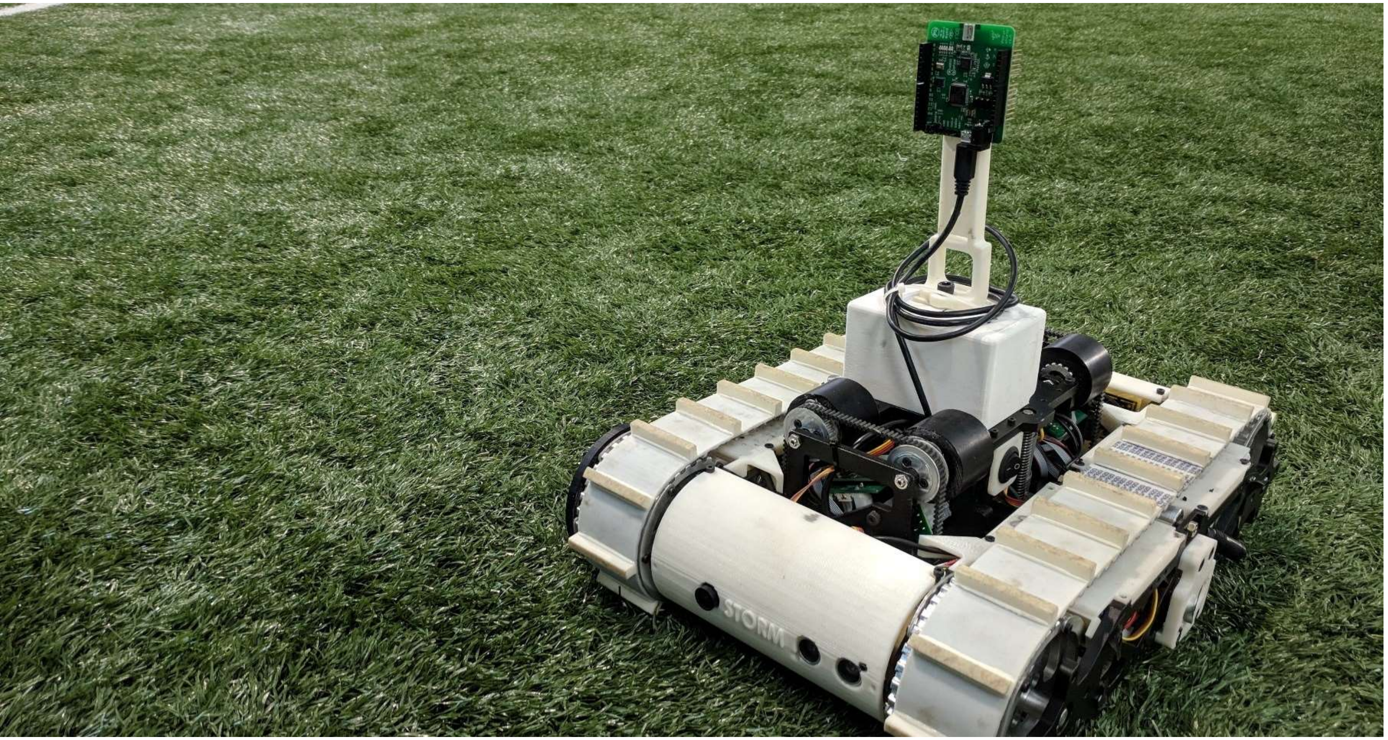


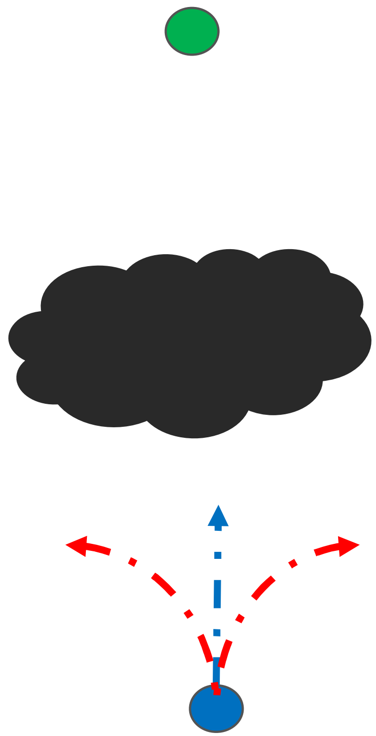
ED 5215
INTRODUCTION TO MOTION PLANNING

Instructor: Bijo Sebastian

MOTIVATION

Reflex agents

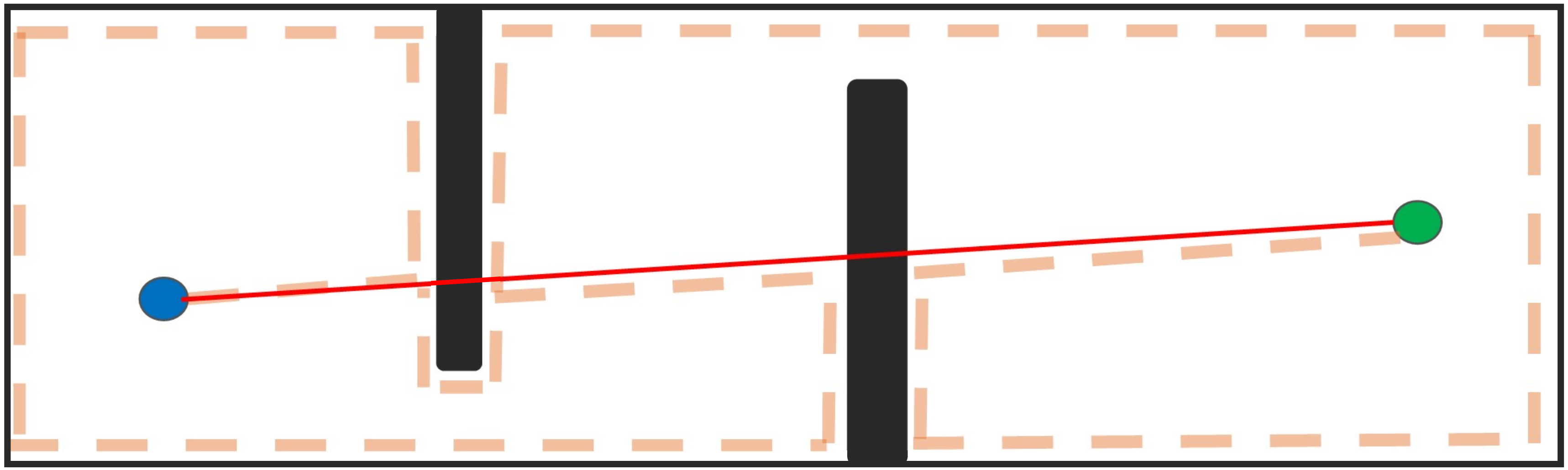
- Example Bug algorithms, robot with Go-to-goal and avoid-obstacle controllers
- Choose action based on current percept (and maybe memory)
