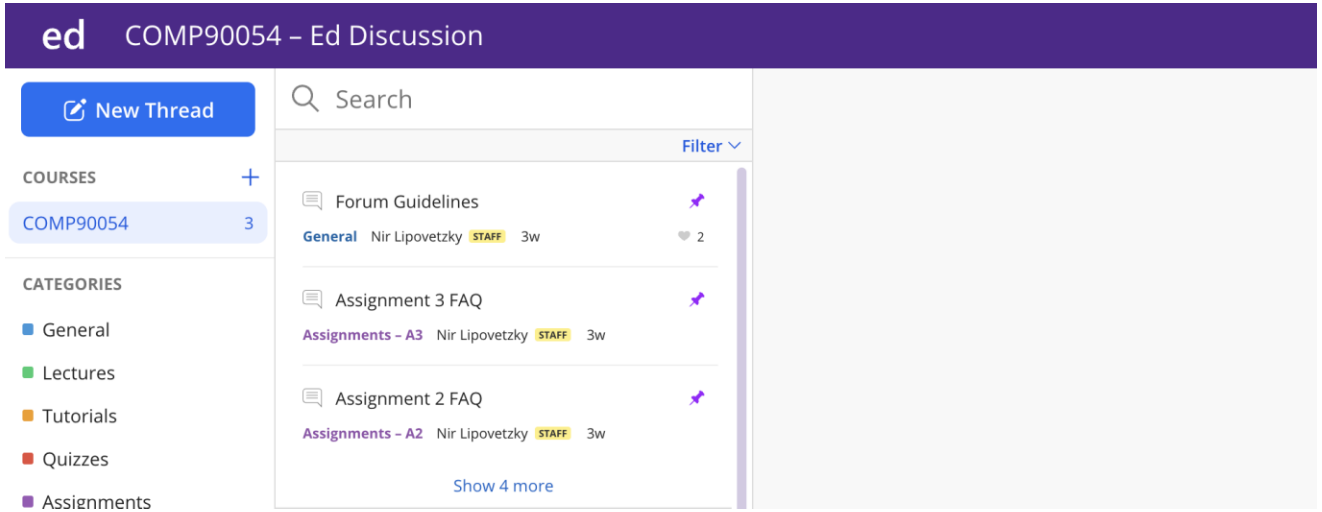
Week 2: Blind Search

COMP90054 – AI Planning for Autonomy

- Tutor Name: Thao Le

- Questions? Ed Discussion



Key concepts

- State-space model
- Blind search algorithms:
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Iterative Deepening (ID)

Problem 1

State space model:

P = $<\boldsymbol{s_0}, \boldsymbol{S}, \boldsymbol{S_G}, \boldsymbol{A}, \boldsymbol{f}, \boldsymbol{c}>$

- $\boldsymbol{s_0}$ is an initial state

$$s1$$

- $\boldsymbol{S}$ is a set that includes all states in the state space

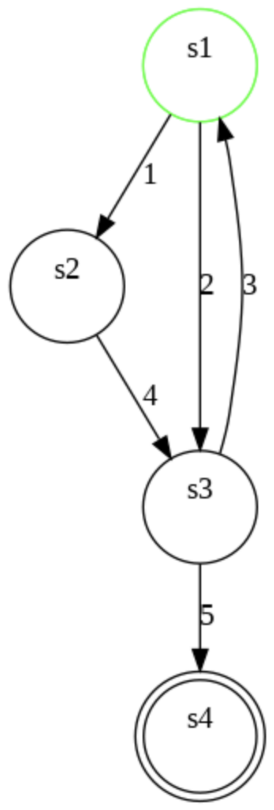
$$\mathrm{S} = \{s1, s2, s3, s4\}$$

- $\boldsymbol{S_G}$ is a goal set with all goal states

$$S_G = \{s4\}$$

▾ Problem 1:

Following the above example, define the state-space model of the graph:



Problem 1

State space model:
P = <s_0, S, S_G, A, f, c>

- $\boldsymbol{A}$ is an action set that includes all possible actions you can take from a state

