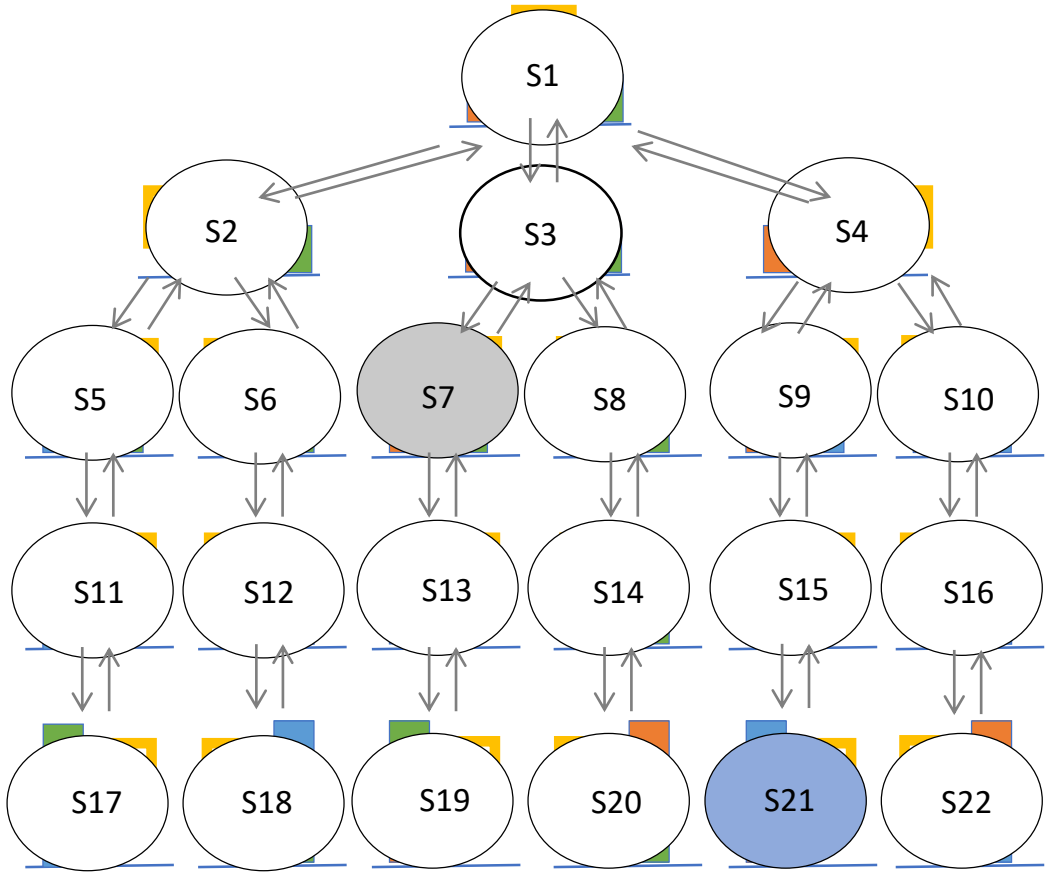


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

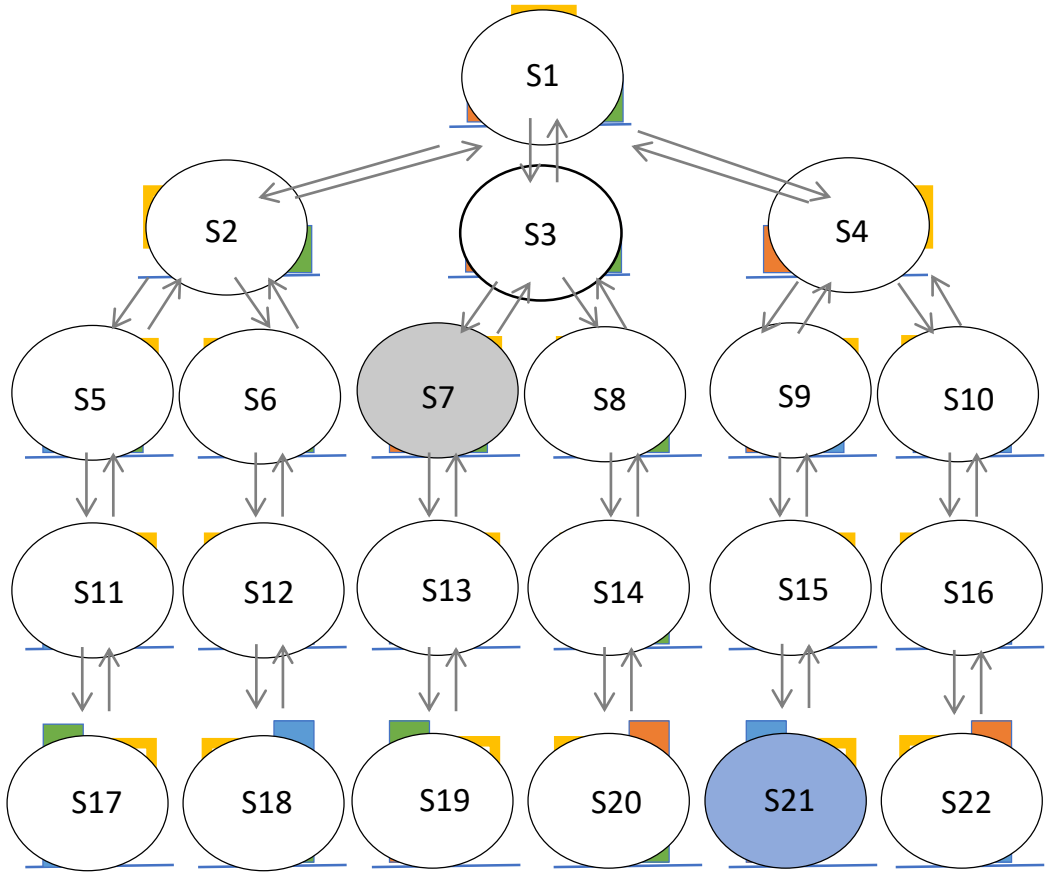


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

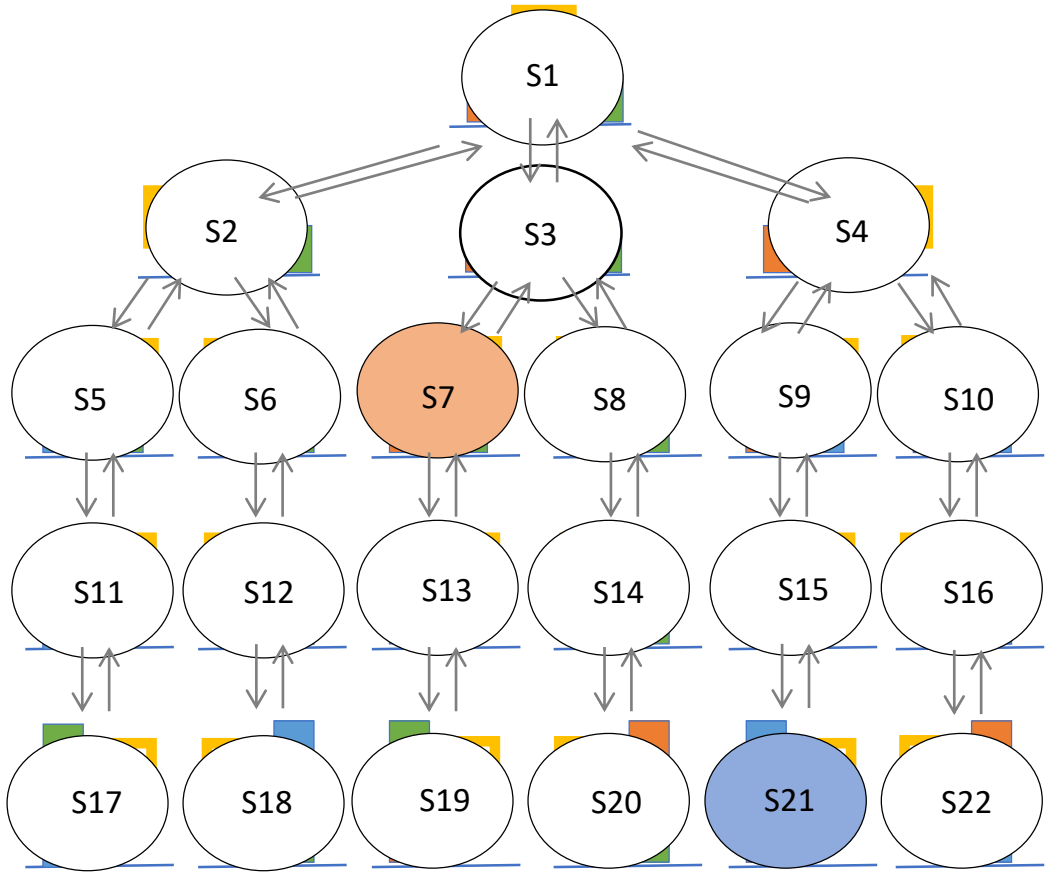


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

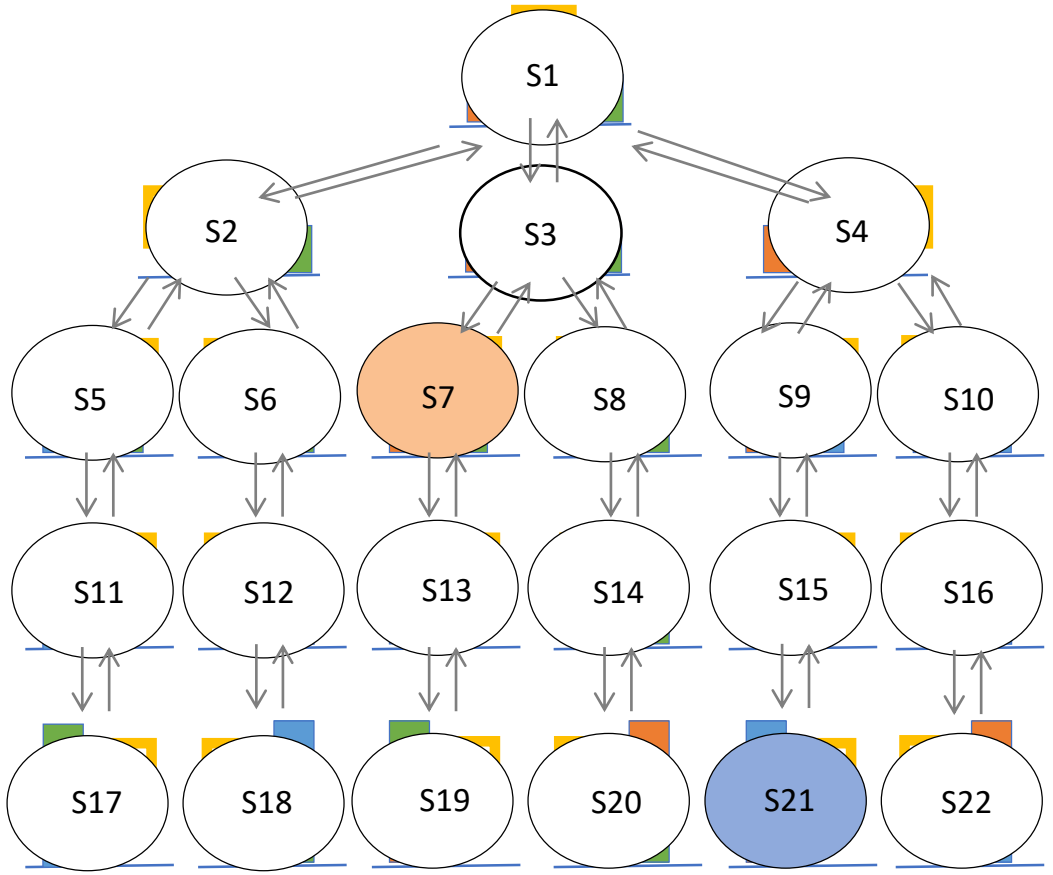


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

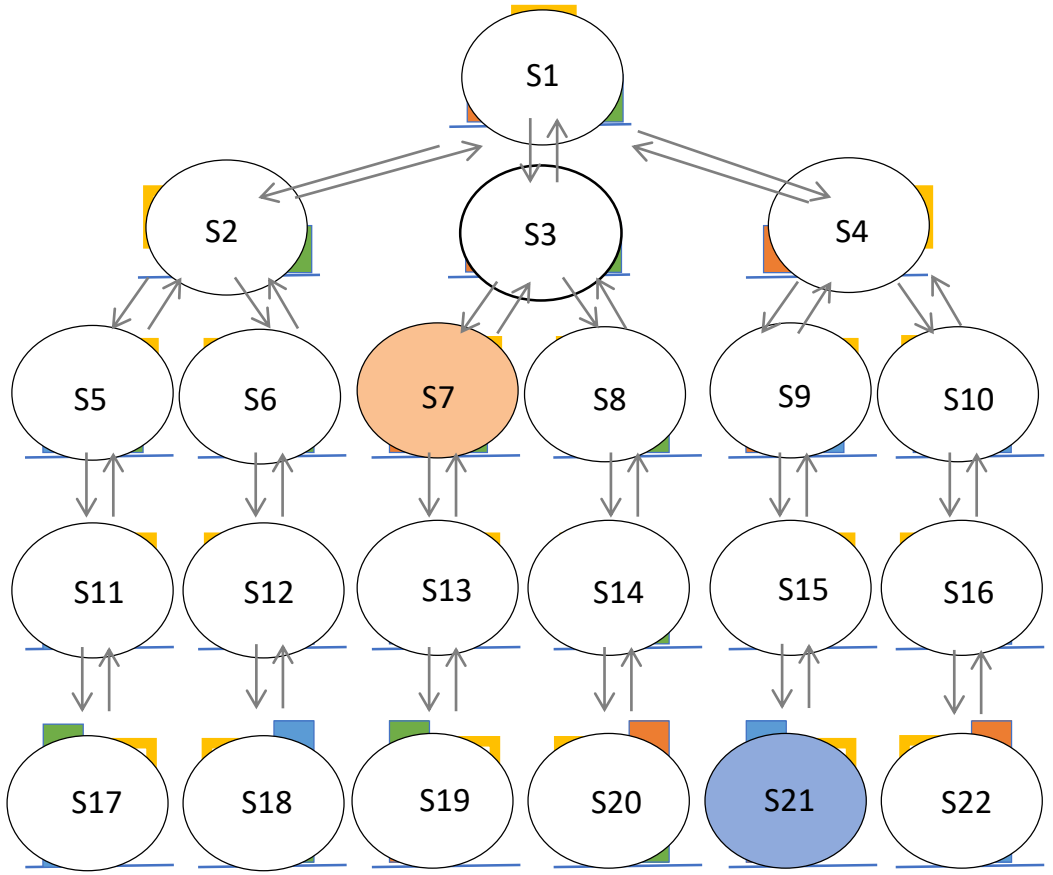


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

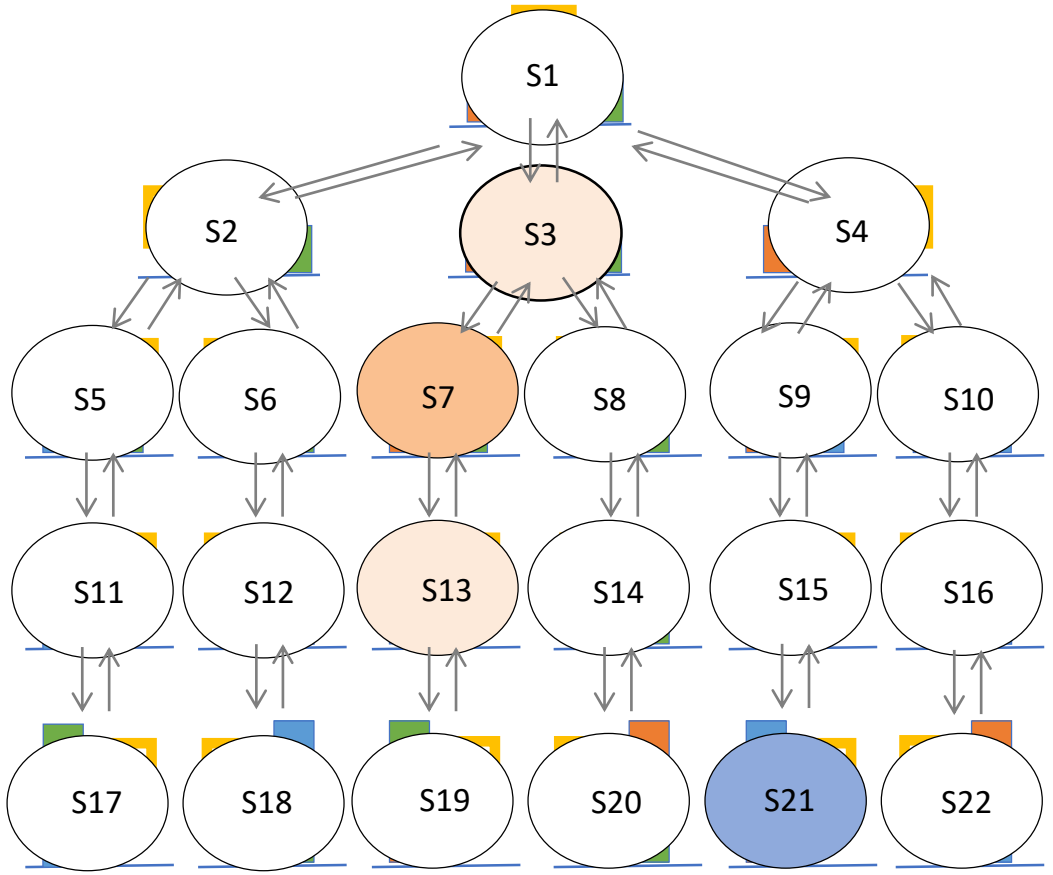


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

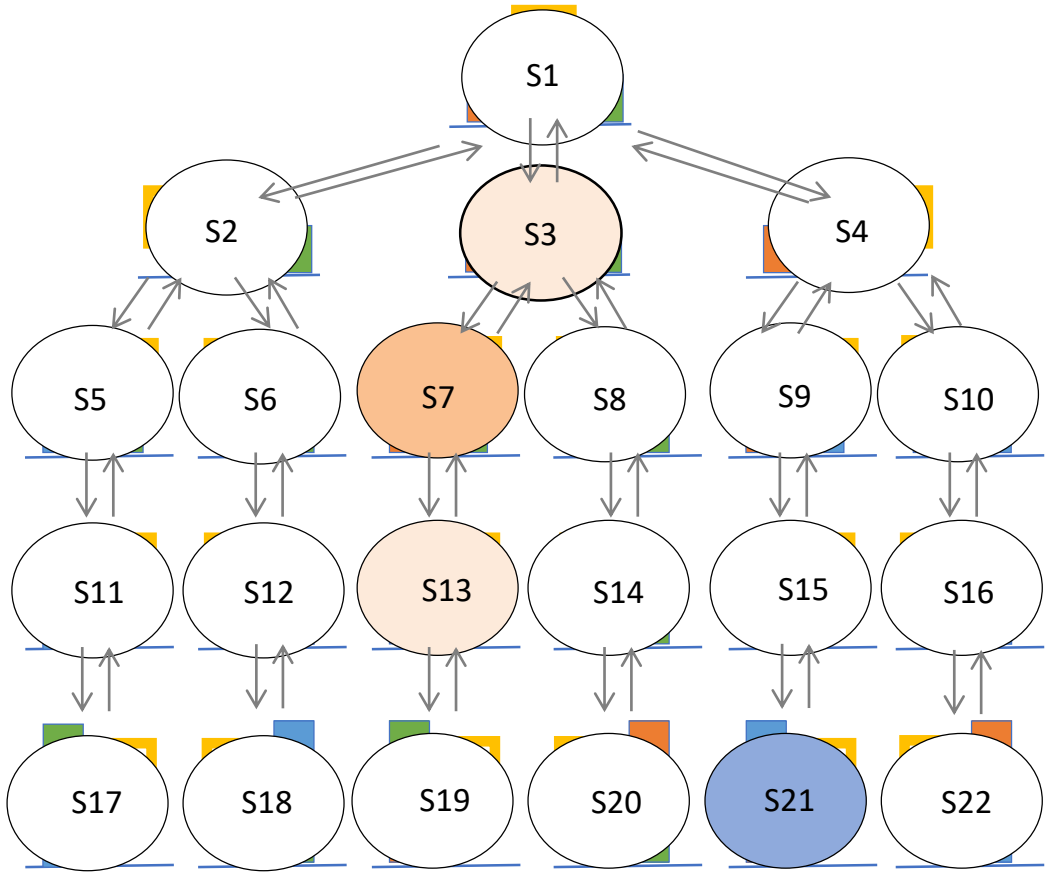