- May have memory or a model of the world's current state
- Do not consider the future consequences of their actions
- Consider how the world IS (which could change if the world is not static)



MOTIVATION

Assuming a model of the world (map) is available, the solution

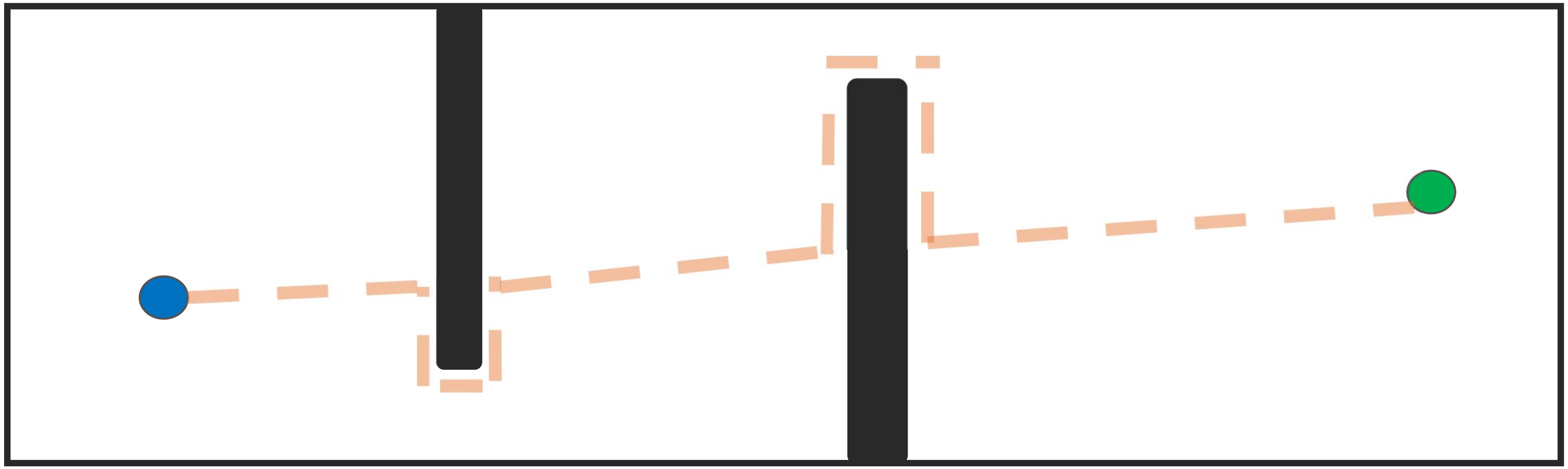
by Bug -2 algorithm is not the shortest path → NOT rational