$A(s1) = \{(s1, s2), (s1, s3)\}$ $A(s4) = ?$

- f is a transition function: $f(s, a) = s'$

state action next state

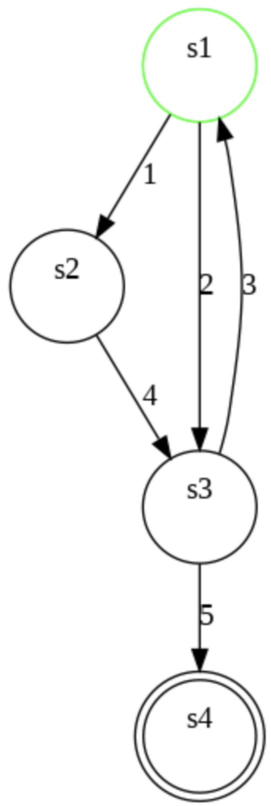
$f(s1, (s1, s2)) = s2$

- c is a cost function between two states

$c(s1, s2) = 1$ $c(s_1, s3) = 2$, $c(s_3, s_4) = 3$

Problem 1:

Following the above example, define the state-space model of the graph:

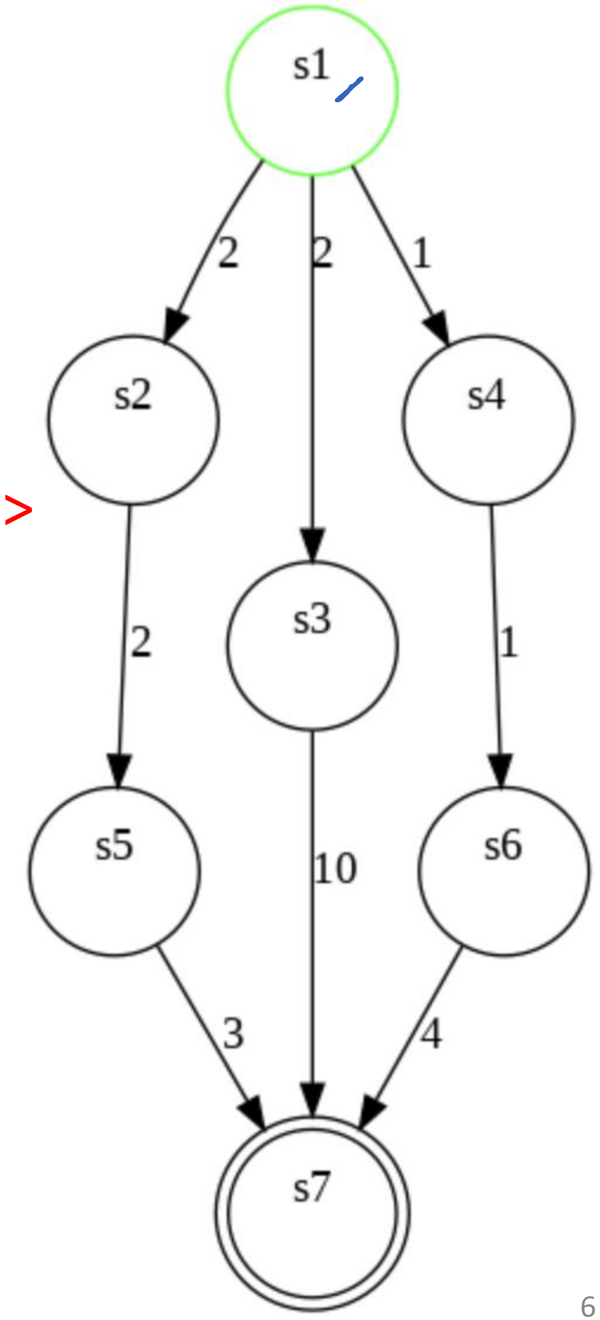


Problem 2

State vs Search node?