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

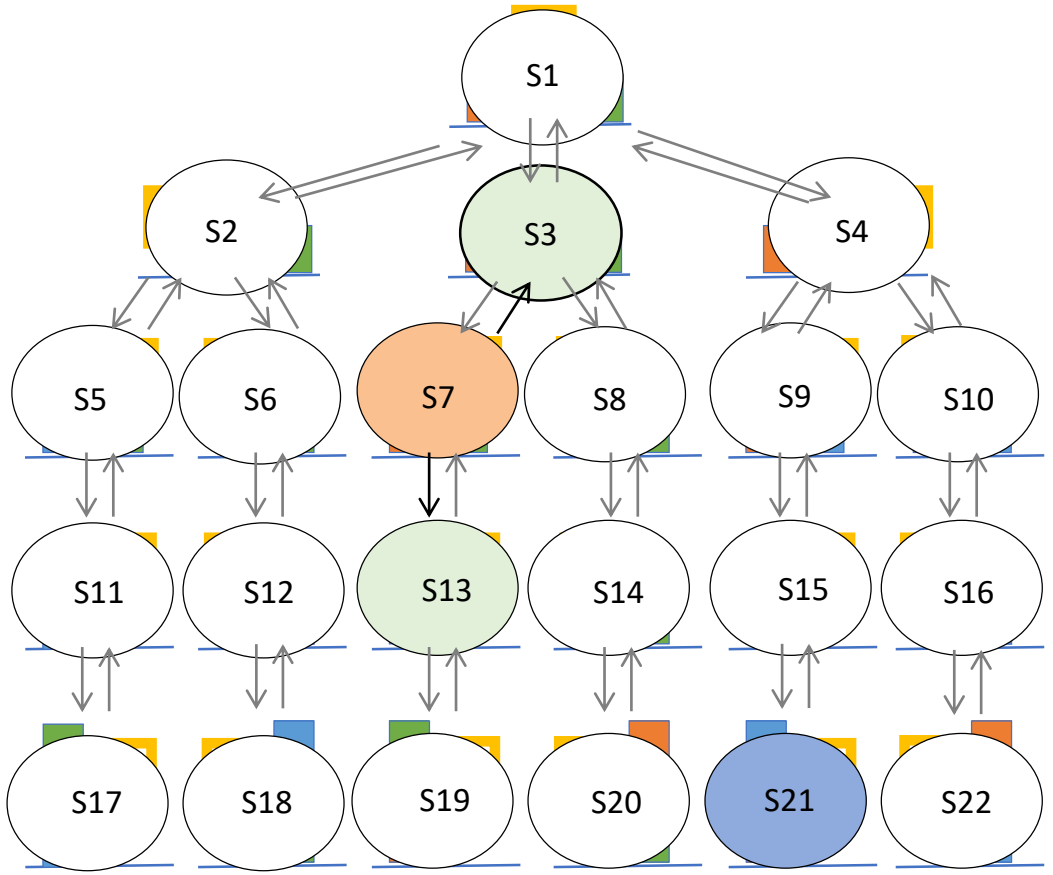
toVisit = []

visited = [S7]



Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

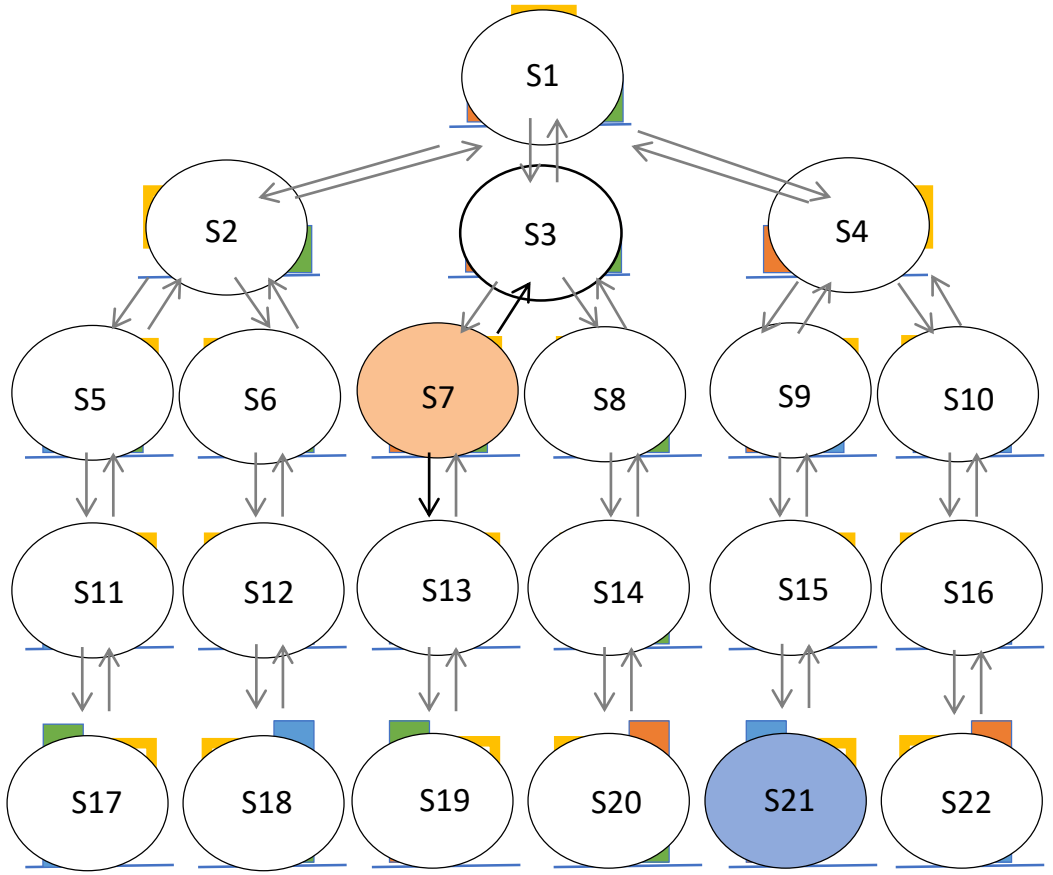


toVisit = [S3,S13]

visited = [S7]

Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```



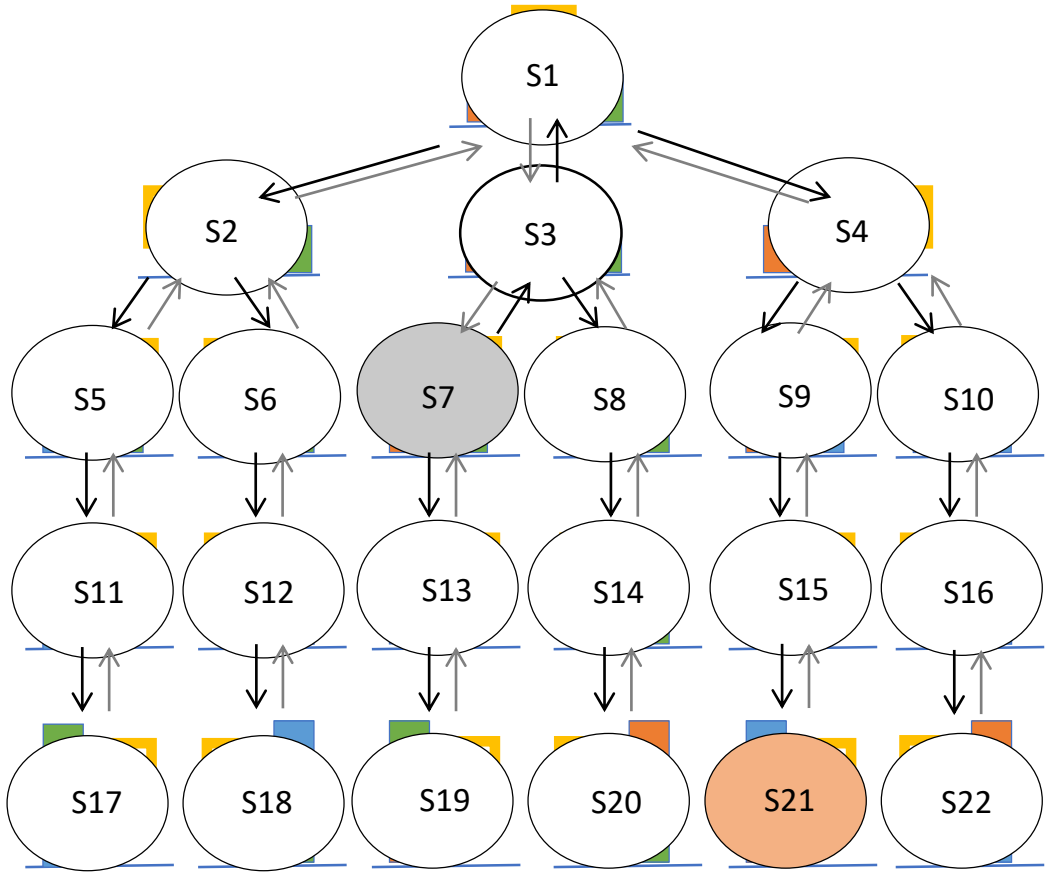
toVisit = [S3,S13]

visited = [S7]

See slides at end of deck for complete BFS

Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

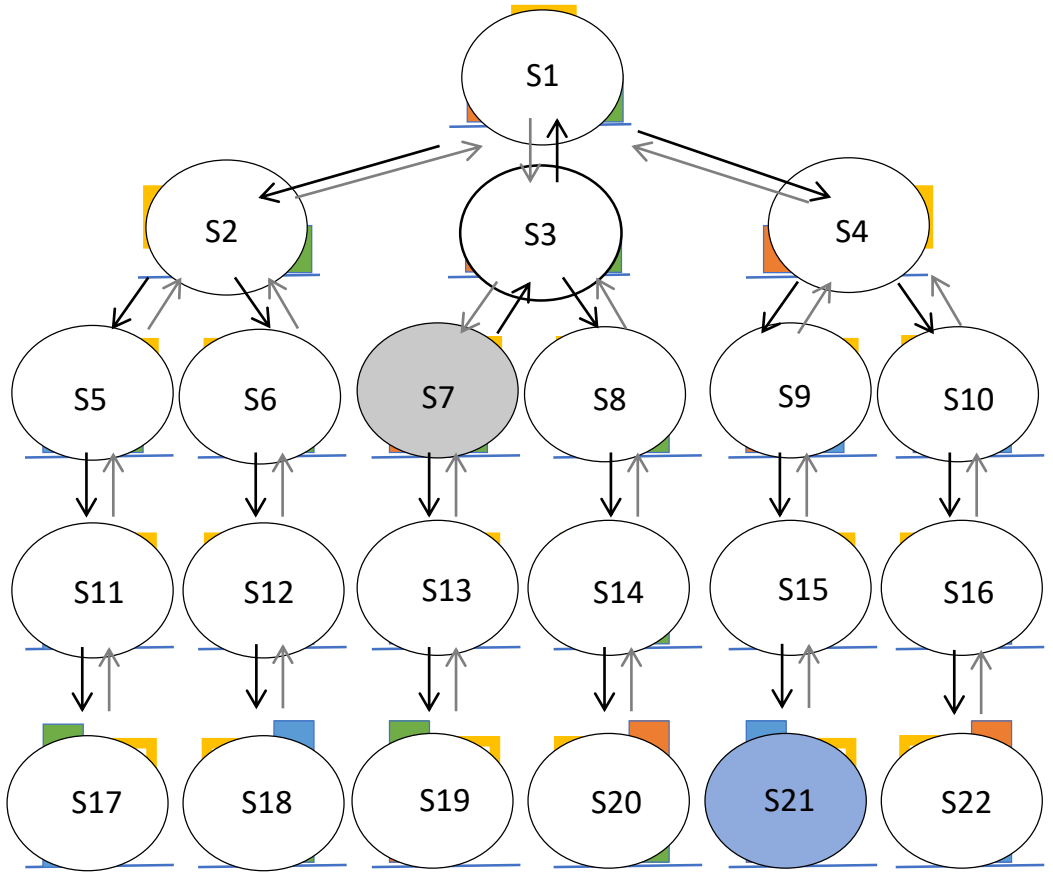


toVisit = [S22]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15,S16,S17,S18,S21]