MOTIVATION

Planning agents:

- Ask “what if”
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal (test)
- Consider how the world WOULD BE (if the world is not static)



MOTIVATION

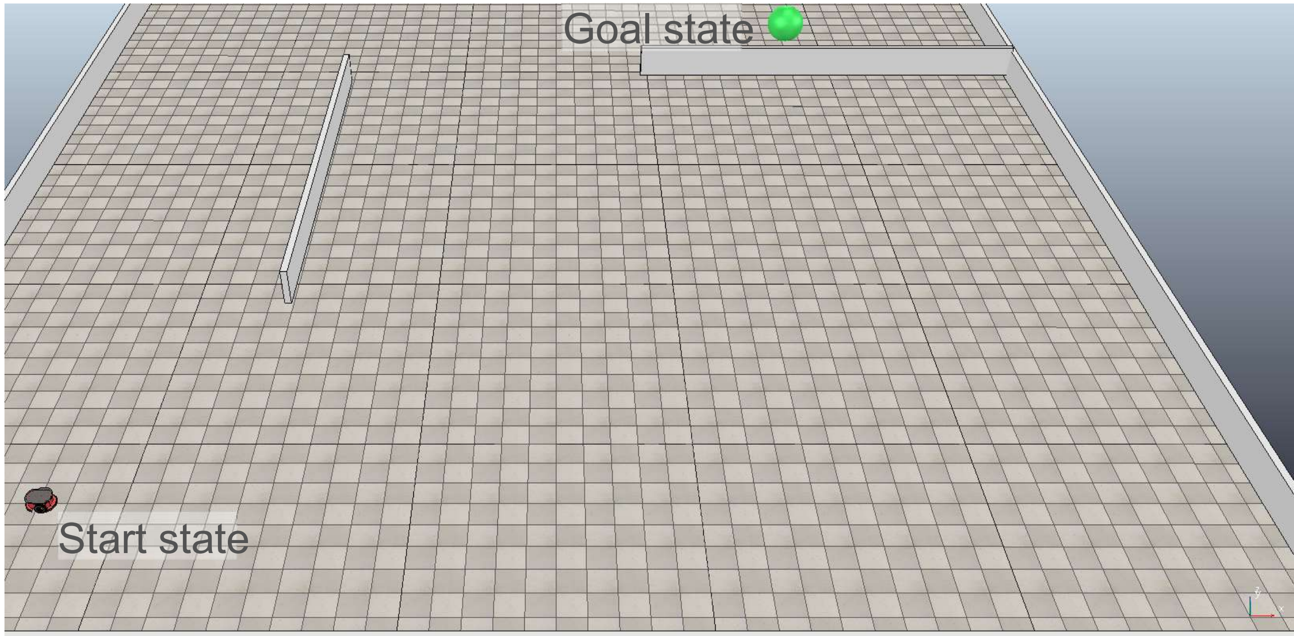
Search algorithms

- Algorithms that allow for rational decision making by planning ahead
- Motivation: Memory and simulation are key to rational decision making

Uninformed search methods:

- Depth first search
- Breadth first search
- Uniform cost search

SEARCH PROBLEM



Consider diff. drive robot, on an environment

with static obstacles

Elements of a search problem

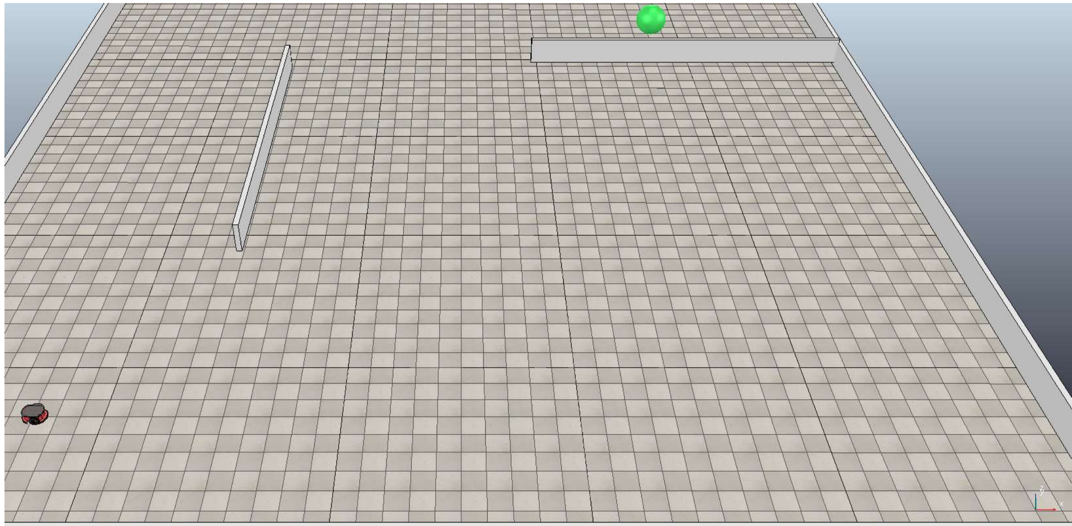
1. State space of the system :

x, y position

- Simplified model of the world: grid map
- Discrete or continuous
- Usually bounded
- Start state, Goal state

(x, y, θ)