Search node n = <state, accumulated cost g(n), id of parent node>

n0 = <s1, 0, None>

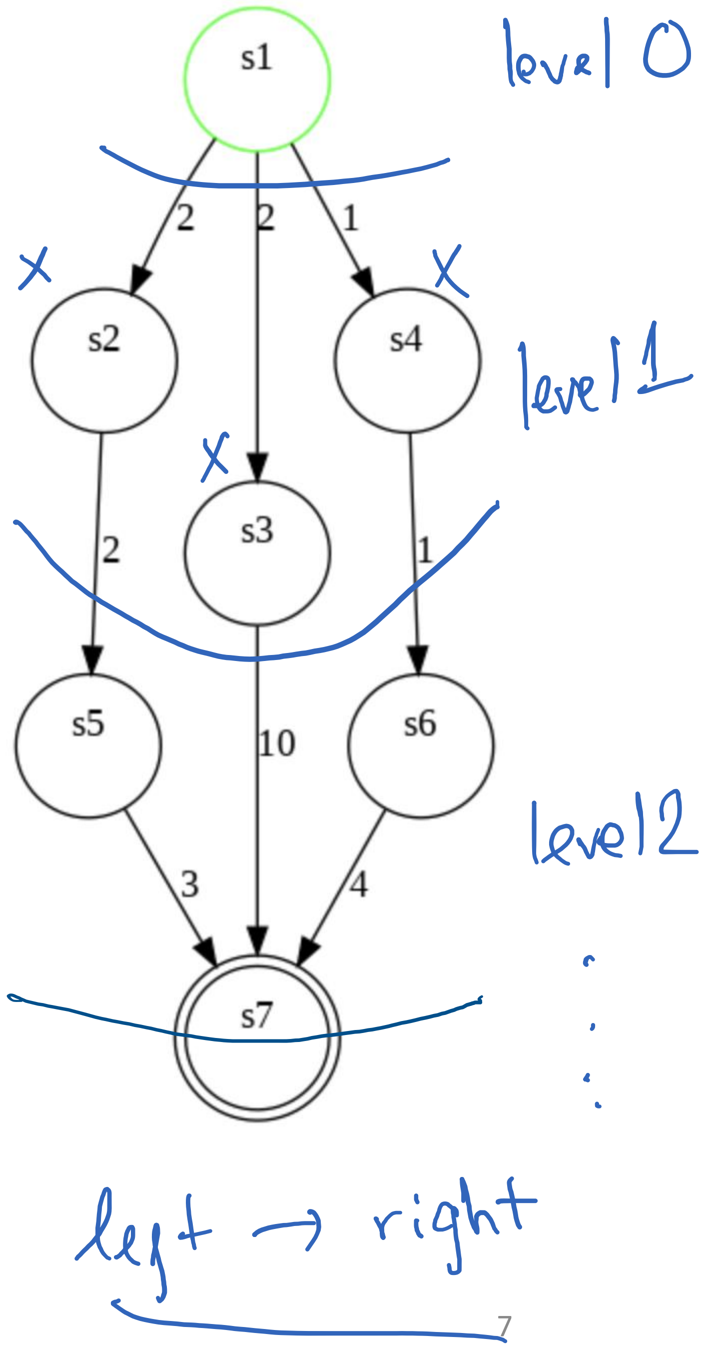


Problem 2

Breadth First Search (BFS)

Queue: FIFO

Search node n = (state, accumulated cost g(n), id of parent node)



Problem 2

Breadth First Search (BFS) Queue: FIFO

Search node n = (state, accumulated cost g(n), id of parent node)

x
n0 = (s1, 0, None)

n1 = (s2, 2, 0)

x
n2 = (s3, 2, 0)

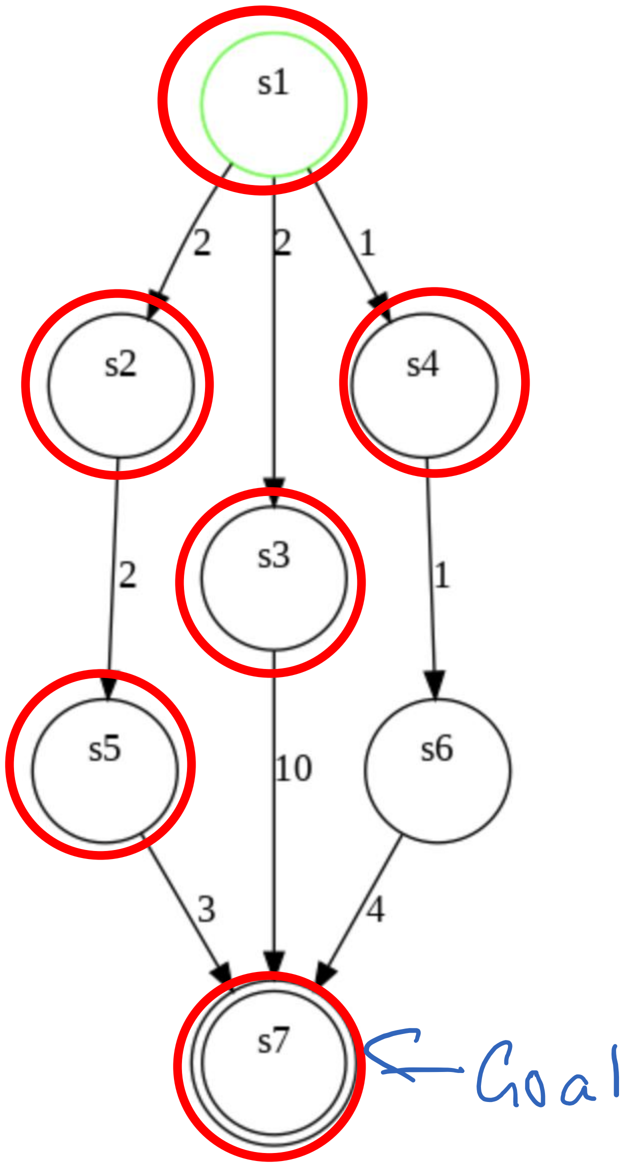
n3 = (s4, 1, 0)