Searching a Graph: BFS

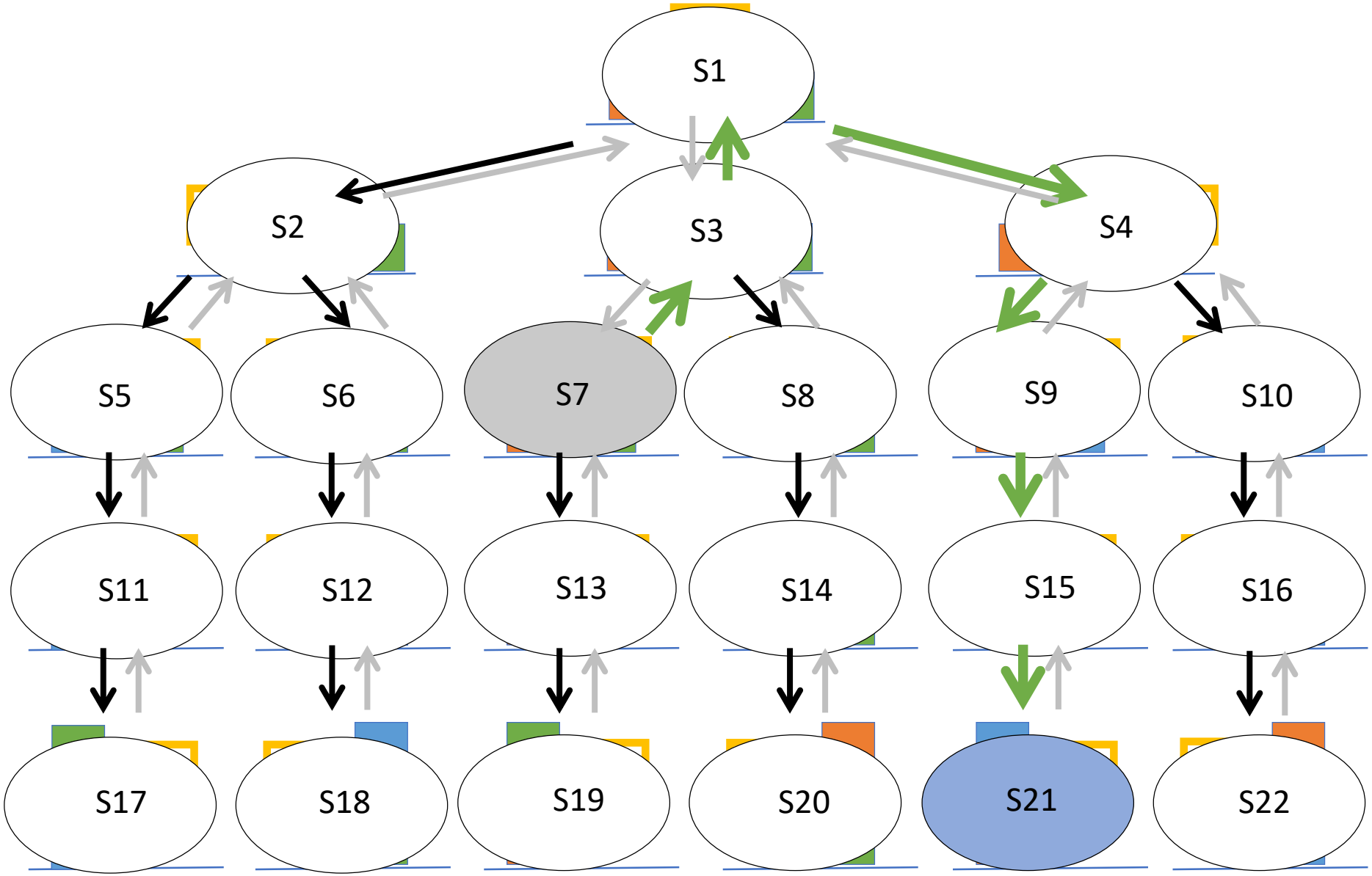
```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```



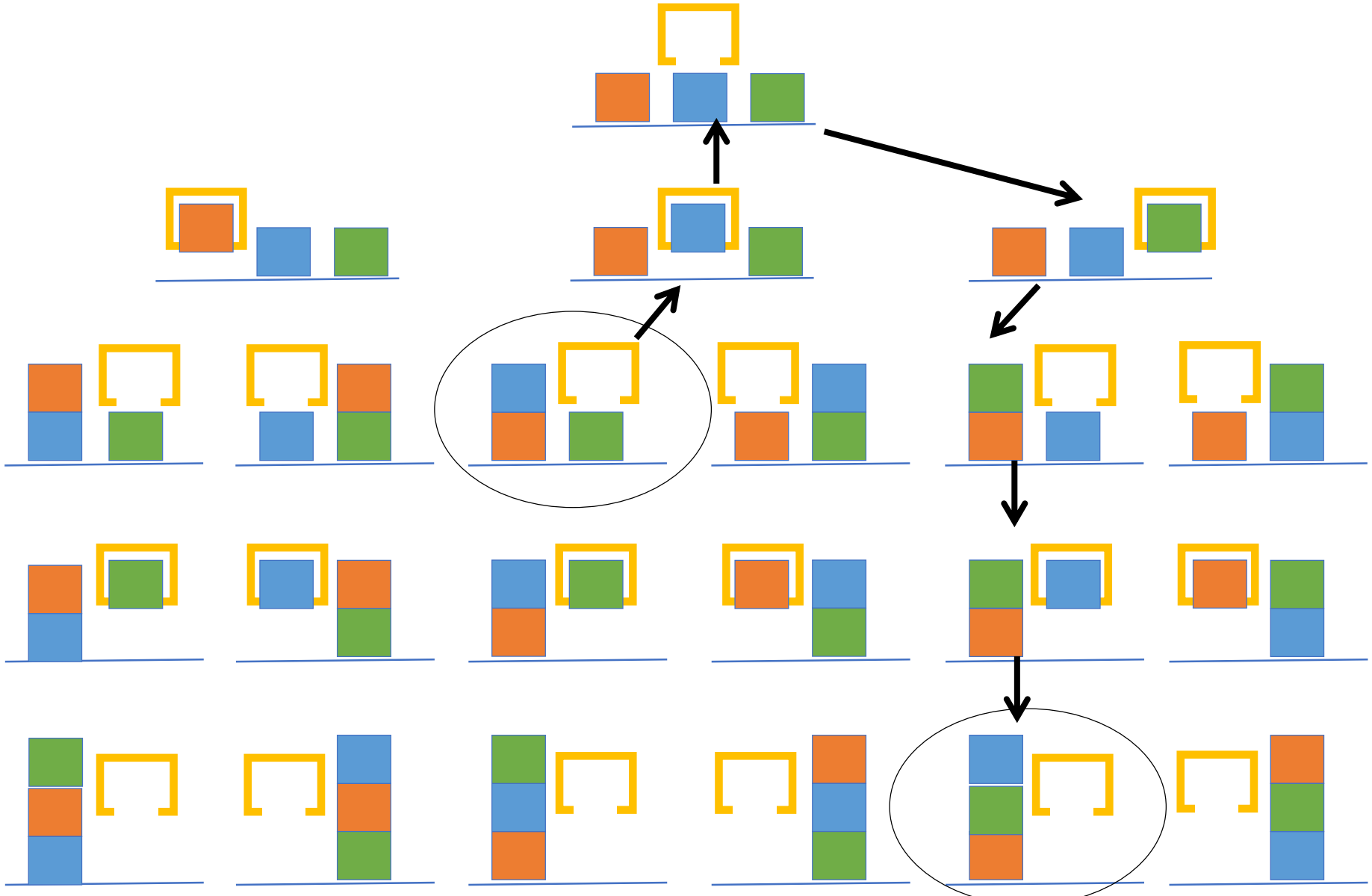
toVisit = [S22]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15,S16,S17,S18,S21]

Follow the Action List



Block Stacking Plan



Informationless States and Actions

What are the pros of this approach?

What are the cons of this approach?

Harder Problems in AI

Harder Problems in AI

Summary

Informationless states and actions

- Easy to program
- Can take a lot of memory
- Hard to change the problem (e.g., adding a block)

Other concerns like uncertainty and errors are unmodeled

- More complex models needed for handling these

Takeaways

A lot of AI is search (and smarter search techniques).

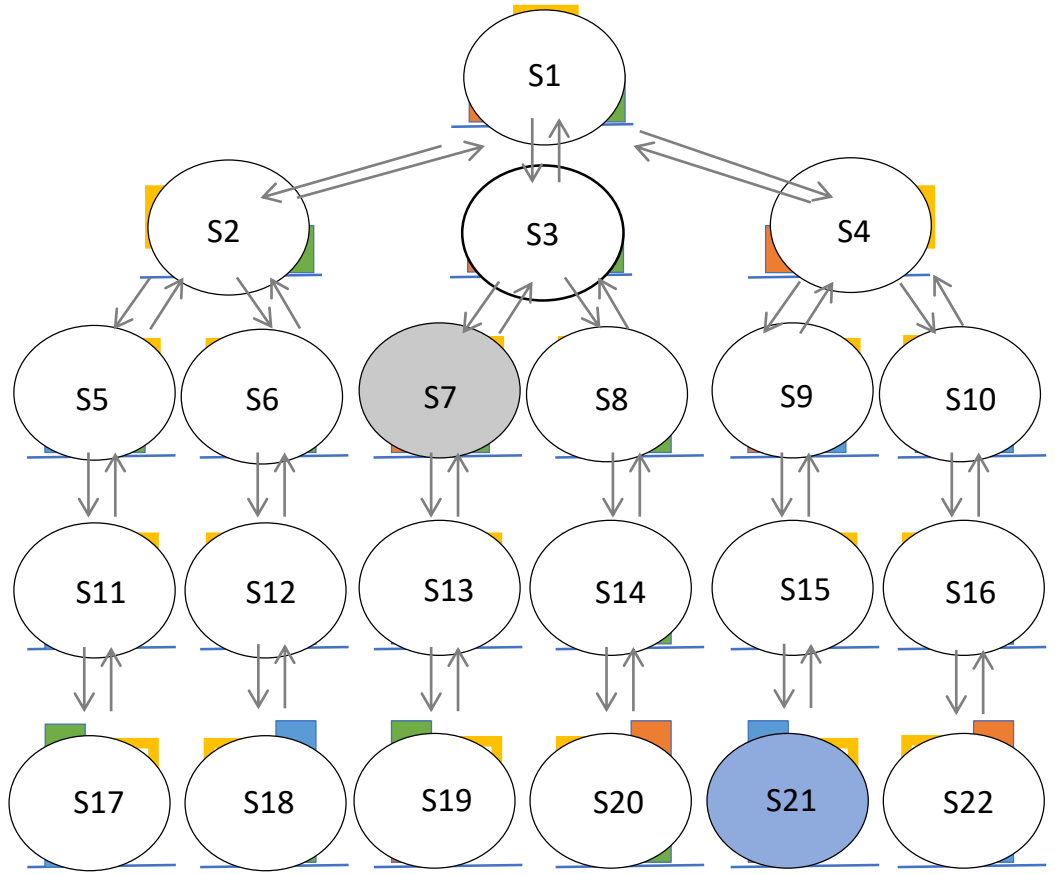
Graphs, BFS and DFS are the starting points for a lot of these techniques.

The challenge to AI is how to represent the problem and then how to solve it efficiently!

BFS Slides

Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