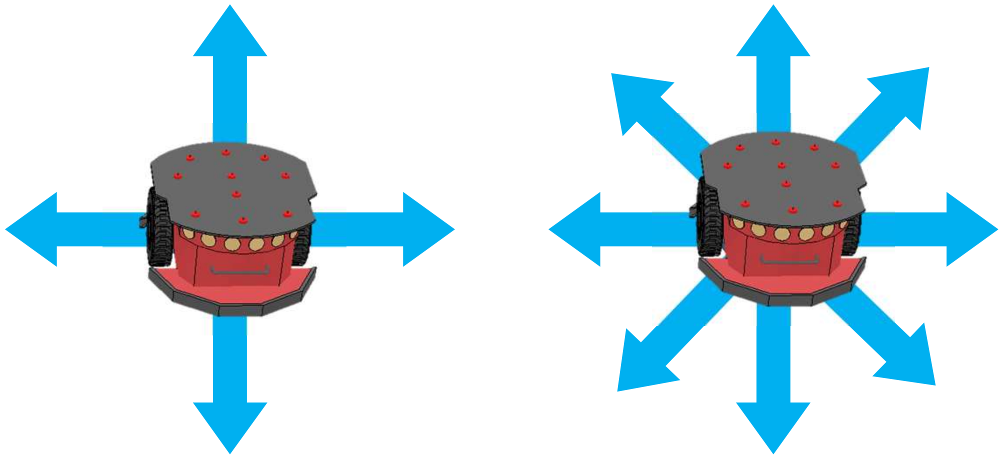
SEARCH PROBLEM



Elements of a search problem

2. Set of possible actions: Move to the four neighboring tiles or eight neighboring tiles

- Easy to implement when world is modelled as grid cells

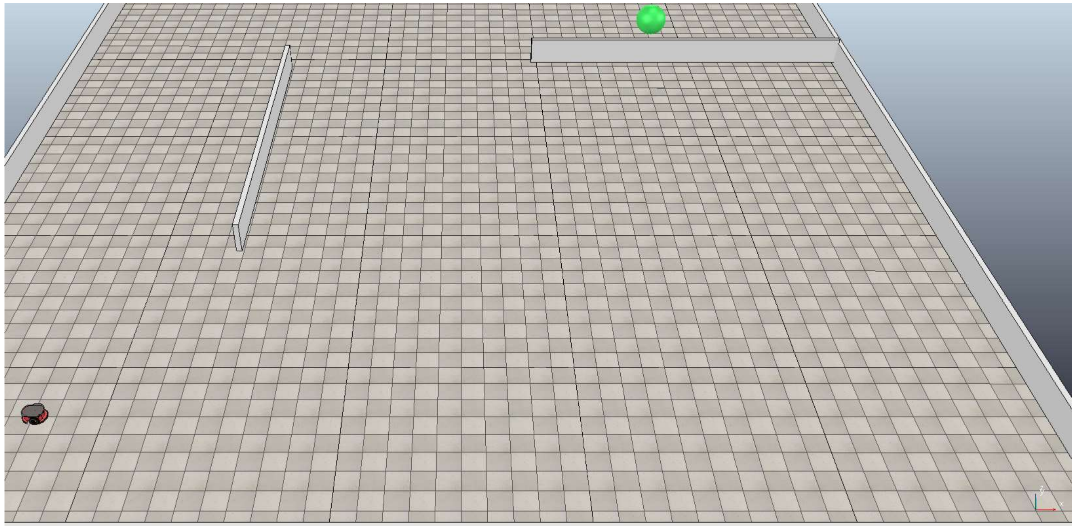


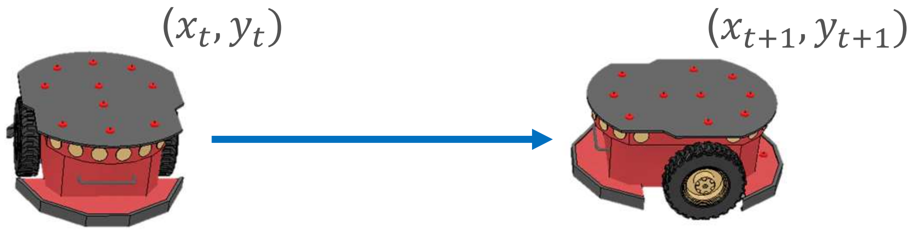
SEARCH PROBLEM

<u>Elements of a search problem</u>

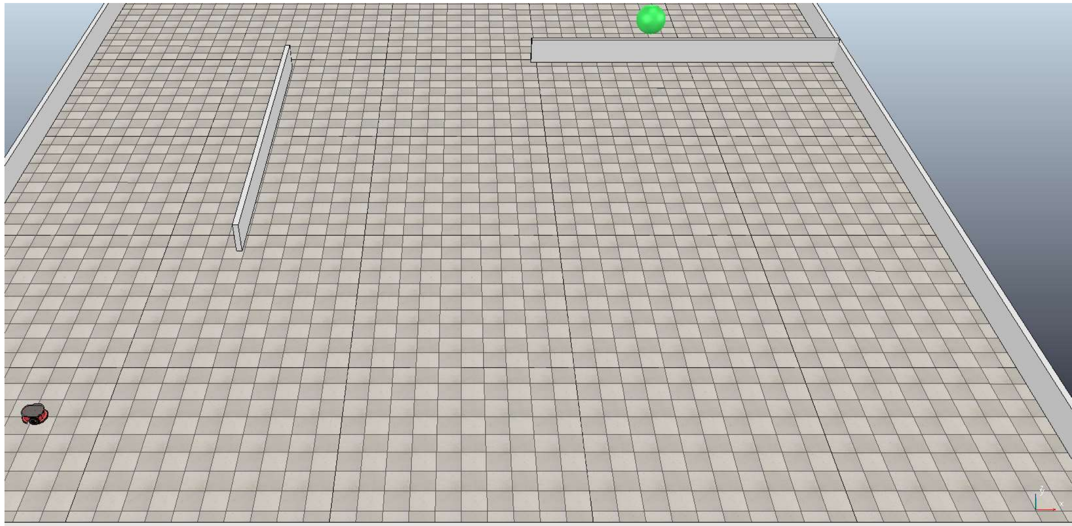
3. <u>State expansion function (successor function)</u>

- Should be able to distinguish between feasible and infeasible motion (such as when a collision might happen)
- Should have a cost associated with feasible motion (such as distance travelled, time taken, energy spent or a combination of these)