n4 = (s5, 4, 1)

x
n5 = (s7, 12, 2)

node expansion order

Solution: s1 -> s3 -> s7



Problem 2

Depth First Search (DFS) Stack: LIFO

Search node n = (state, accumulated cost g(n), id of parent node)

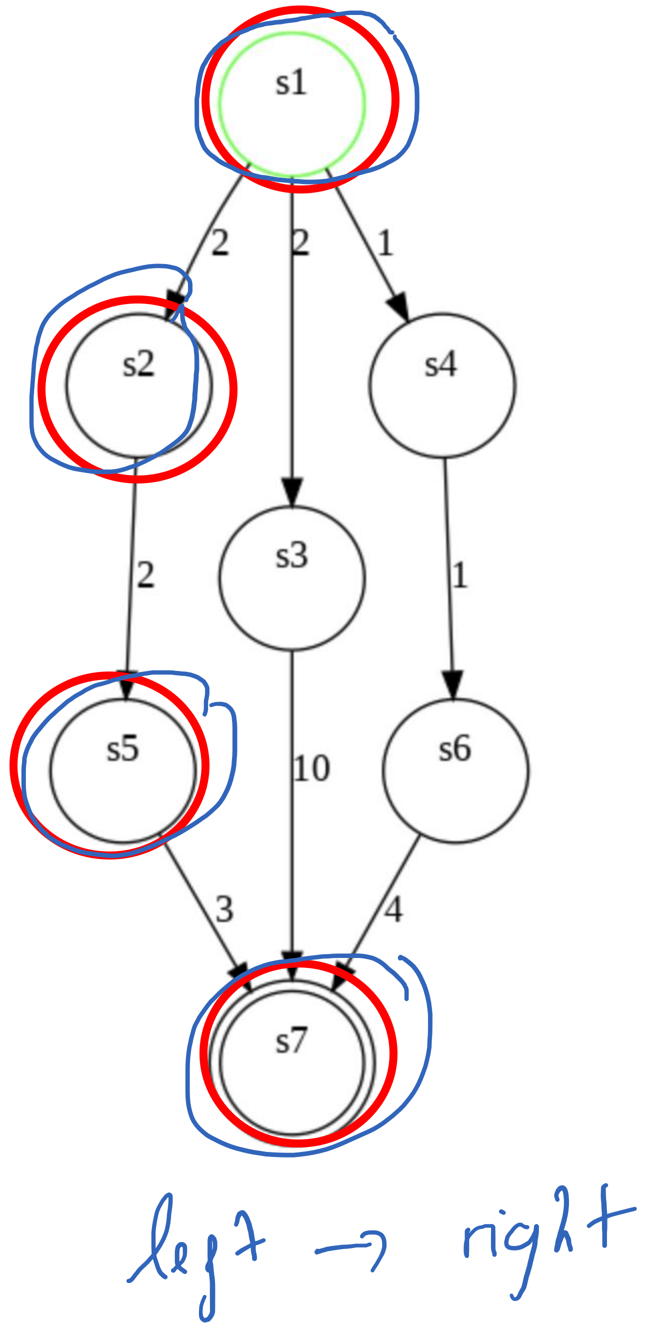
n0 = (s1, 0 ,None)

n1 = (s2, 2 ,0)

n2 = (s5, 4 ,1)

n3 = (s7, 7 ,2)