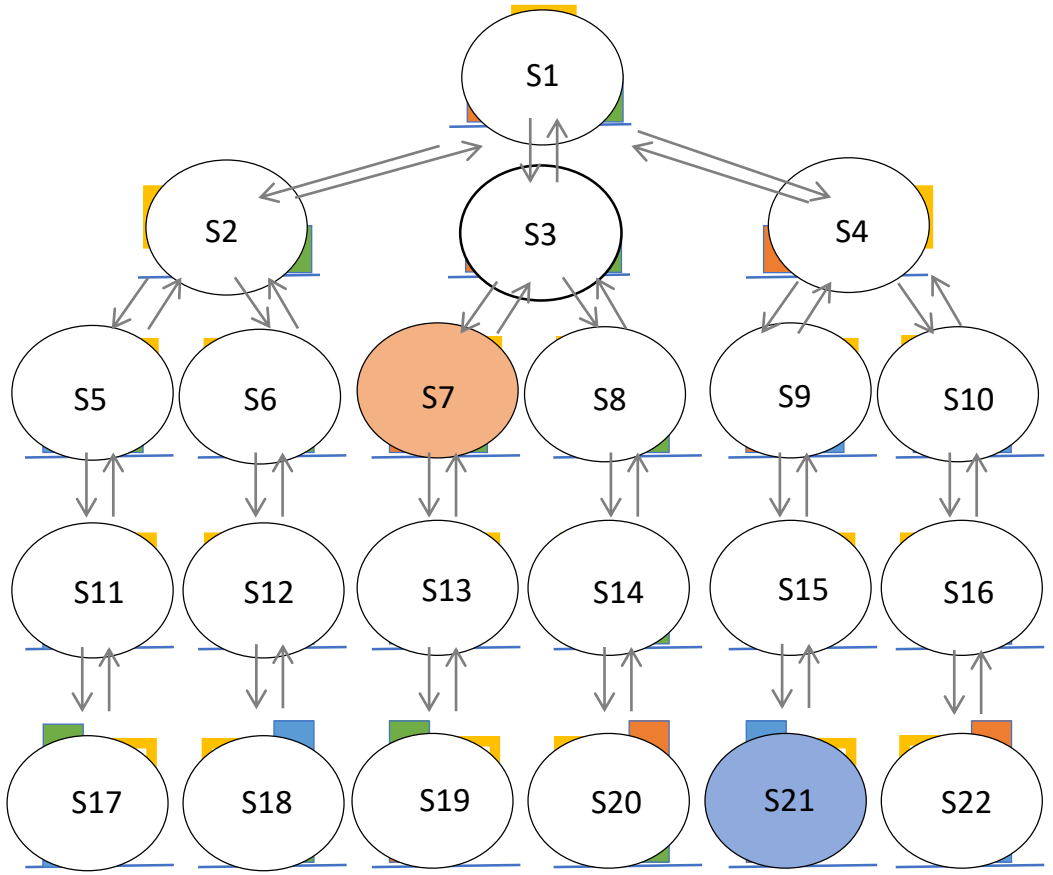


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

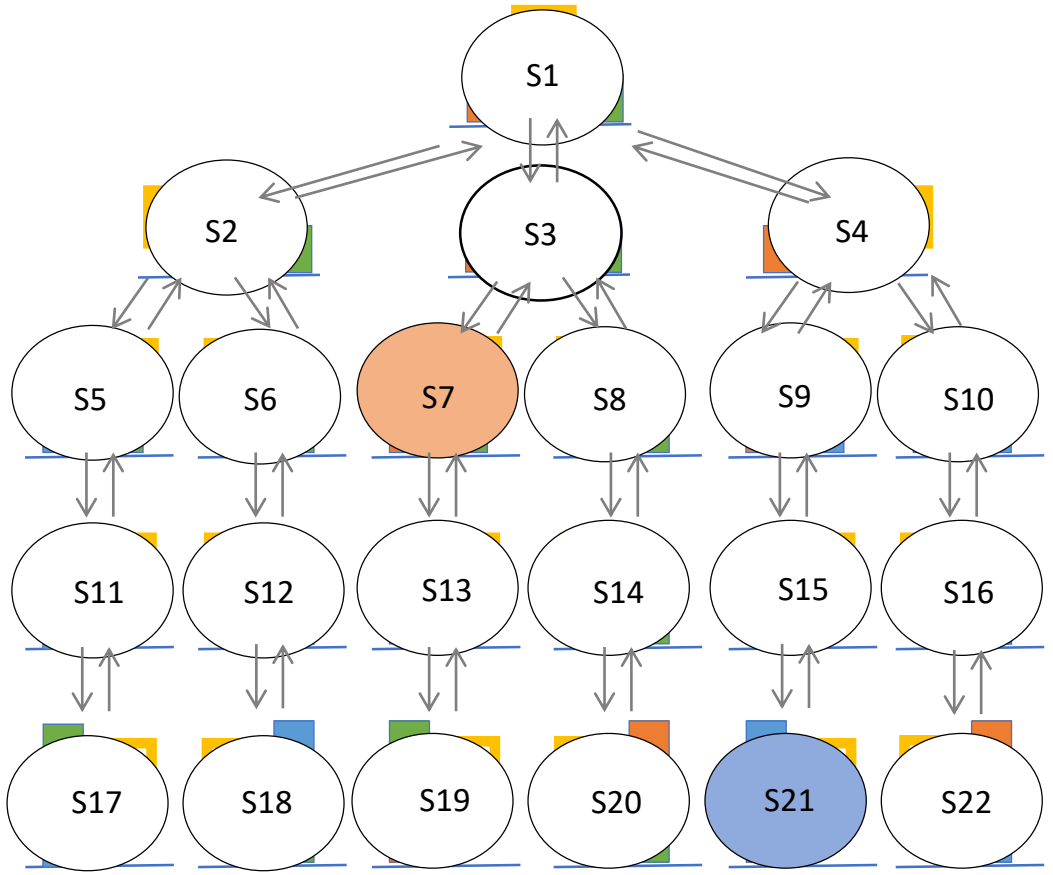


Searching a Graph: BFS

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

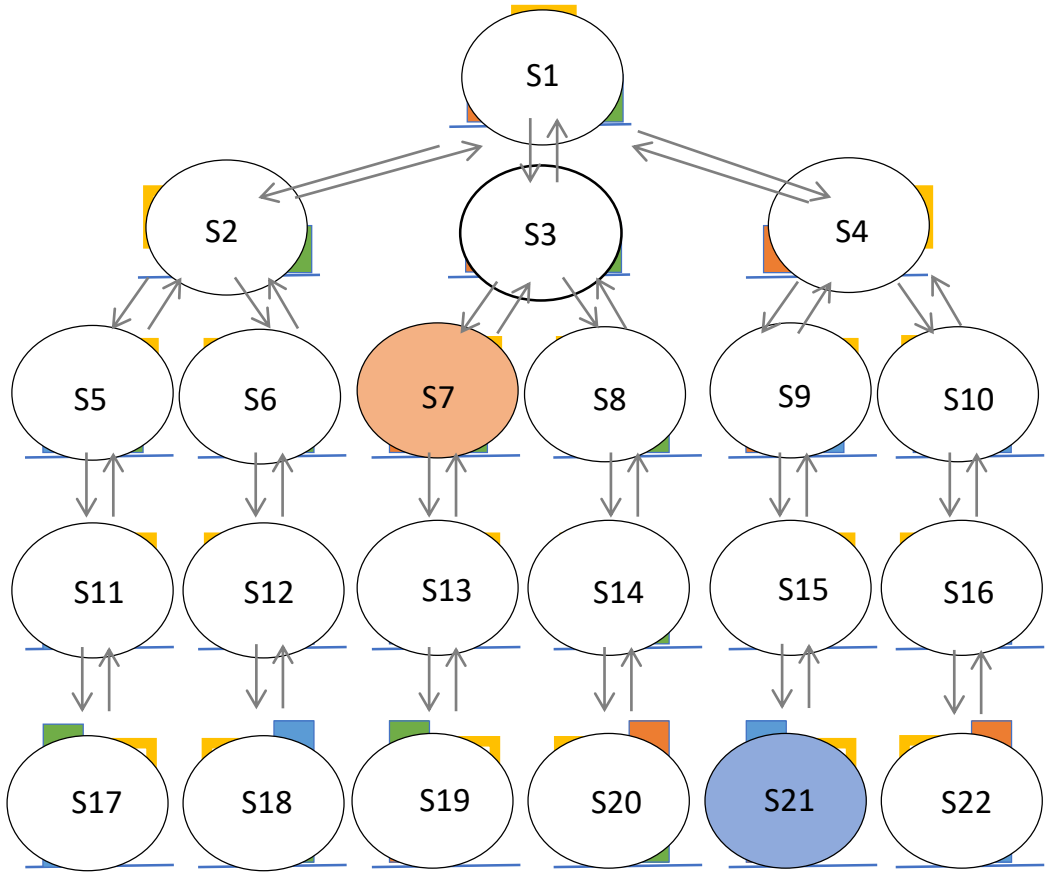


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S7]

visited = []

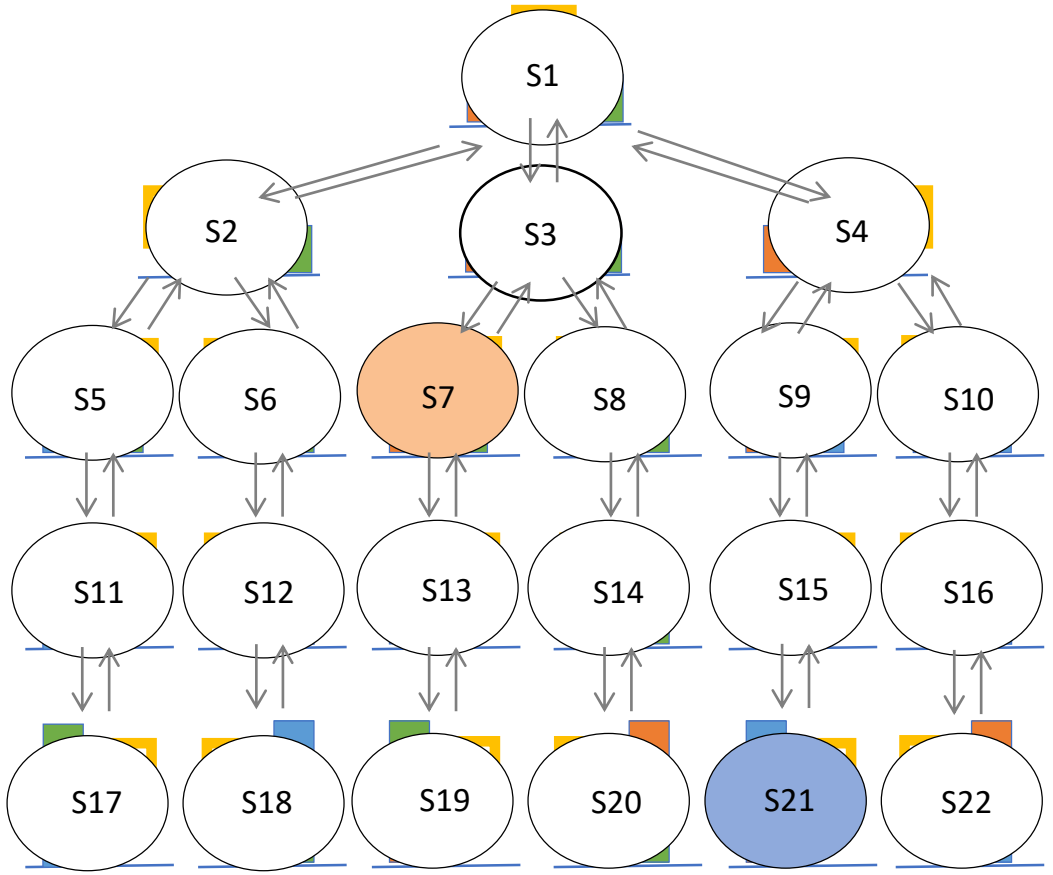


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

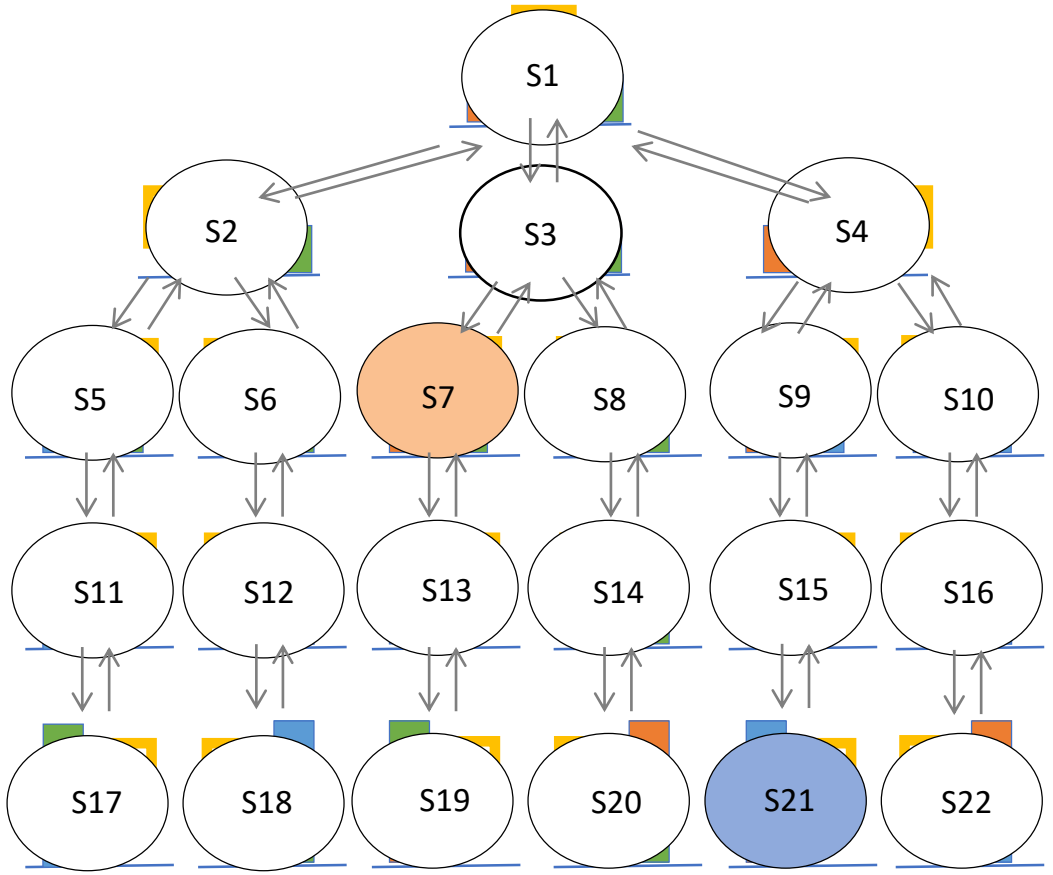


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

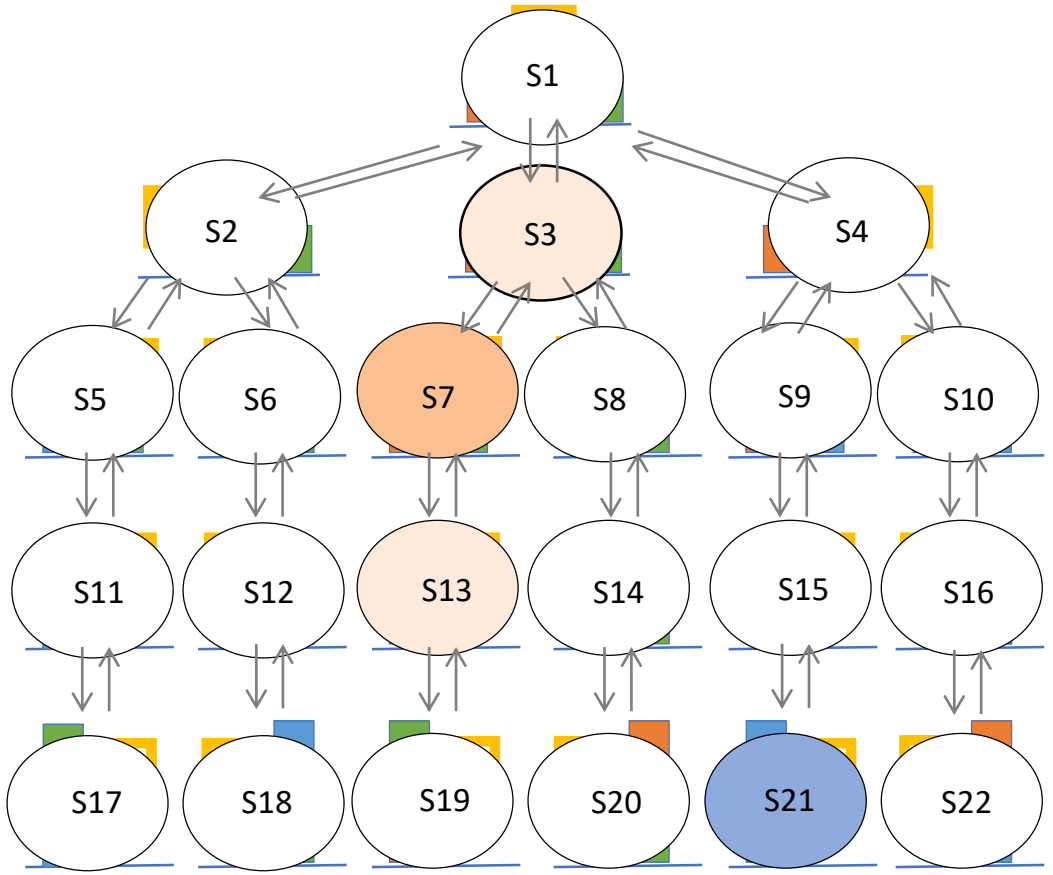


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

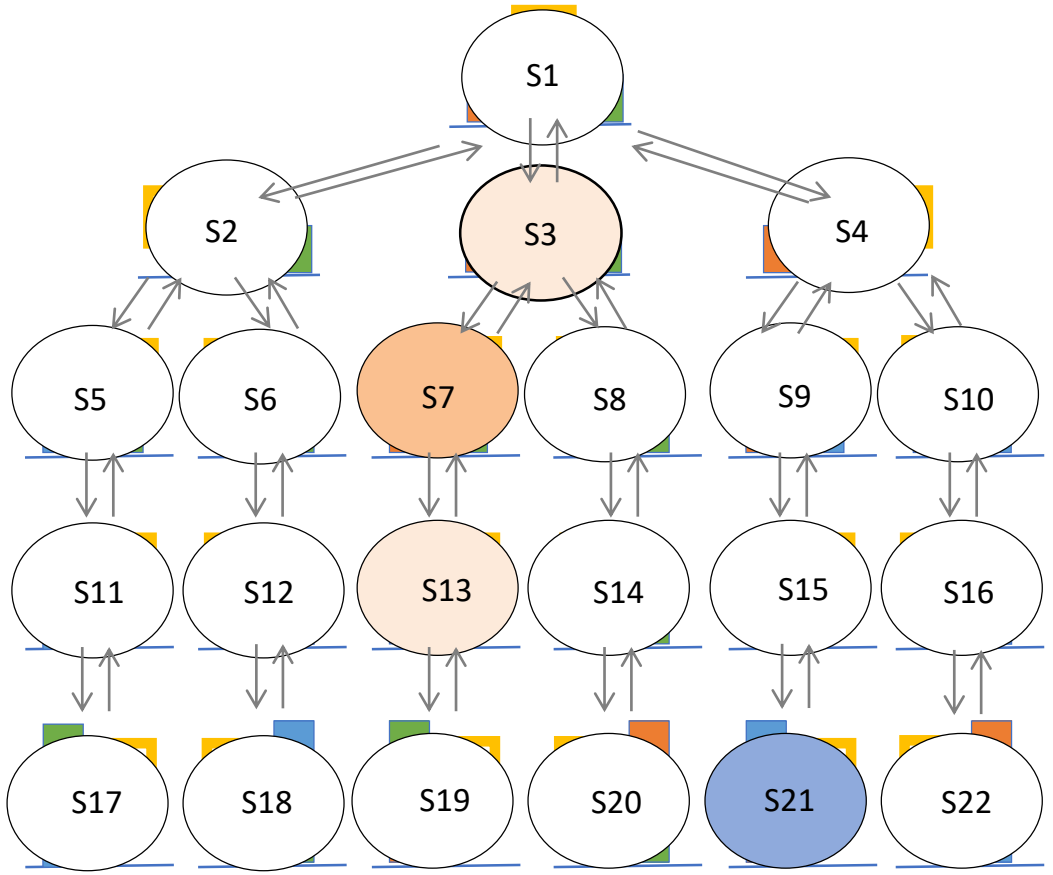


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = []

visited = [S7]

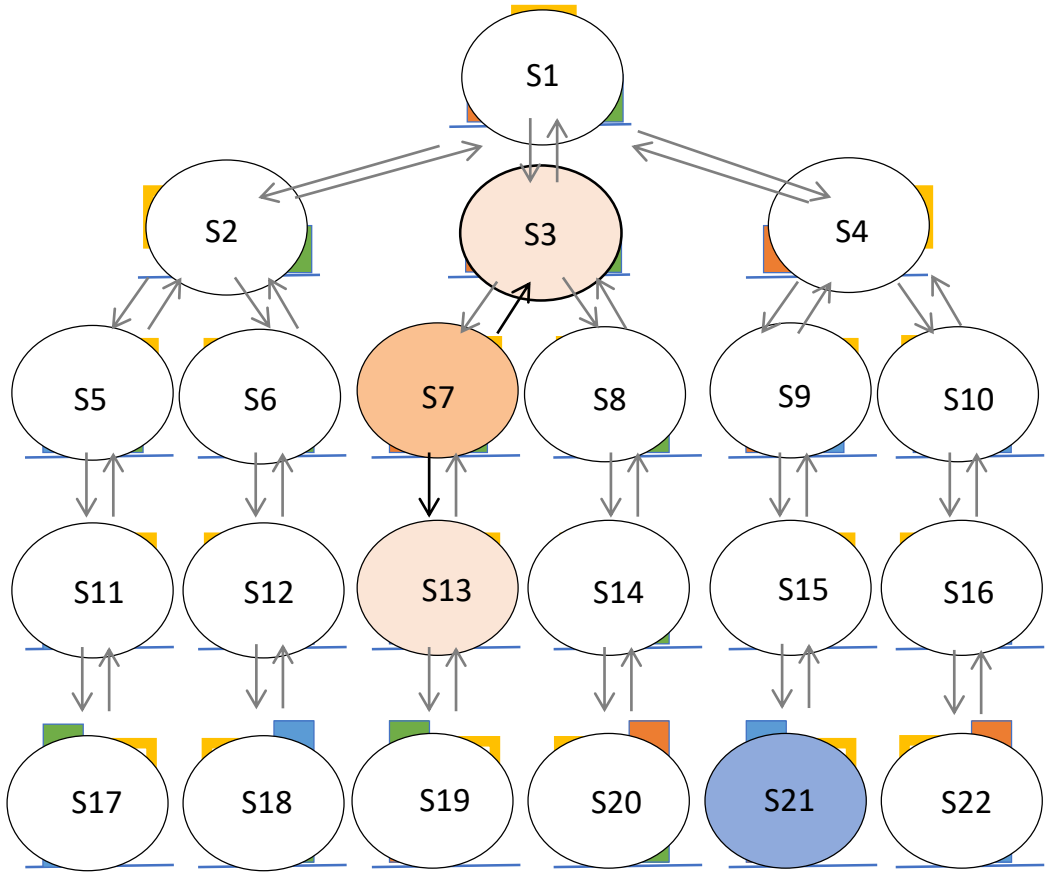


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S3,S13]

visited = [S7]

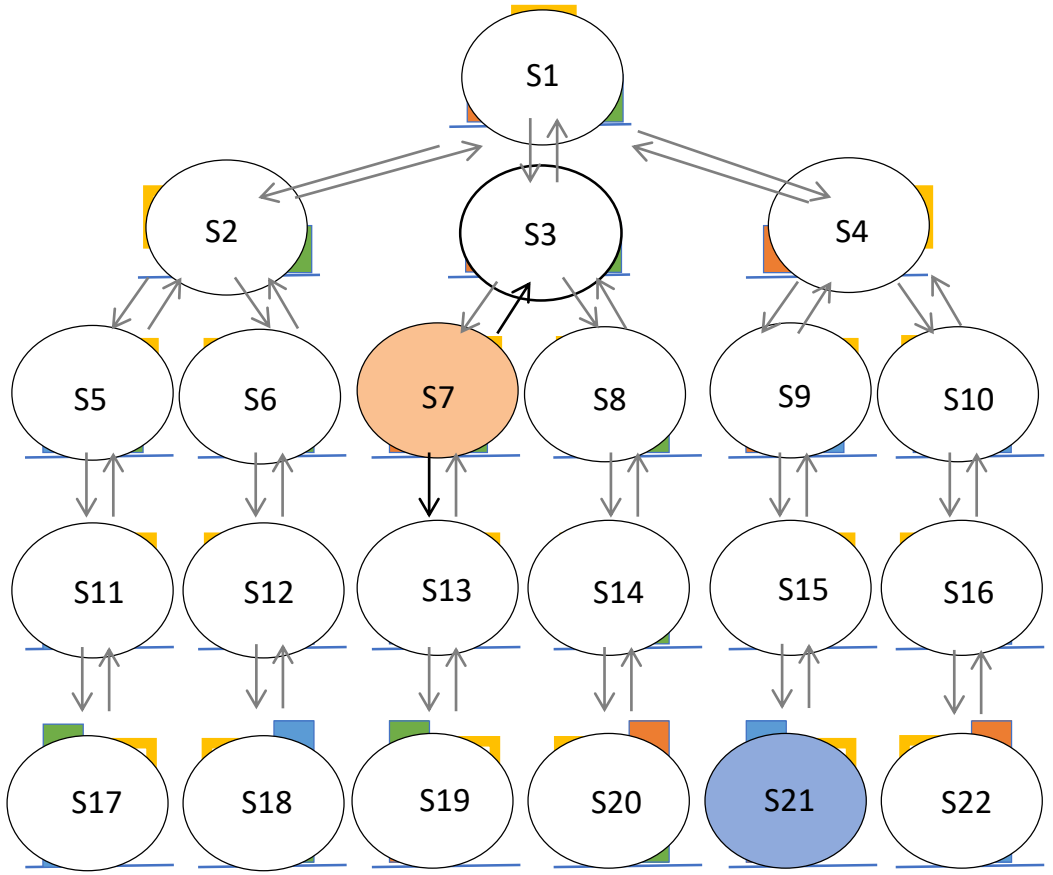


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

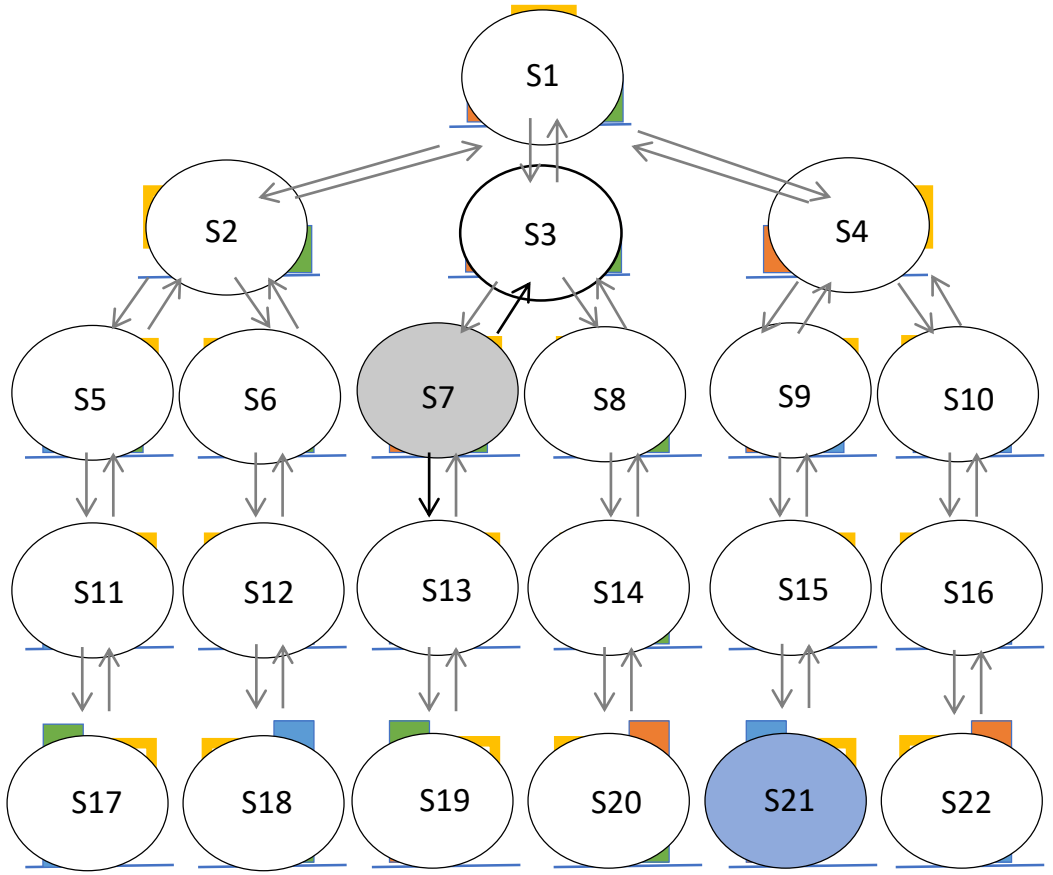
toVisit = [S3,S13]

visited = [S7]



Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```



toVisit = [S3,S13]

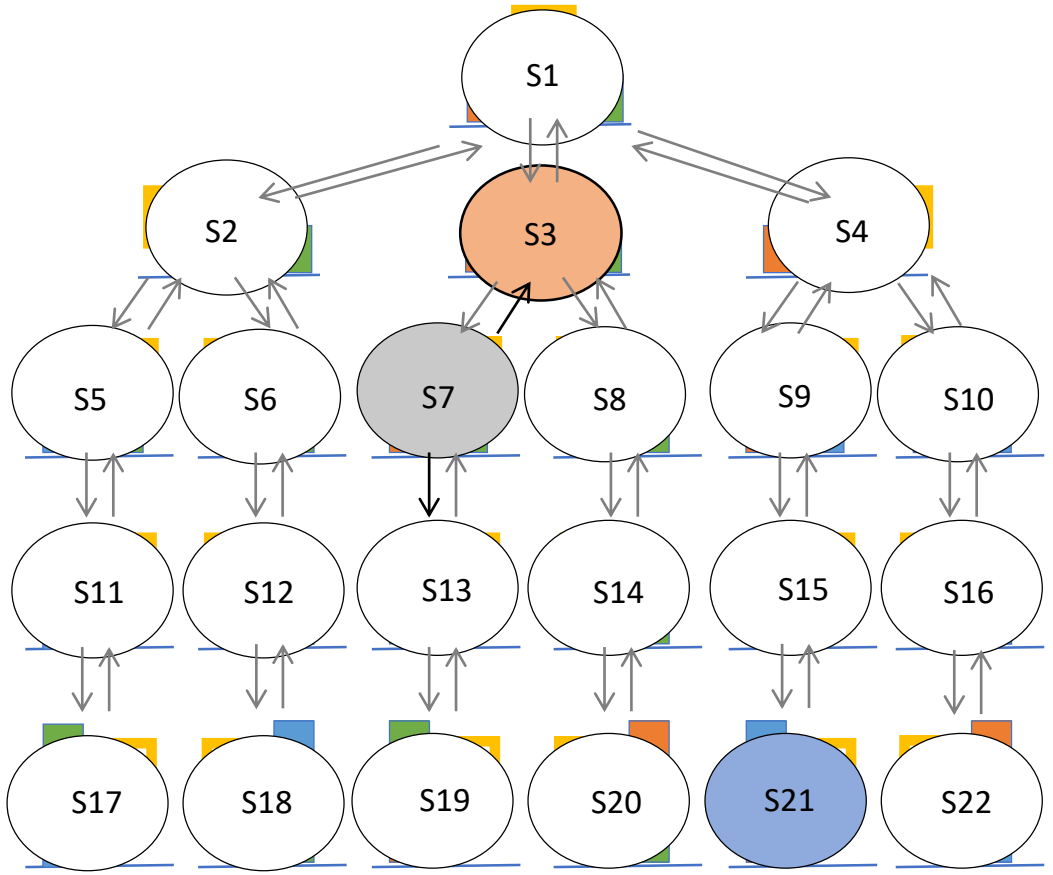
visited = [S7]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S3,S13]

visited = [S7]

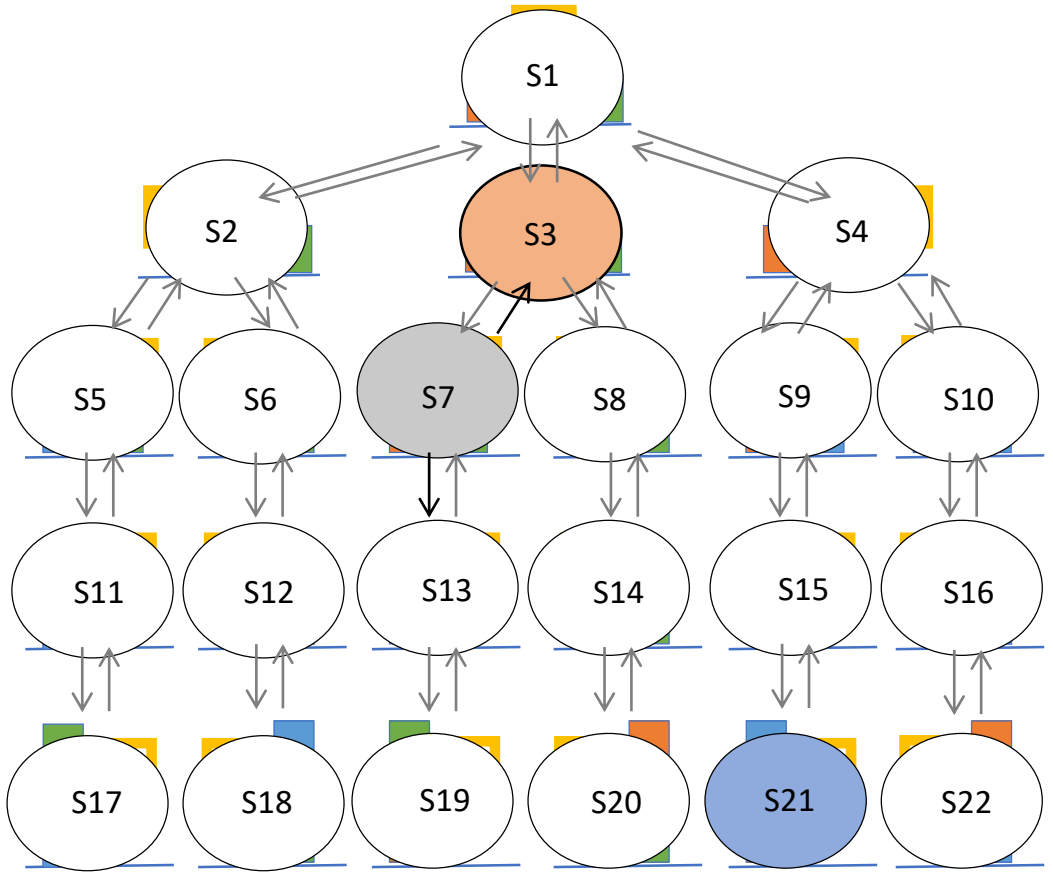


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S13]

visited = [S7,S3]

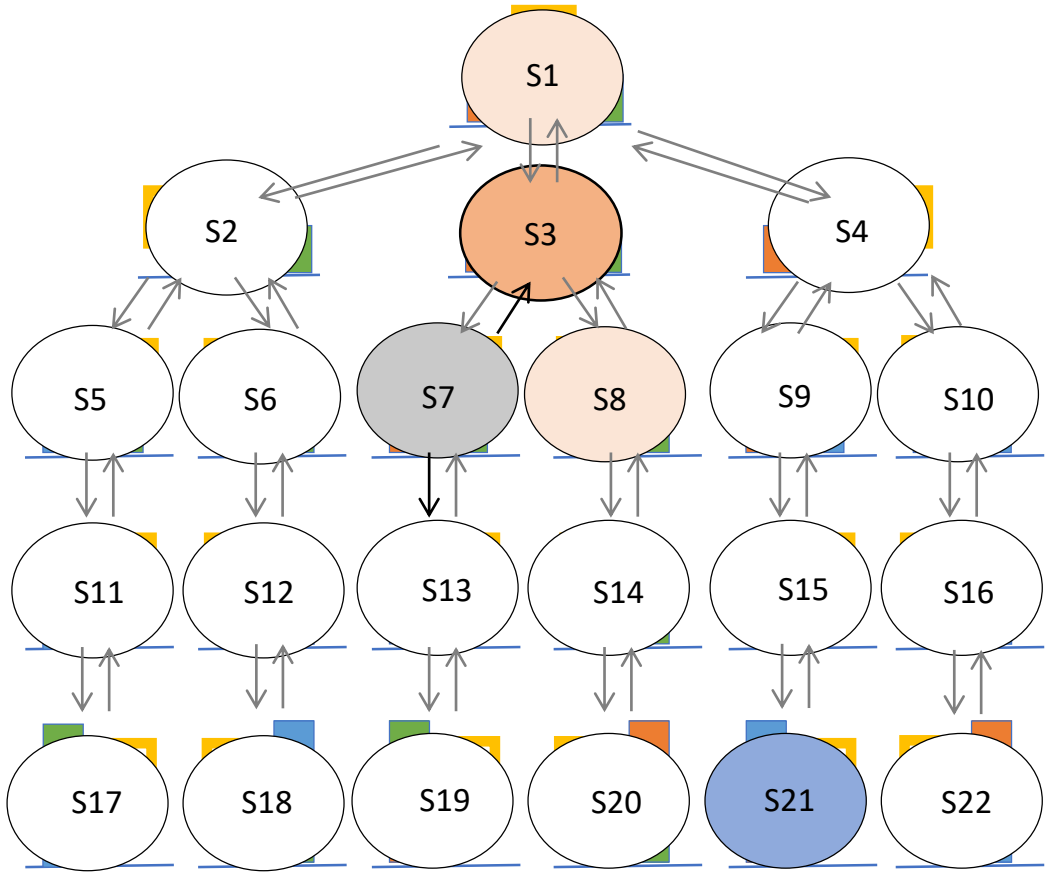


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S13]

visited = [S7,S3]

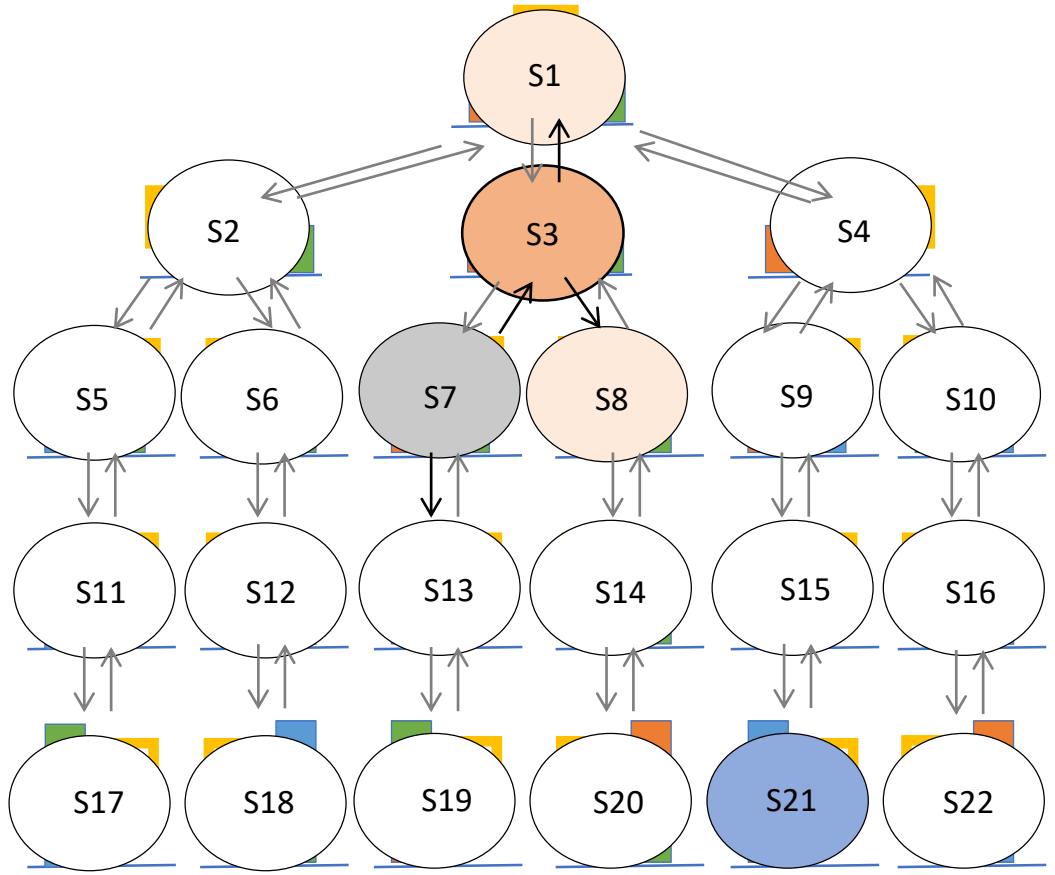


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S13,S1,S8]

visited = [S7,S3]

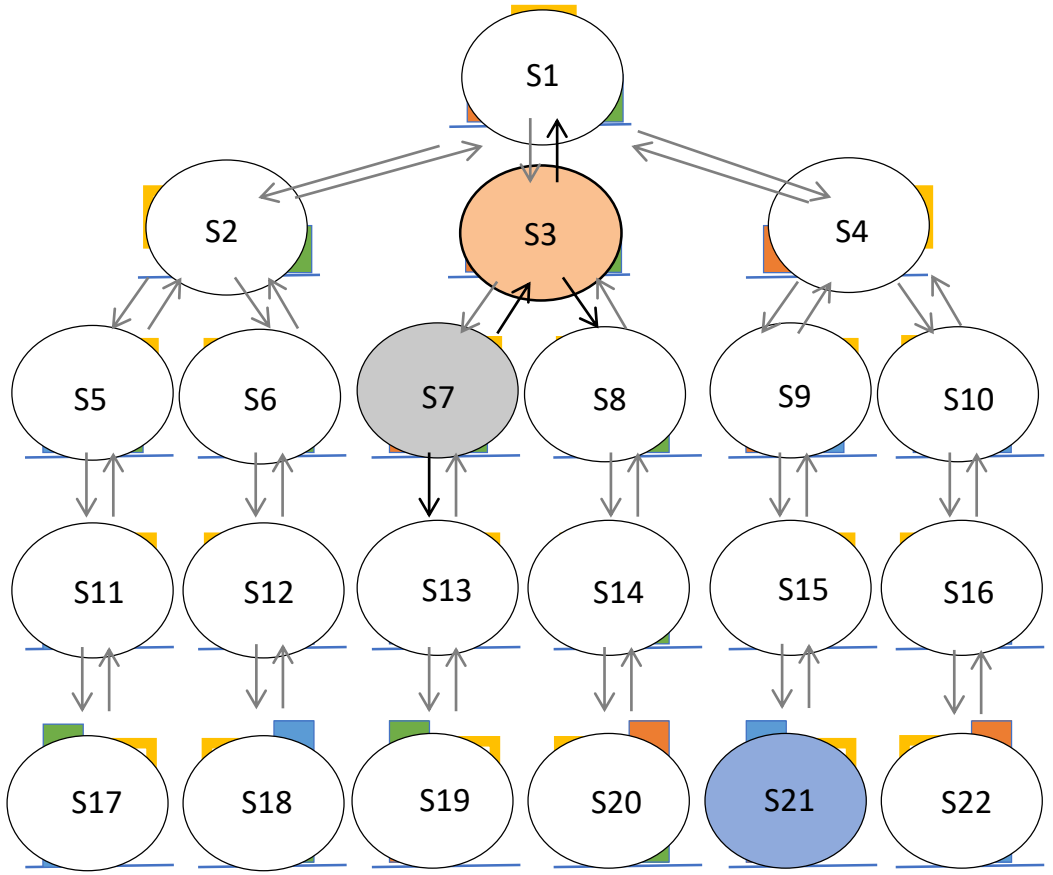


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

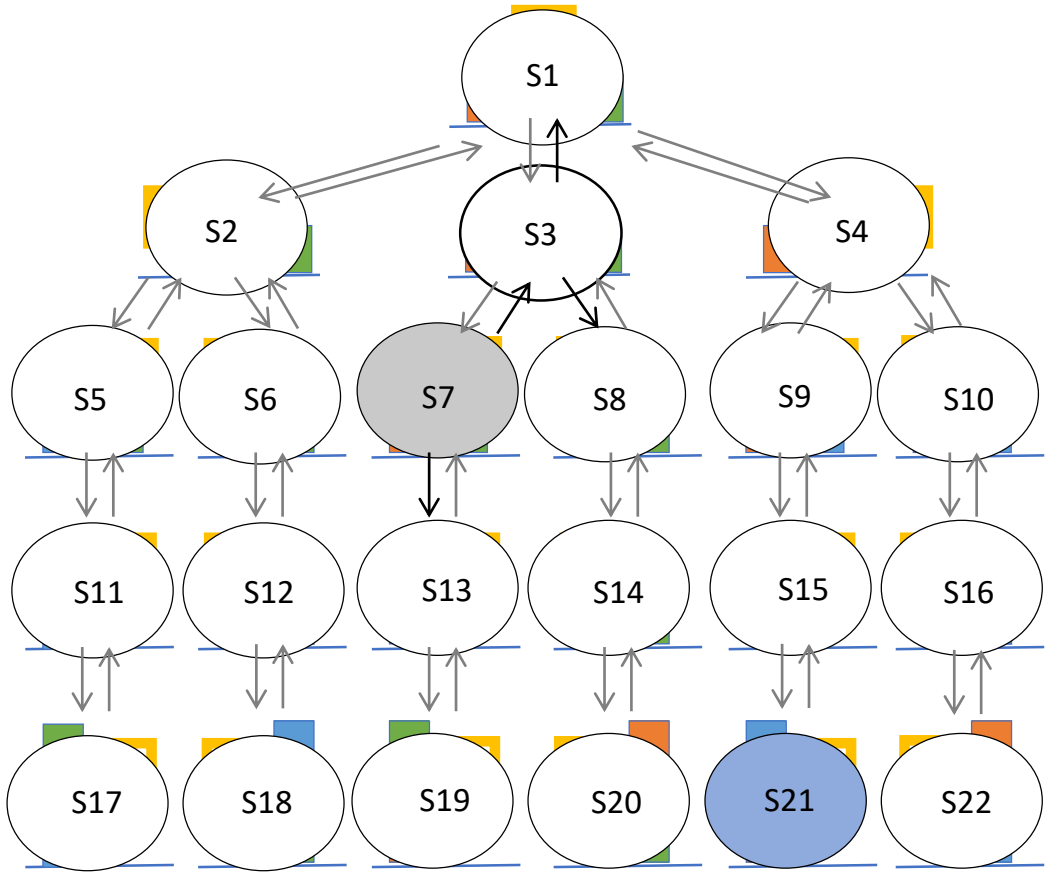
toVisit = [S13,S1,S8]

visited = [S7,S3]



Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

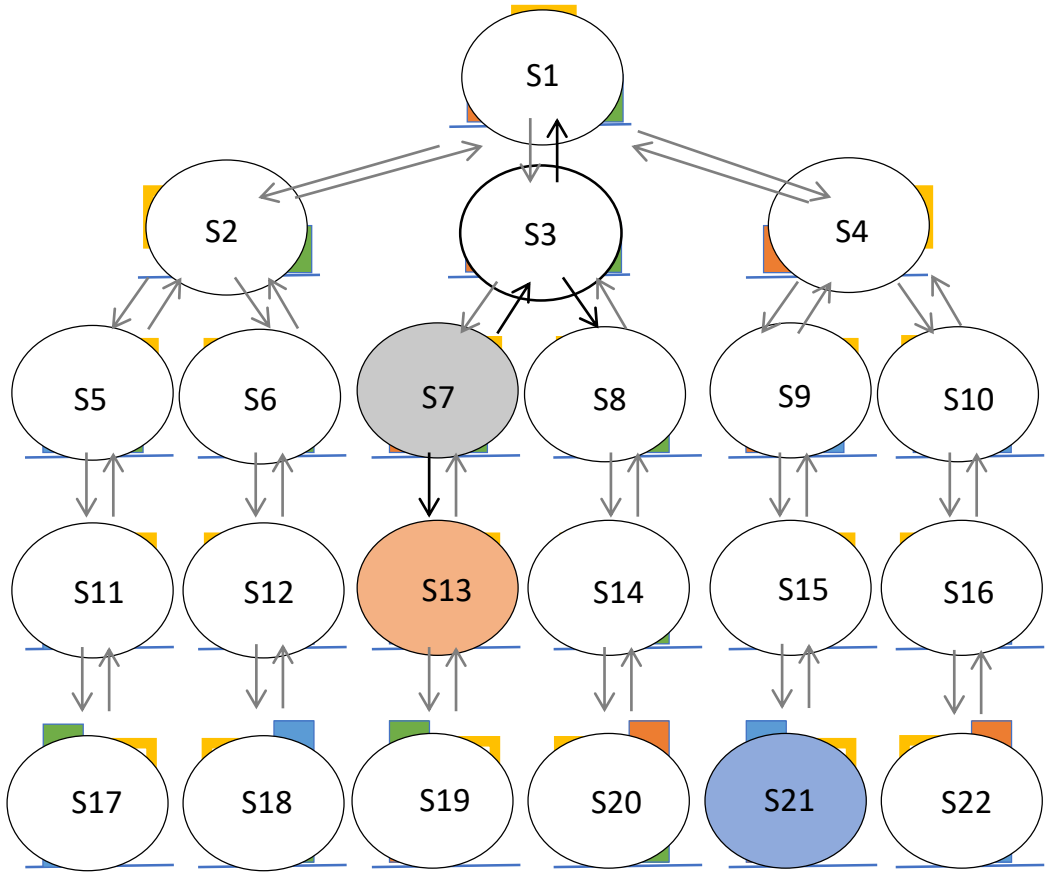


toVisit = [S13,S1,S8]

visited = [S7,S3]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```



toVisit = [S1,S8]

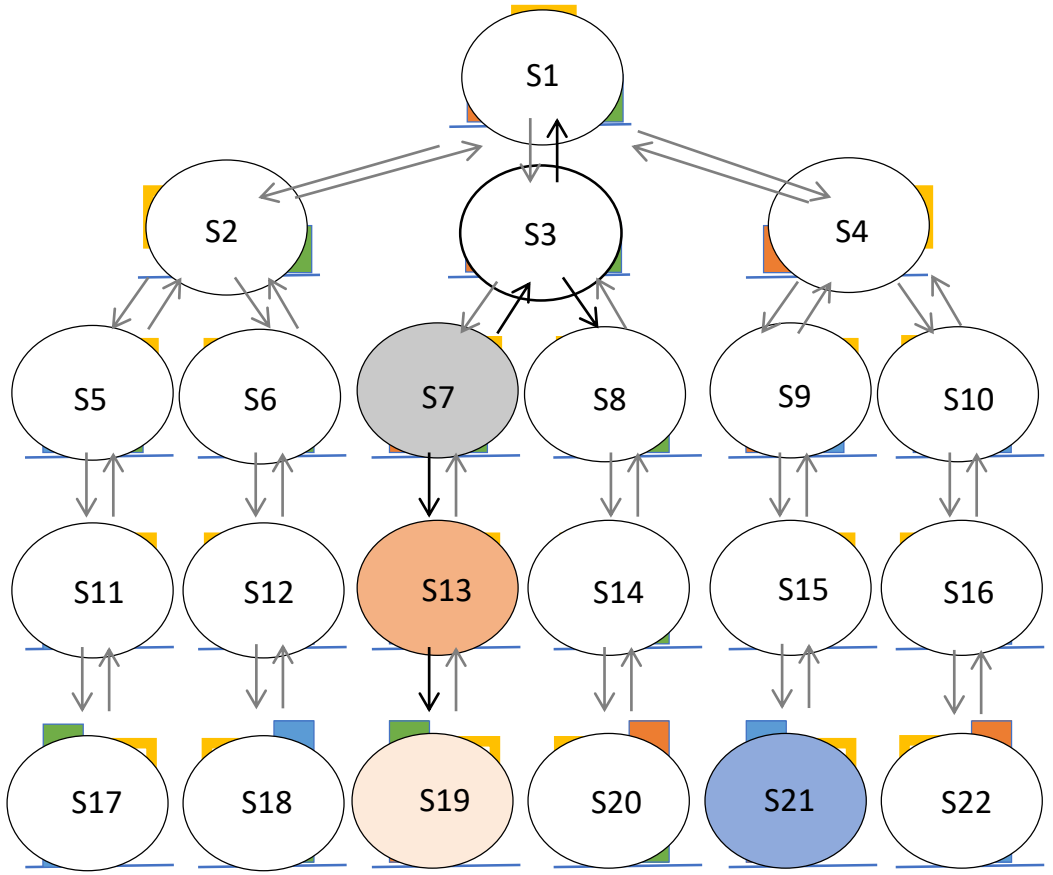