Solution: s1 -> s2 -> s5 -> s7

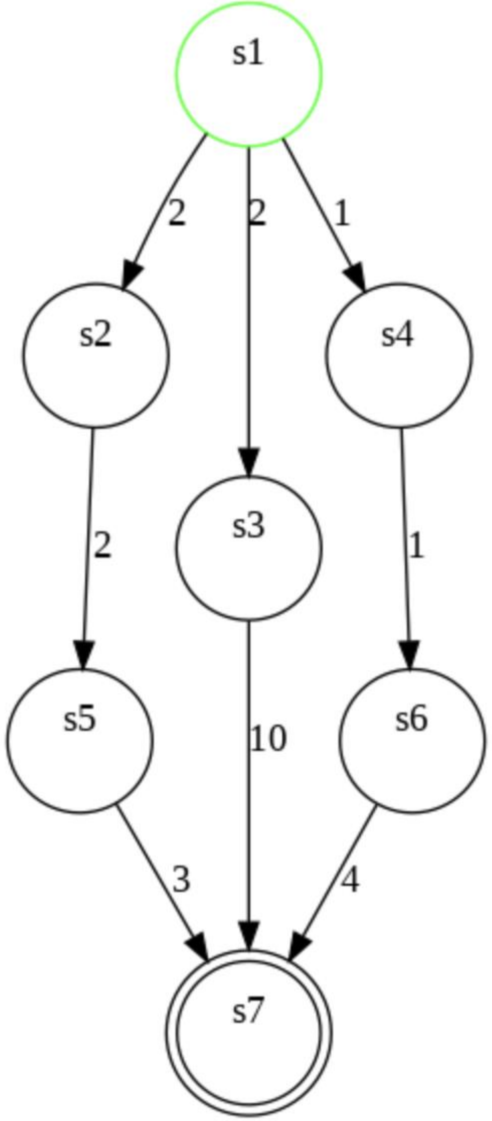


Question 2

Depth First Search (DFS)

	Open (Stack)	Close (Visited)
Iteration 1	n0 = <s1, 0, None>	
Iteration 2	n1 = <s2, 2, 0> n2 = <s3, 2, 0> n3 = <s4, 1, 0>	n0
Iteration 3	n4 = <s5, 4, 1> n2 = <s3, 2, 0> n3 = <s4, 1, 0>	n0, n1
Iteration 4	n5 = <s7, 7, 4> n2 = <s3, 2, 0> n3 = <s4, 1, 0>	n0, n1, n4
Iteration 5	n2 = <s3, 2, 0> n3 = <s4, 1, 0>	n0, n1, n4, n5

(s1, 0 ,None),
(s2, 2 ,0),
(s5, 4 ,1),
(s7, 7 ,2)

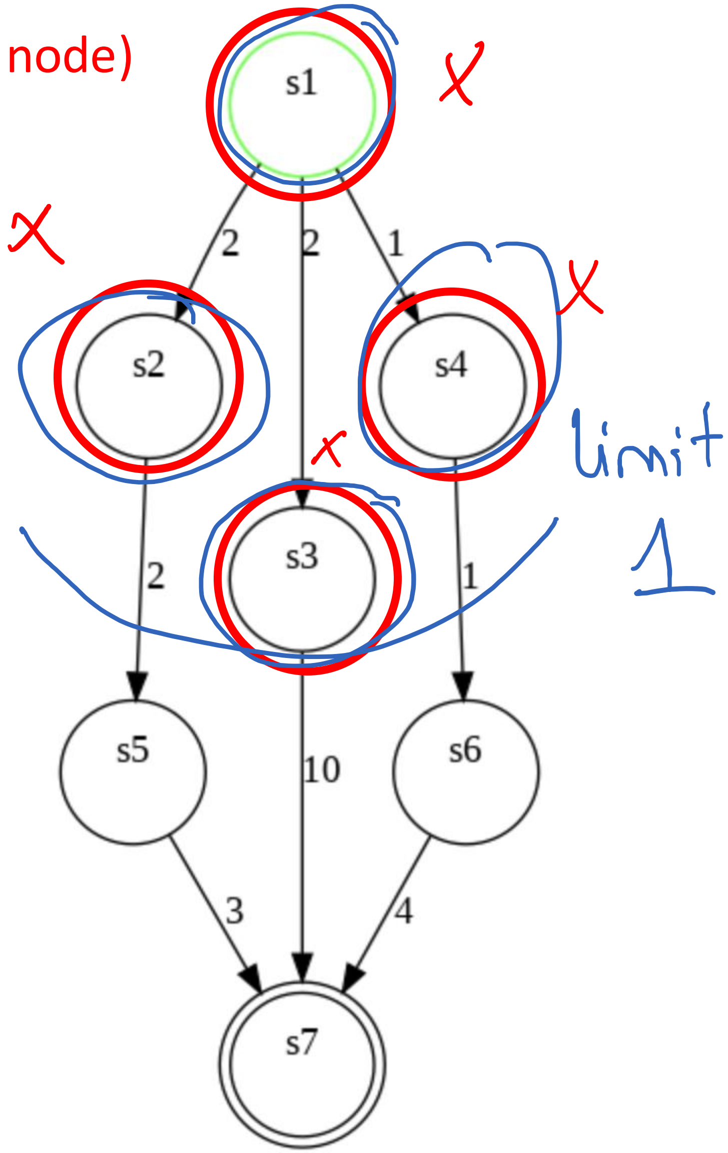


Problem 2

Search node n = (state, accumulated cost g(n), id of parent node)

Iterative Deepening (ID) DFS with a depth limit

Limit	Step	Open (Stack)	Close
0	1	n0 = (s1 , 0, None)	
	2		n0
1	3	n1 = (s1, 0, None)	
	4	n2 = (s2, 2, 1) n3 = (s3, 2, 1) n4 = (s4, 1, 1)	n1
	5	n3 = (s3, 2, 1) n4 = (s4, 1, 1)	n1, n2
	6	n4 = (s4, 1, 1)	n1, n2, n3
	7		n1, n2, n3, n4

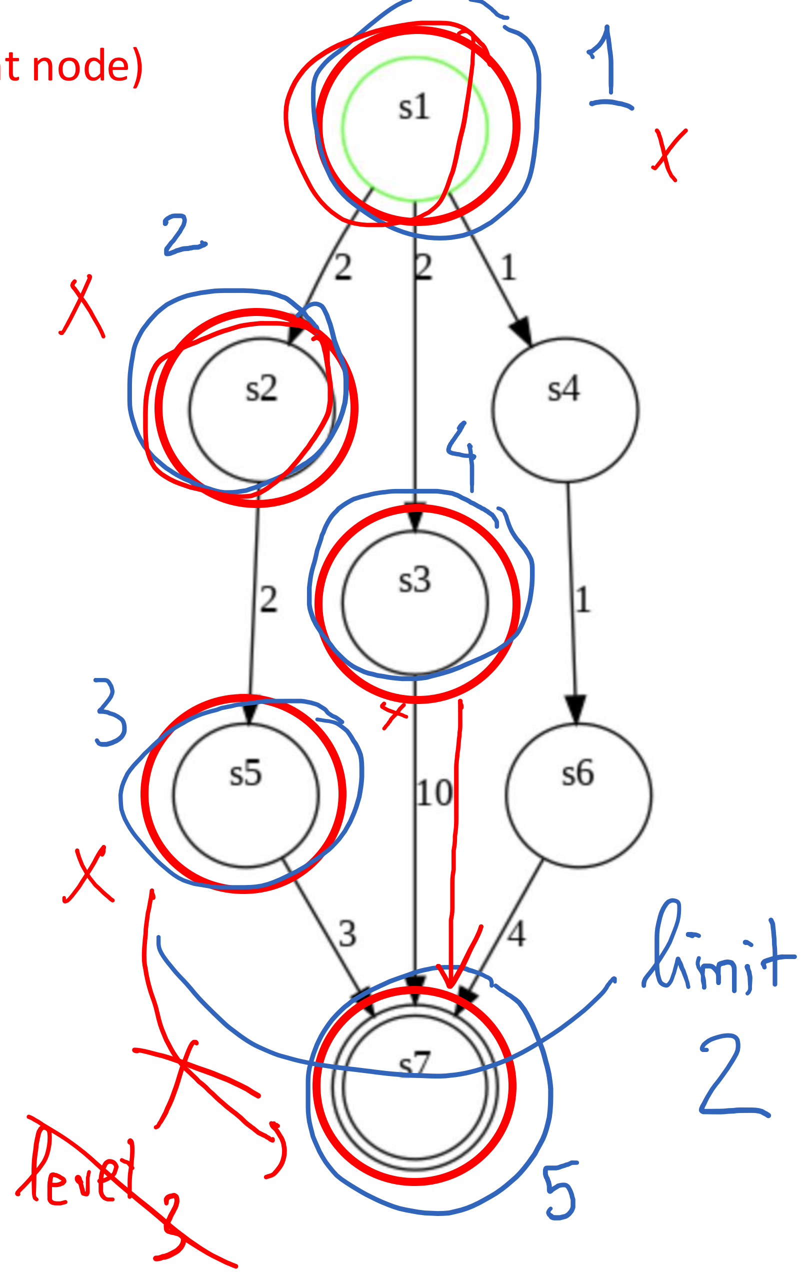


Problem 2

Search node n = (state, accumulated cost g(n), id of parent node)