visited = [S7,S3,S13]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

toVisit = [S1,S8,S19]

visited = [S7,S3,S13]

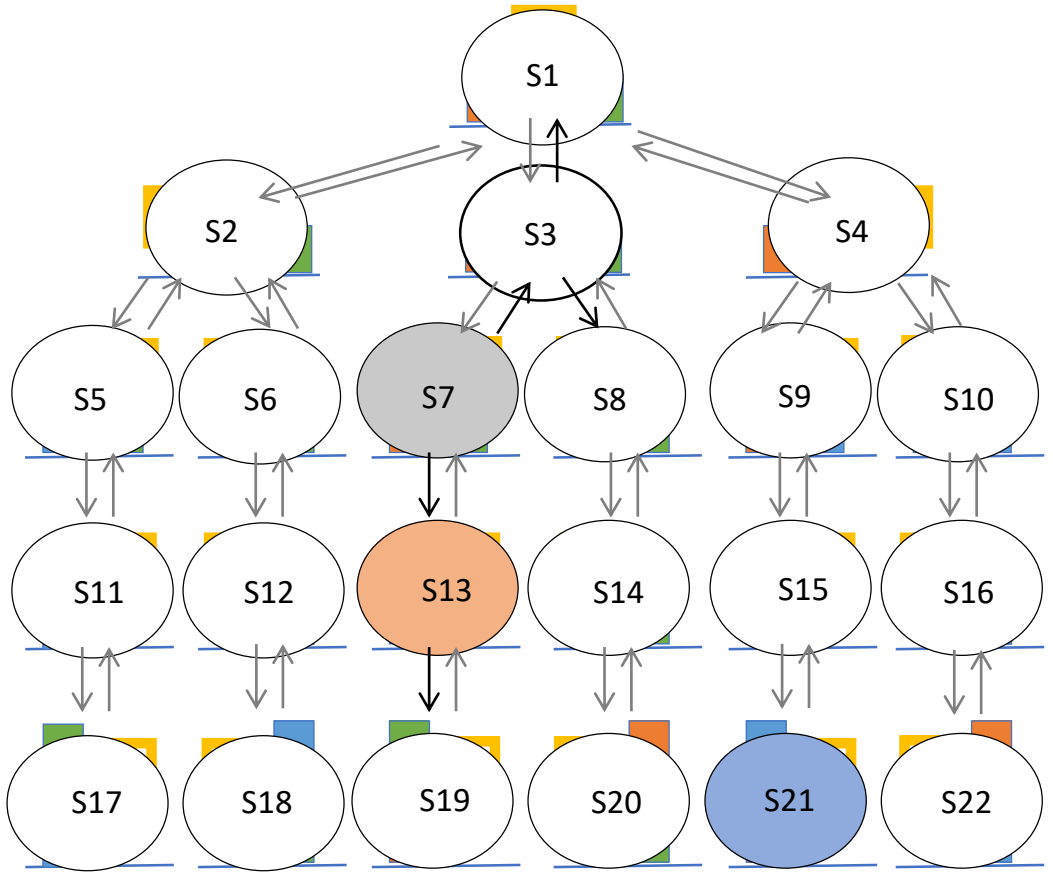


Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

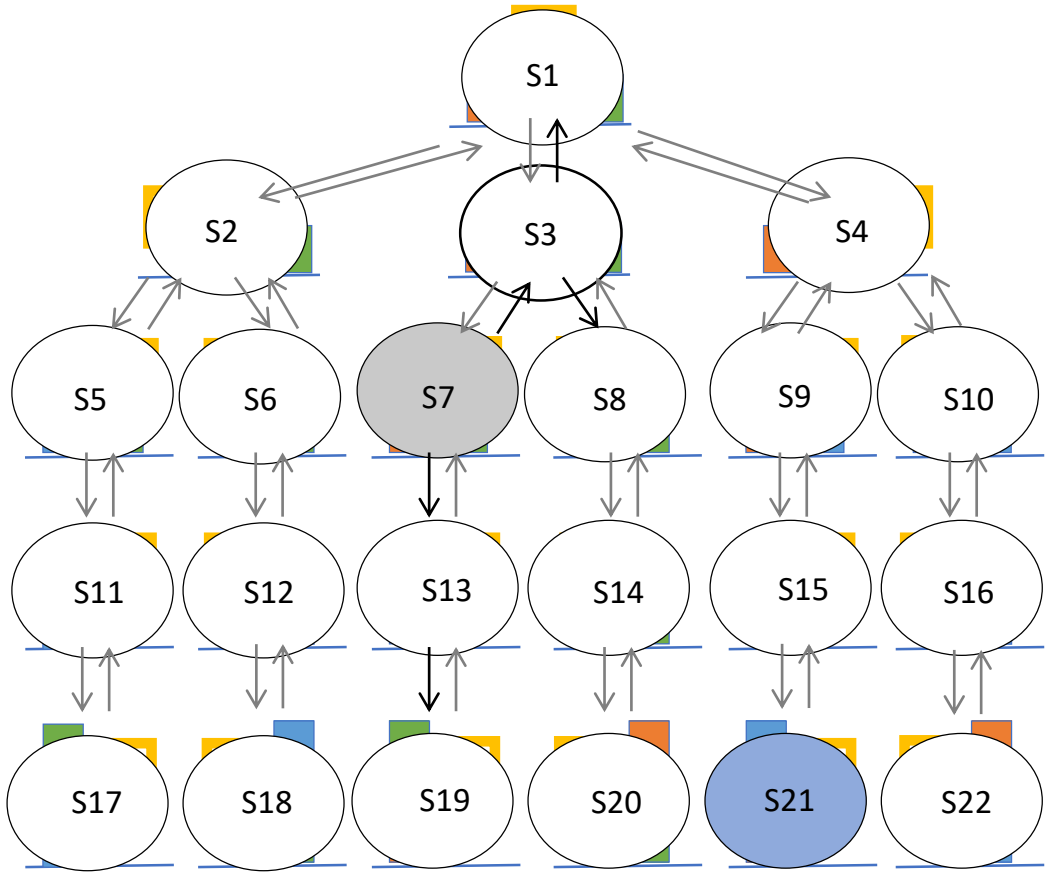
toVisit = [S1,S8,S19]

visited = [S7,S3,S13]



Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

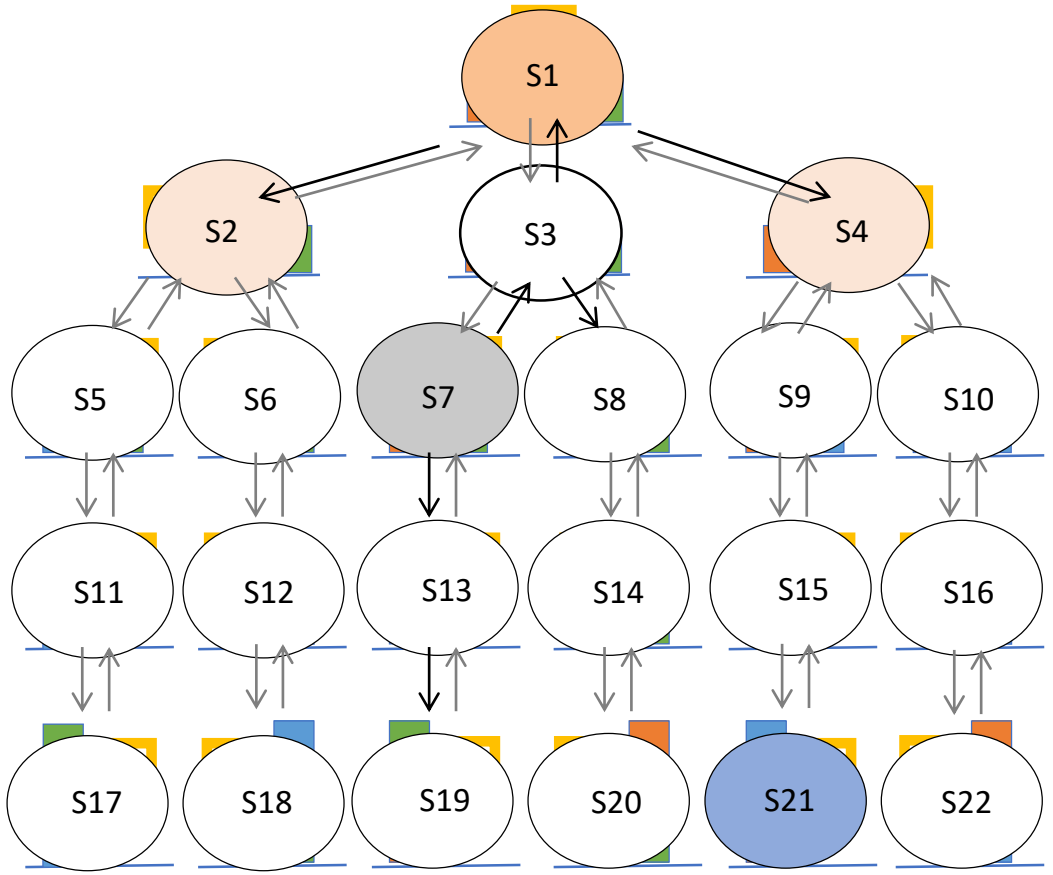


toVisit = [S1,S8,S19]

visited = [S7,S3,S13]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

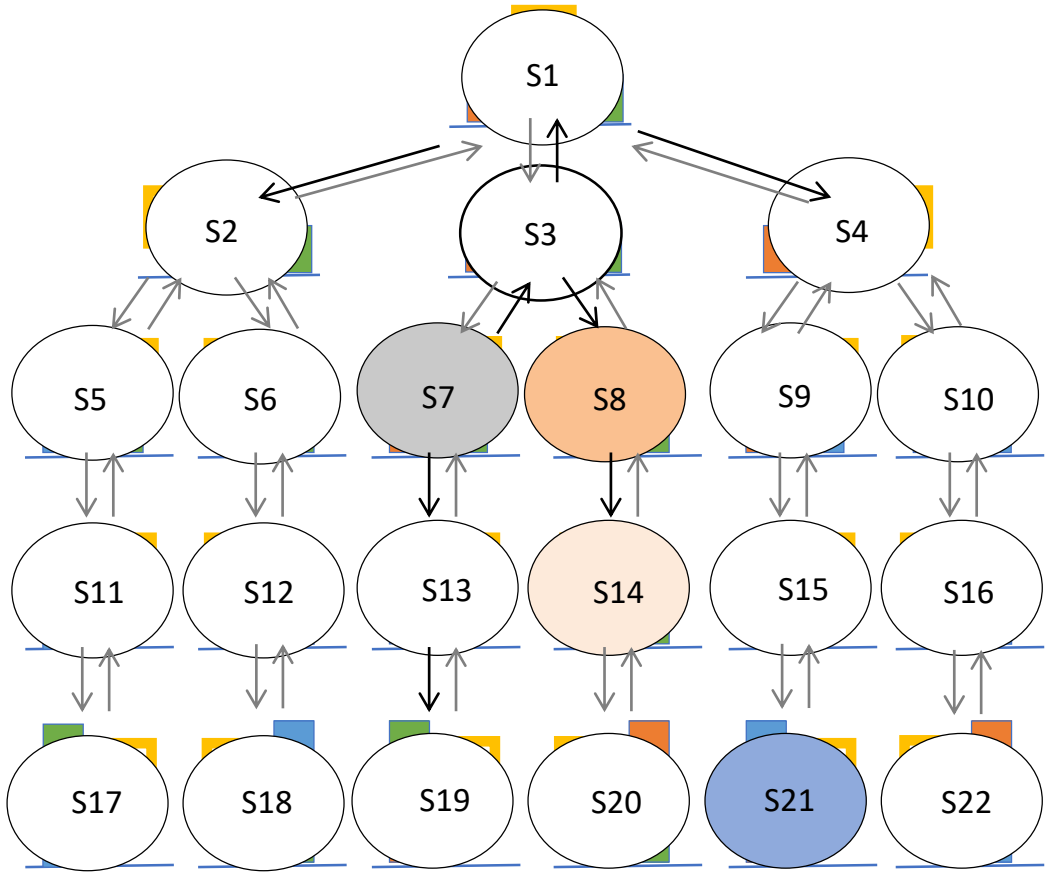


toVisit = [S8,S19,S2,S4]

visited = [S7,S3,S13,S1]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

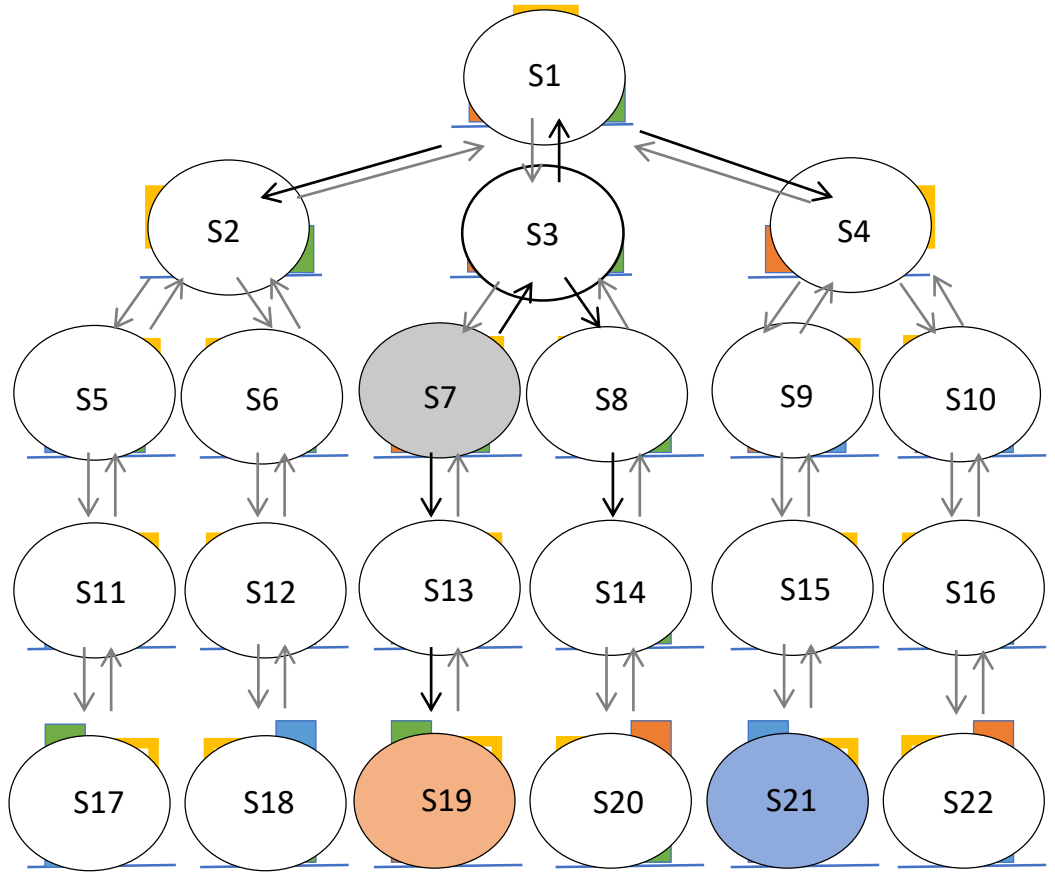


toVisit = [S19,S2,S4,S14]

visited = [S7,S3,S13,S1,S8]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

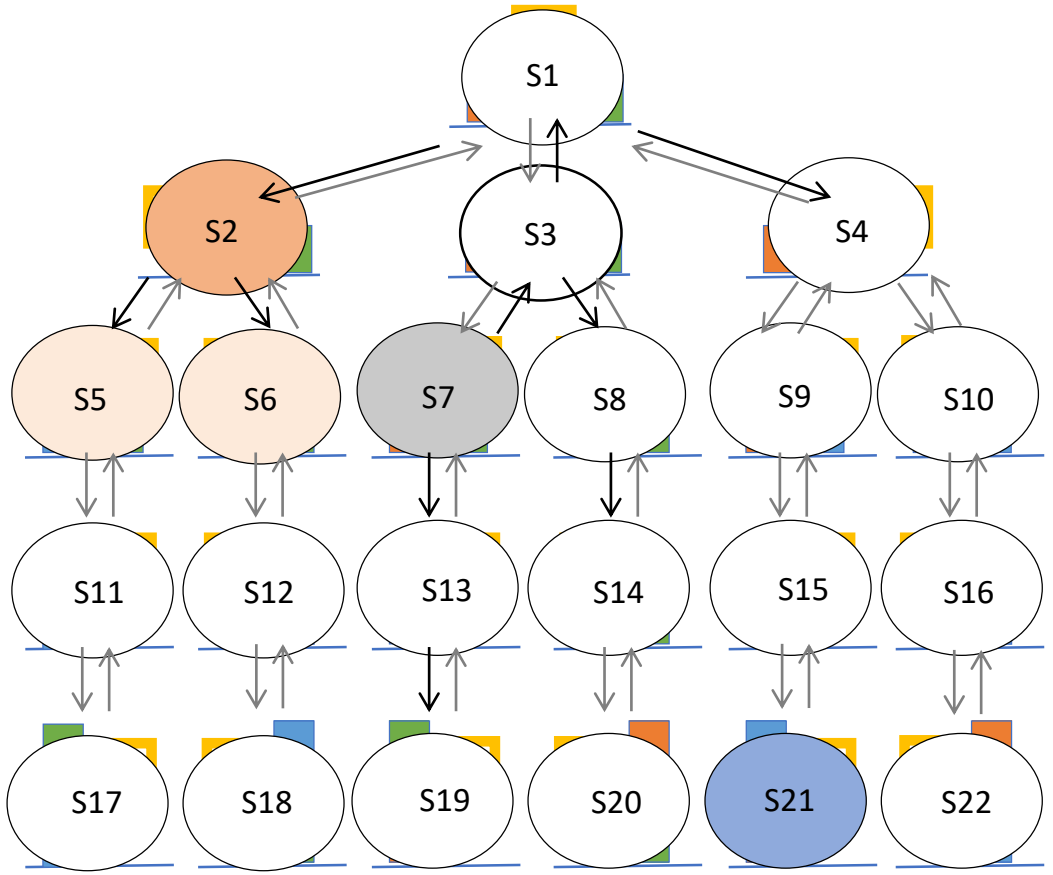


toVisit = [S2,S4,S14]

visited = [S7,S3,S13,S1,S8,S19]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

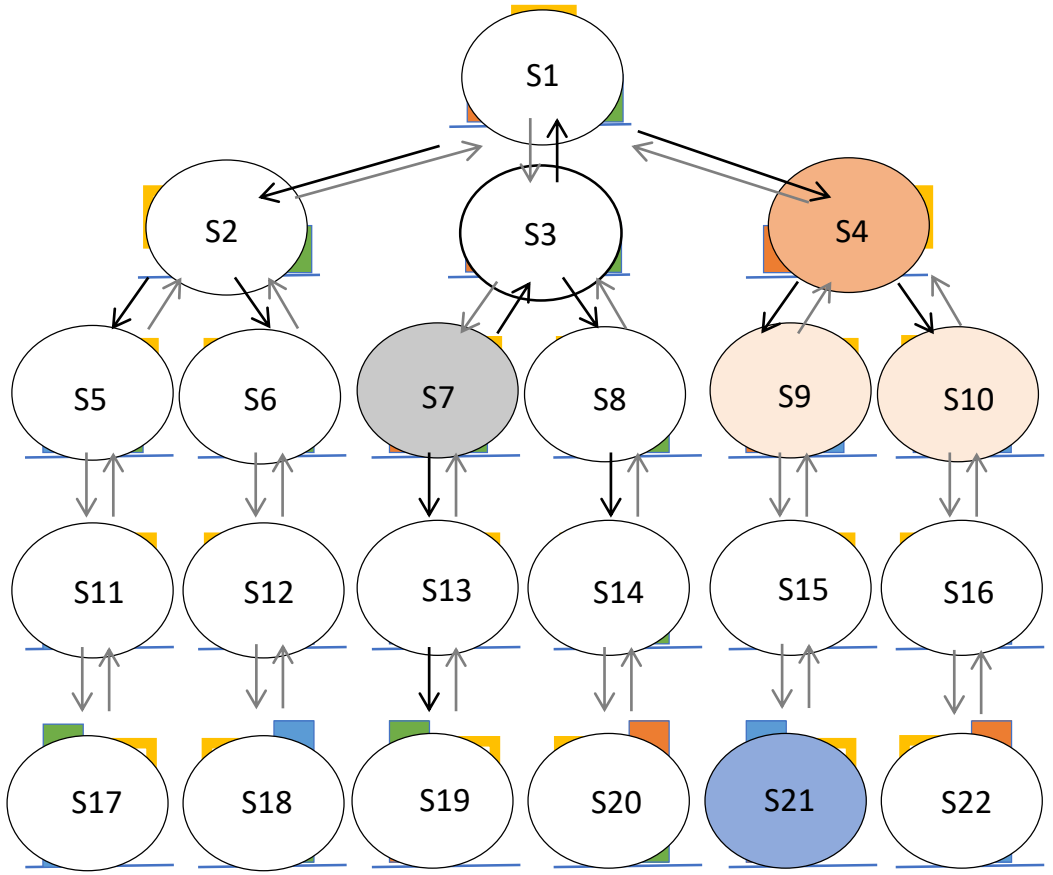


toVisit = [S4,S14,S5,S6]