Iterative Deepening (ID) DFS with depth limit

Limit	Step	Open (Stack)	Close
2	8	n5= (s1, 0, None)	
	9	n6 = (s2, 2, 5) n7= (s3, 2, 5) n8= (s4, 1, 5)	n5
	10	n9= (s5, 4, 6) n7= (s3, 2, 5) n8= (s4, 1, 5)	n5, n6
	11	n7 = (s3, 2, 5) n8 = (s4, 1, 5)	n5, n6, n9
	12	n10 = (s7, 12, 7) n8 = (s4, 1, 5)	n5, n6, n9, n7
	13	n8 = (s4, 1, 5)	n5, n6, n9, n7, n10



n0,
n1, n2, n3, n4,
n5, n6, n9, n7, n10

Problem 2

Search node n = (state, accumulated cost g(n), id of parent node)

Iterative Deepening (ID)

DFS with depth limit

Expansion node order

n0 = (s1, 0, None),

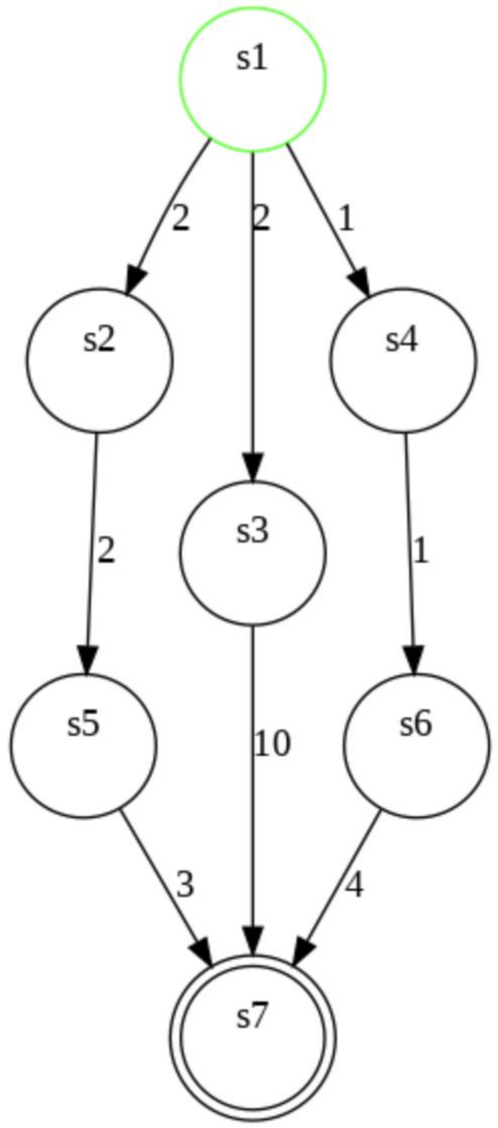
n1 = (s1, 0, None),
n2 = (s2, 2 ,1),
n3 = (s3, 2 ,1),
n4 = (s4, 1 ,1),

n5 = (s1, 0, None),
n6 = (s2, 2 ,5),
n9 = (s5, 4, 6),
n7 = (s3, 2 ,5)
n10 = (s7, 12, 7)

Expansion node order

n0,
n1, n2, n3, n4,
n5, n6, n9, n7, n10

Solution: s1 -> s3 -> s7



Problem 2

Q2: What is the actual optimal solution?