visited = [S7,S3,S13,S1,S8,S19,S2]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

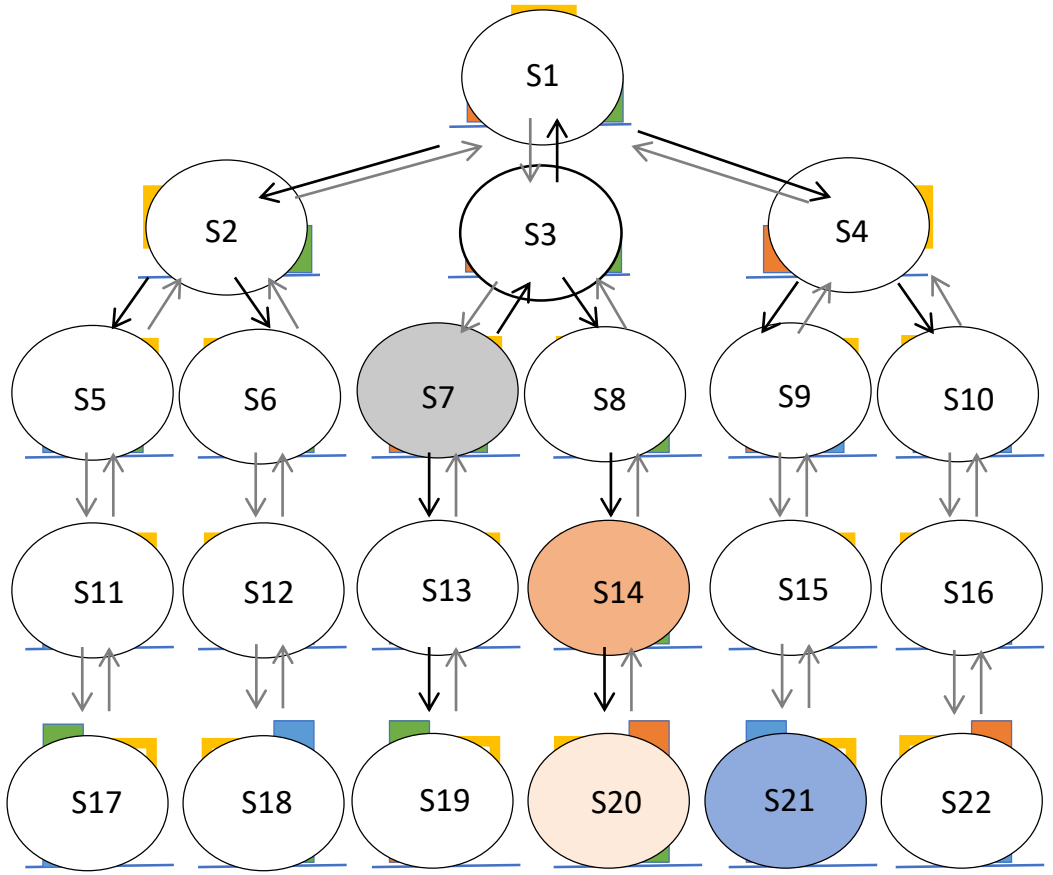


toVisit = [S14,S5,S6,S9,S10]

visited = [S7,S3,S13,S1,S8,S19,S2,S4]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

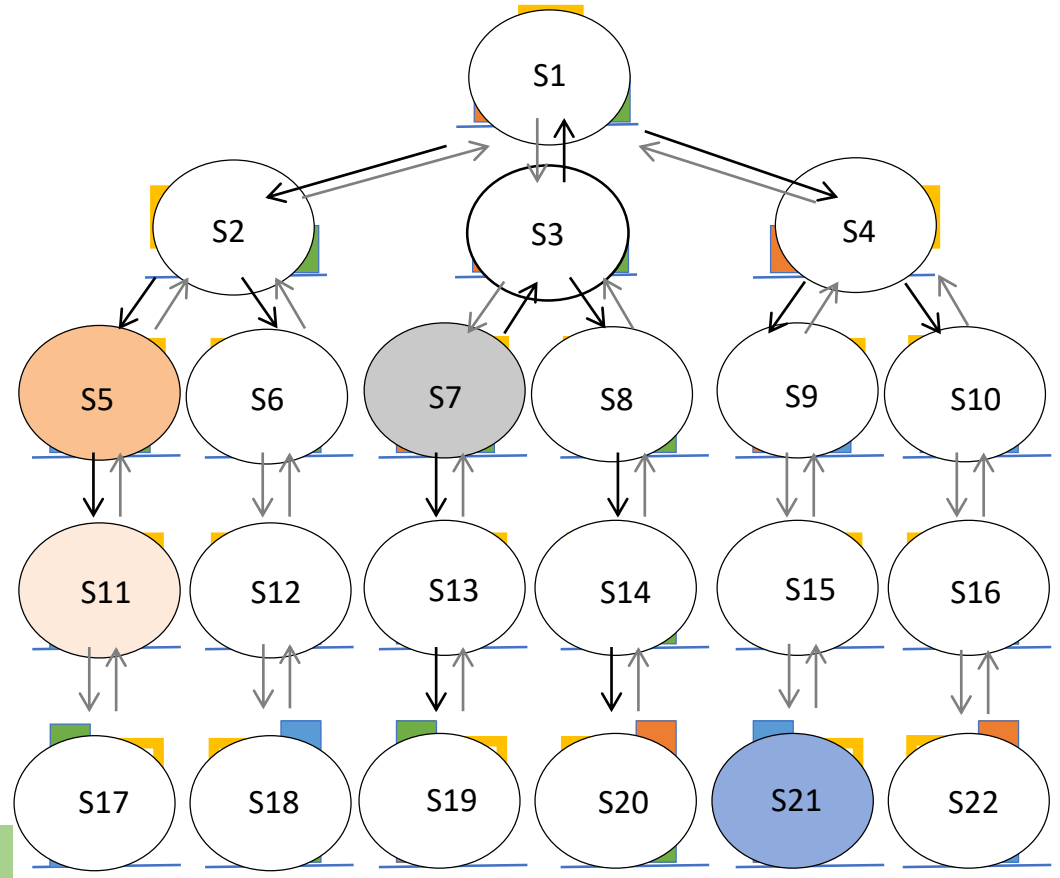


toVisit = [S5,S6,S9,S10,S20]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

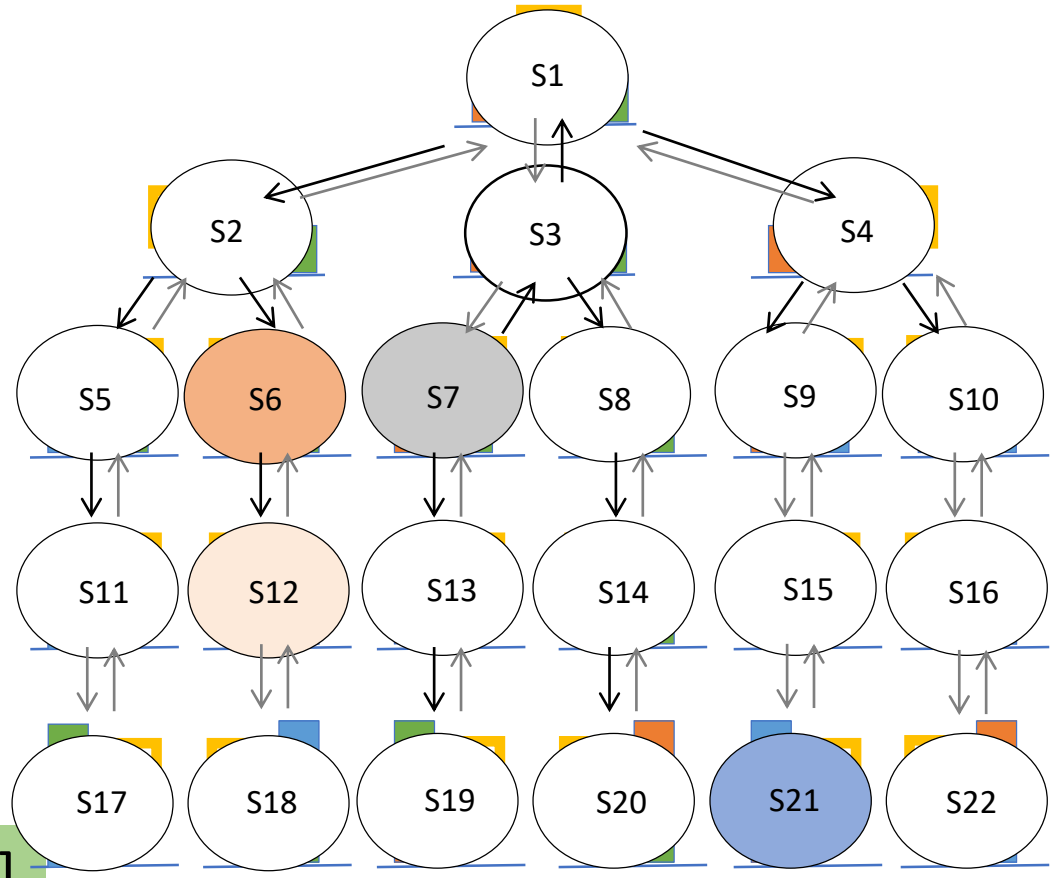


toVisit = [S6,S9,S10,S20,S11]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

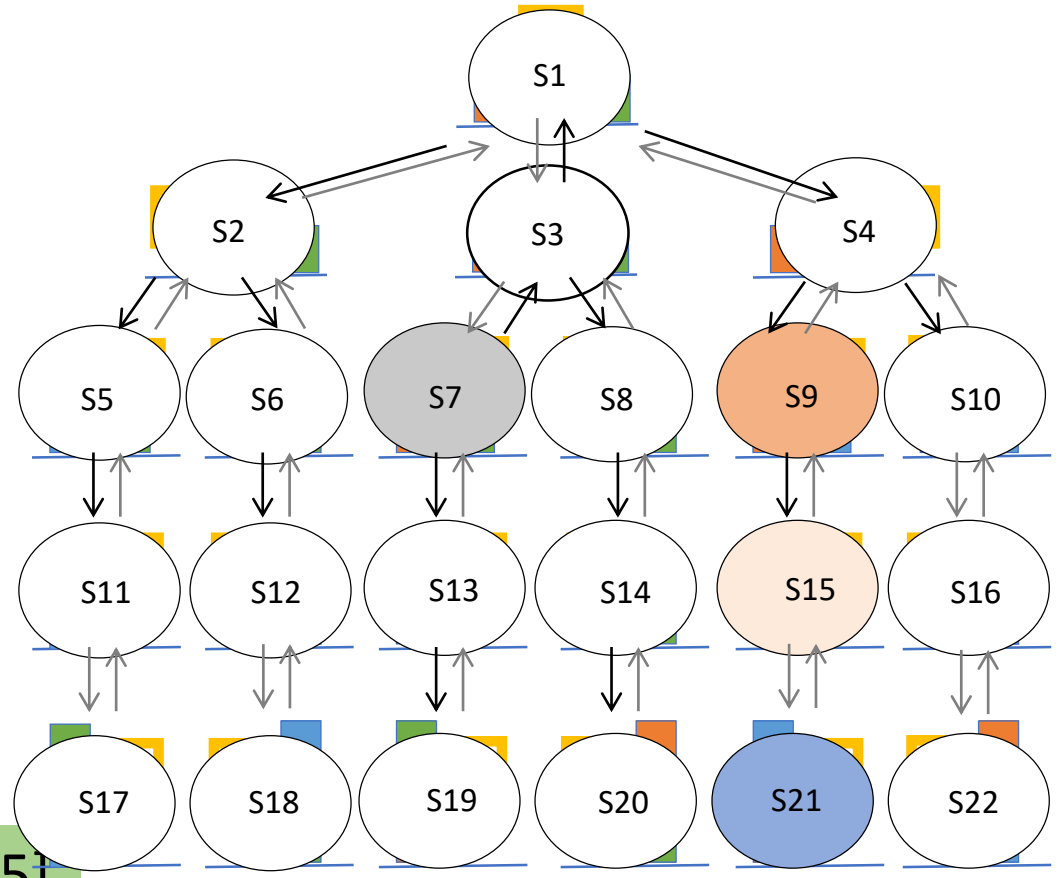


toVisit = [S9,S10,S20,S11,S12]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

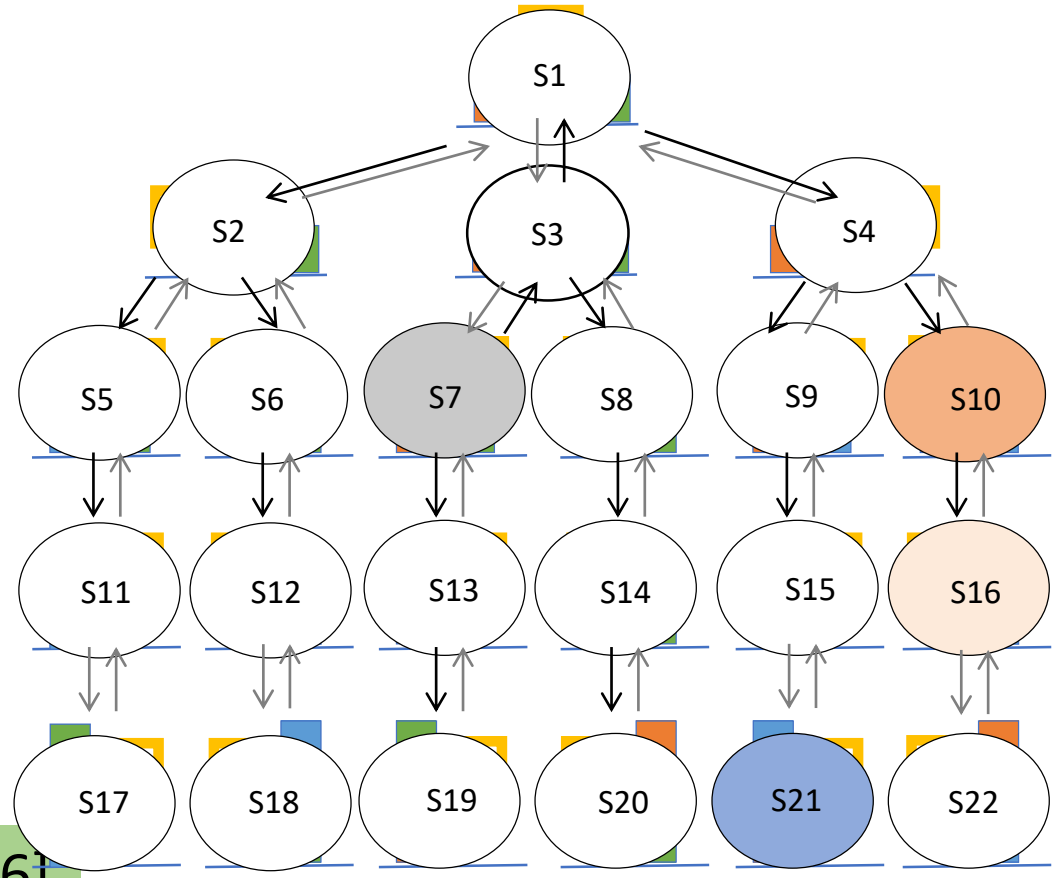


toVisit = [S10,S20,S11,S12,S15]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

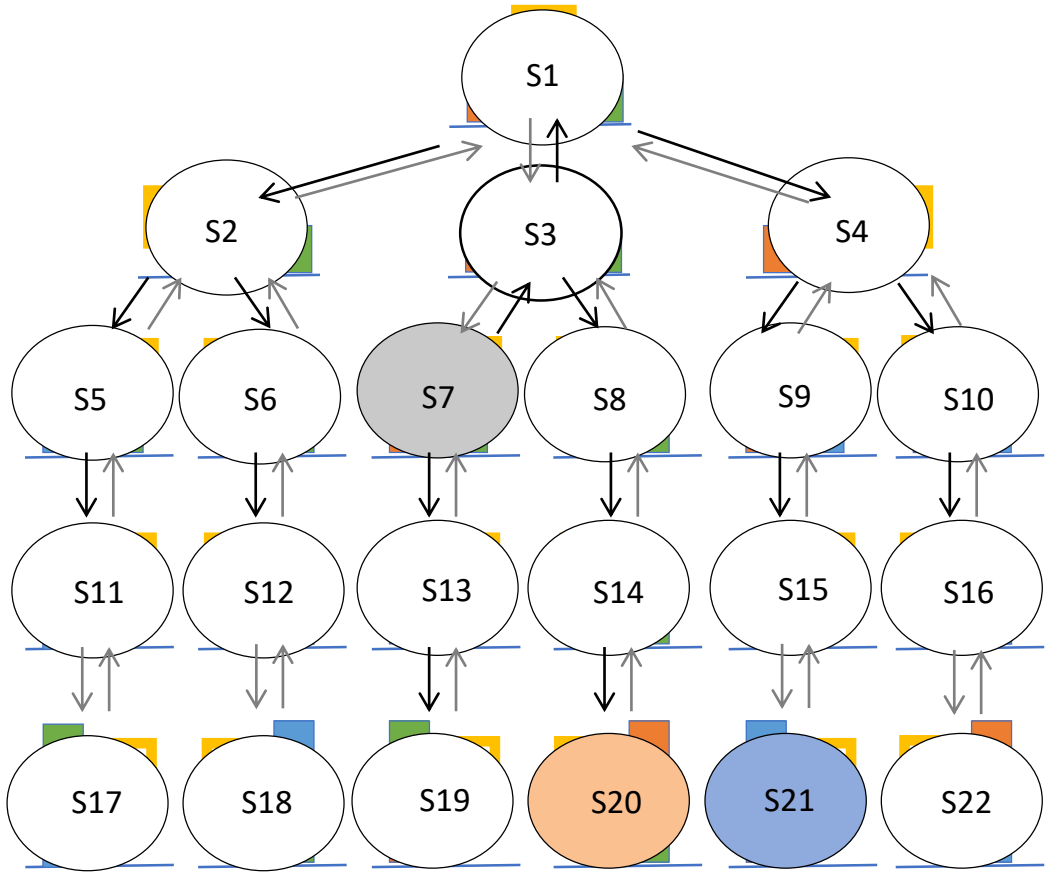


toVisit = [S20,S11,S12,S15,S16]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

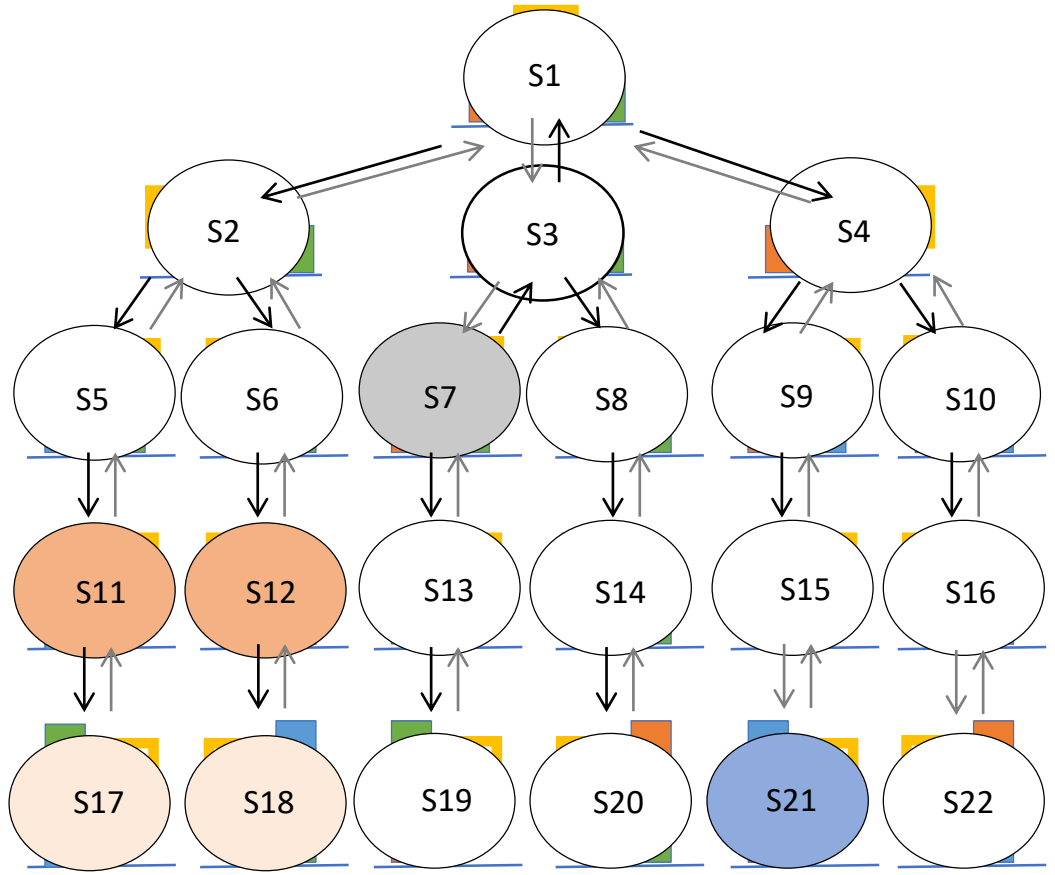


toVisit = [S11,S12,S15,S16]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