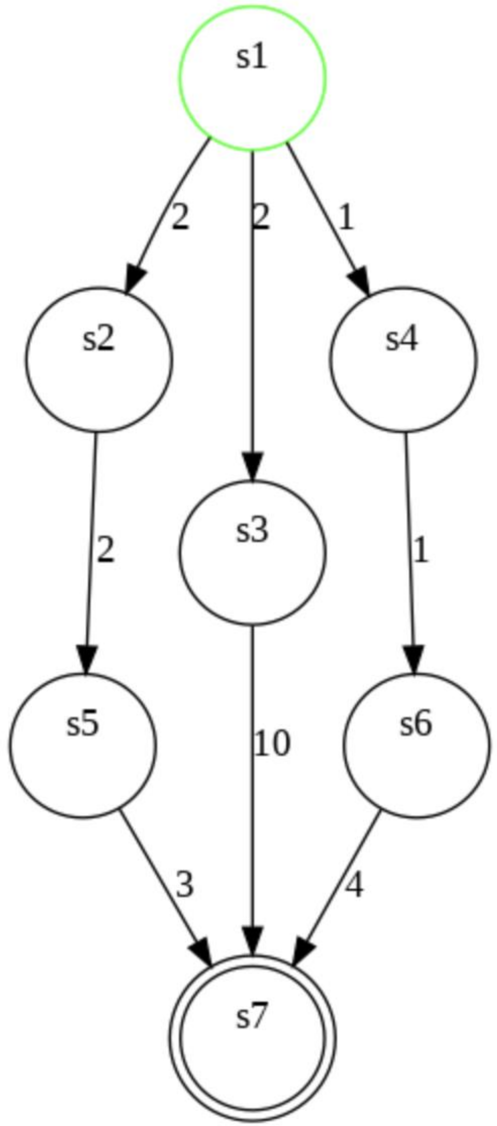
s1 -> s4 -> s6 -> s7

Q3: Explain under which conditions the algorithms guarantee optimality?

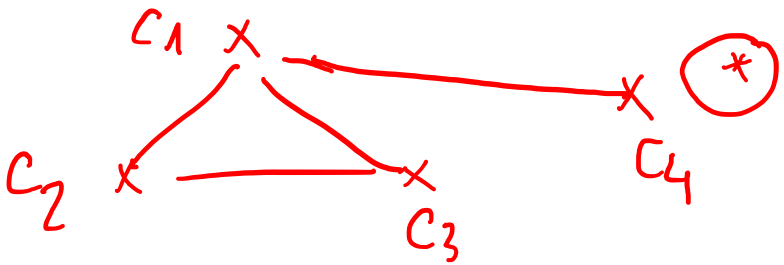
BrFS and ID will be optimal if the costs are equal, such as, all cost are 1

Q4: Can any of the previous algorithms be adapted to account for $g(n)$ ***in order to make it optimal?***

Dijkstra. Expanding the node with lowest accumulated cost, instead of the lowest depth.



Problem 3



Describe a simple example of *Travelling Salesman Problem* along with its corresponding **State Space Model**.

Definition should be brief, clear, and *compact* (*compact* means using mathematical notation to define sets, i.e. $S = \{x | x \in V\}$ to define that there are as many states as elements in the set V, and pseudo-code, i.e. to define the transition function.)

1. State space S
2. Initial state $s_0 \in S$
3. Set of goal states $S_G \subseteq S$
4. Applicable actions function $A(s)$ for each state $s \in S$
5. Transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
6. Cost of each action $c(a)$ for $a \in A(s)$

Note for the TSP :

- A city can be visited more than once
- You can start at any city
- The goal state is to visit all cities and no need to go back to the starting city

Hint: Given

- V = a set of cities
- v_{start} = a starting city location
- E = a set of edges specifying if there is an edge between two cities <v1, v2>
- V' = a set of cities that have been visited

Problem 3

a state = <current city, a set of visited cities>

an edge/action= <current city, next city>

Hint: Given

- V = a set of cities
- v_{start} = a starting city location
- E = a set of edges specifying if there is an edge between two cities <v1, v2>
- V' = a set of cities that have been visited

Initial state $s_0 = < v_{start}, \{v_{start}\} >$

Goal state $S_G = \{< v_{current}, V > | \, v_{current} \in V\}$

$V' = V$

State $S = \{< v_{current}, V' > | \, v_{current} \in V \wedge V' \subseteq V\}$

Action $A(< \underbrace{v_{current}, V'}_{\text{curr state}} >) = \{< v_{current}, v_{next} > | < v_{current}, v_{next} > \in E\}$

Transition $f(< \underbrace{v_{current}, V'}_{\text{curr state}} >, < \underbrace{v_{current}, v_{next}}_{\text{action}} >) = < \underbrace{v_{next}, V' \cup \{v_{next}\}}_{\text{next state}} >$

$c(< v_{current}, v_{next} >)$ = cost$(< v_{current}, v_{next} >)$