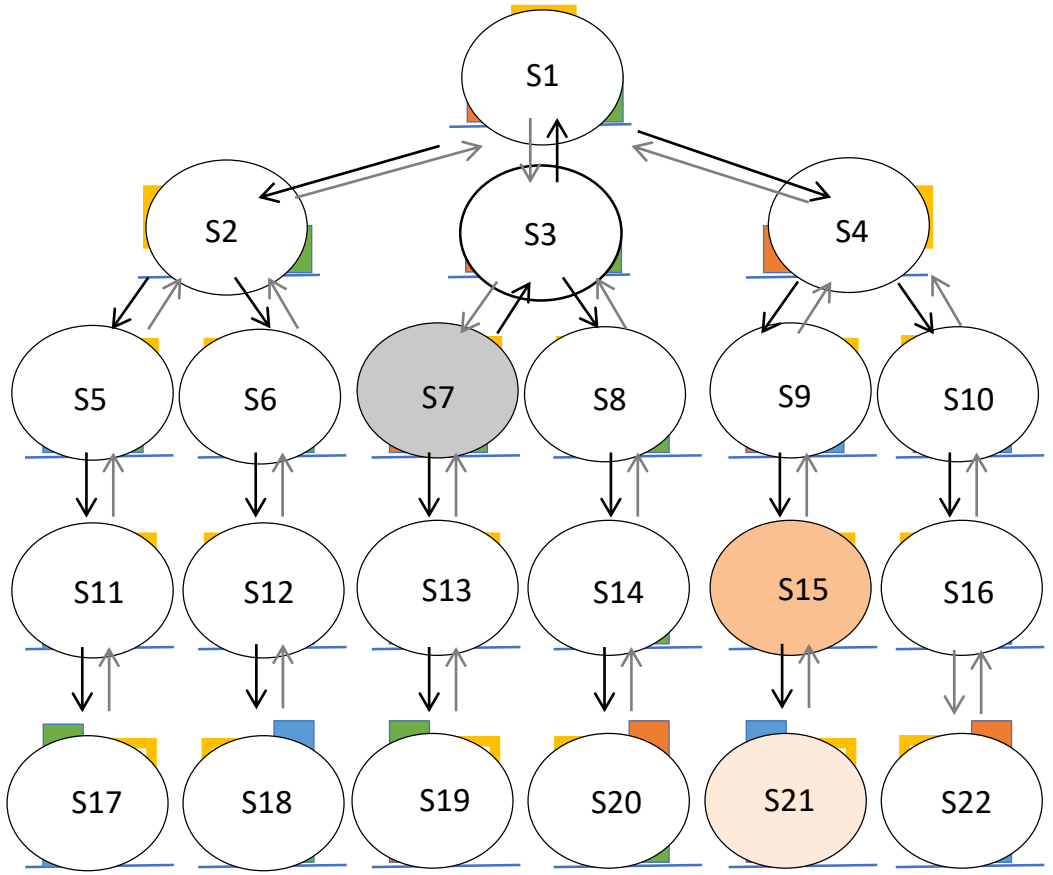


toVisit = [S15,S16,S17,S18]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

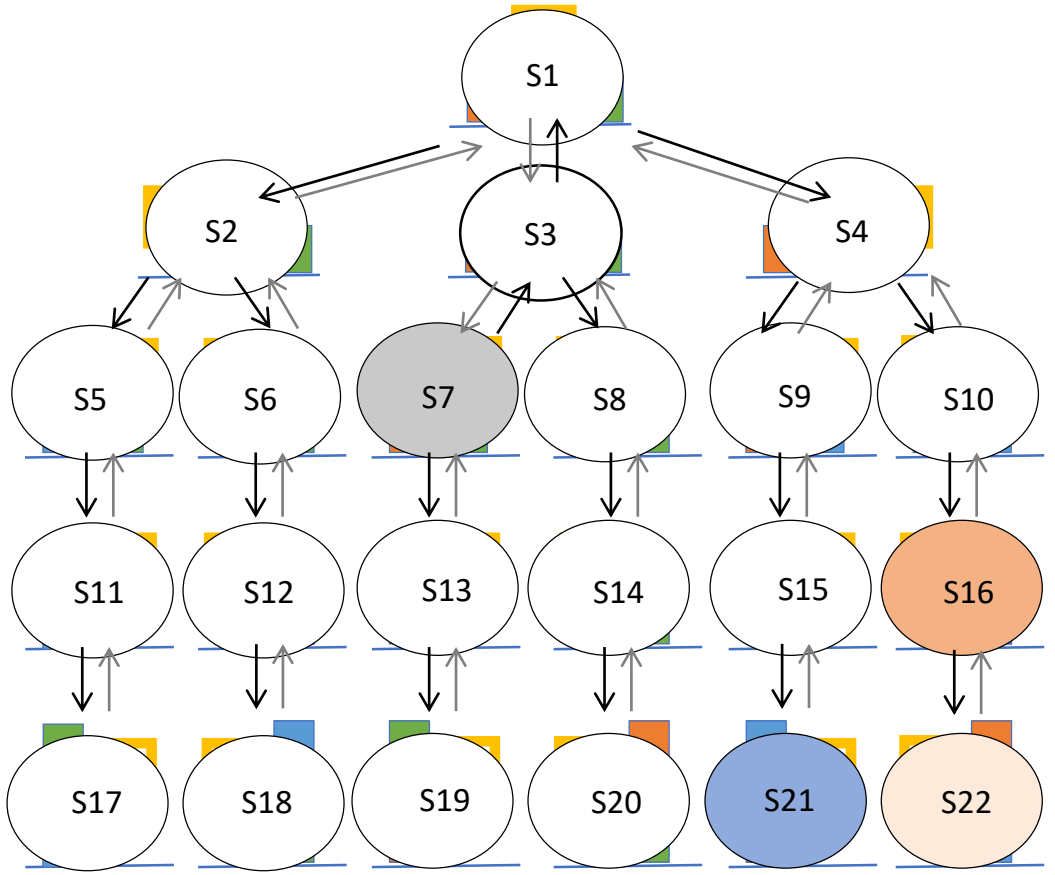


toVisit = [S16,S17,S18,S21]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

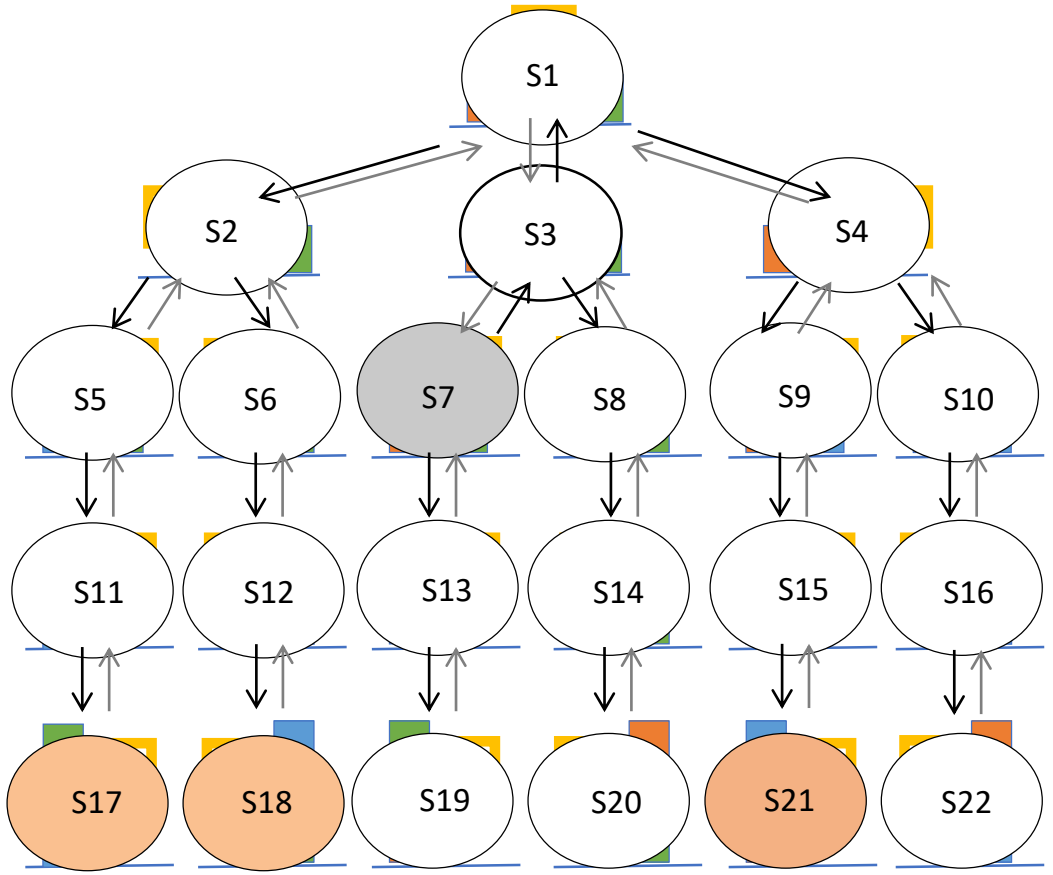


toVisit = [S17,S18,S21,S22]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15,S16]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```

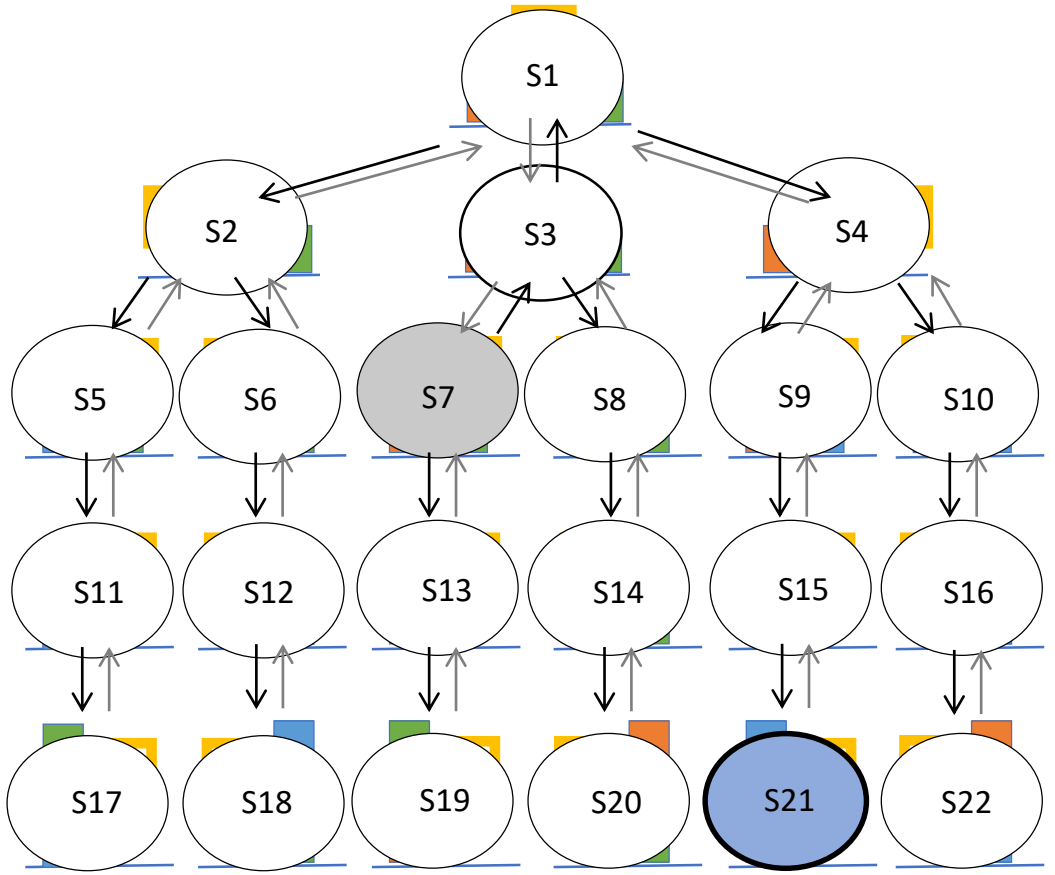


toVisit = [S21,S22]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15,S16,S17,S18]

Searching a Graph

```
toVisit = [start]
visited = []
actions = []
while len(toVisit) > 0:
        state = toVisit[0]
        visited.append(state)
        toVisit.remove(state)
        neighbors = graph[state]
        for n in neighbors:
                actions.append(n)
                toVisit.append(n[1])
        if state == goal:
                break
```



toVisit = [S22]

visited = [S7,S3,S13,S1,S8,S19,S2,S4,S14,S5,S6,S9,S10,S20,S11,S12,S15,S16,S17,S18,S21]