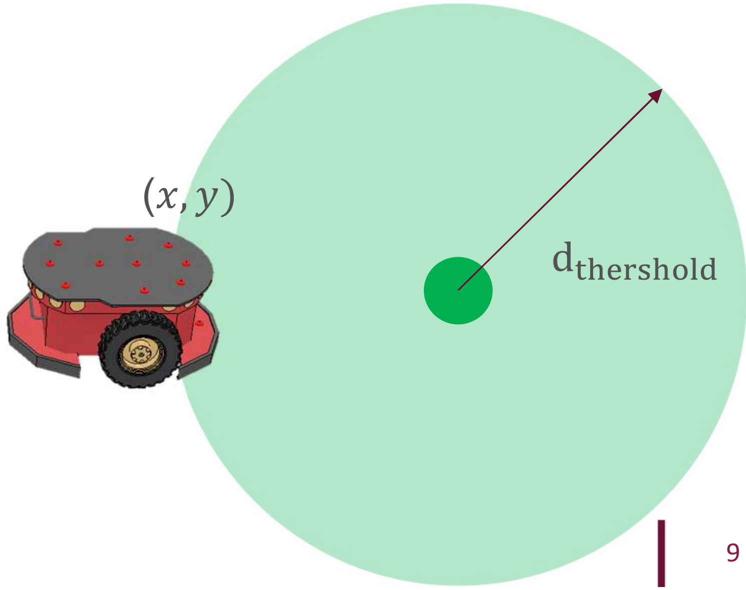
SEARCH PROBLEM



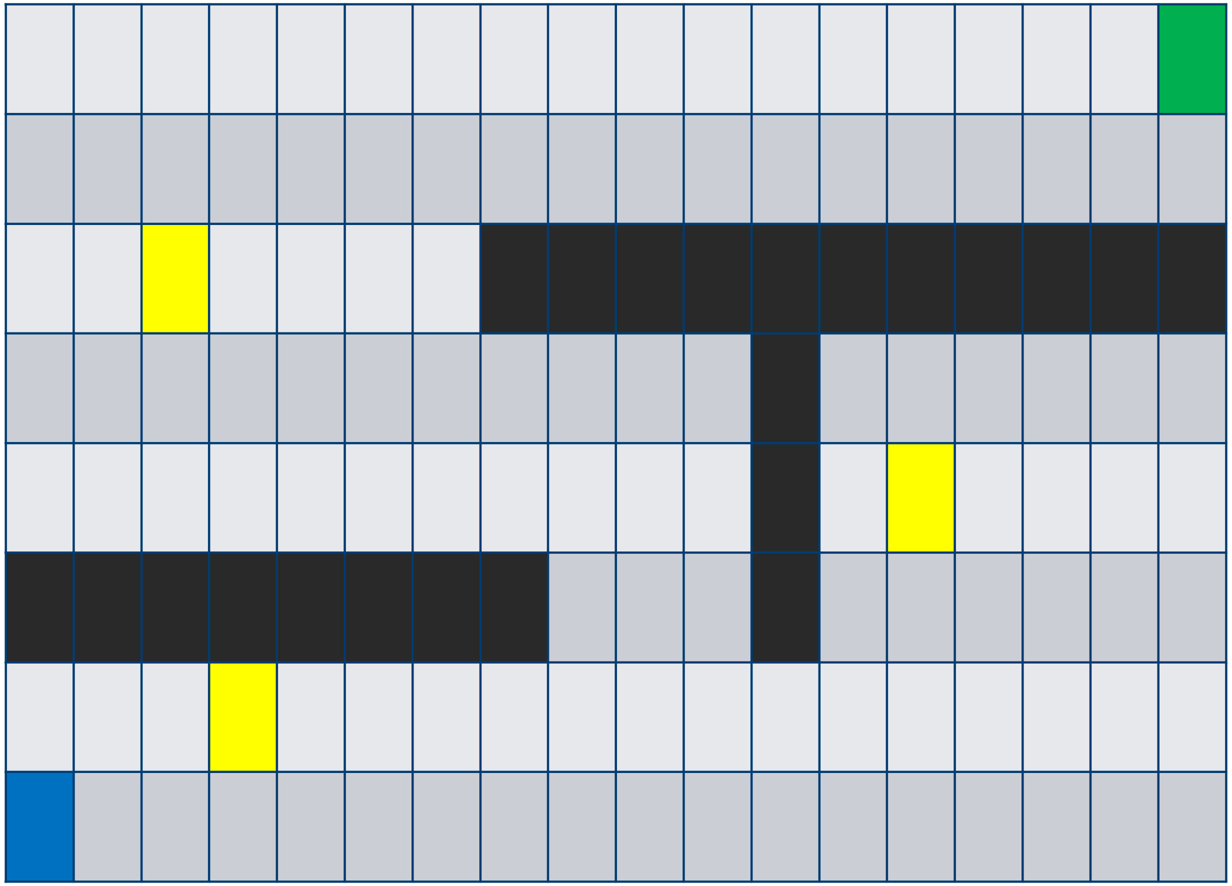
Elements of a search problem

4. Goal test

- Example if robot is within threshold distance of the goal point/goal state (in cases where orientation is also important) declare success



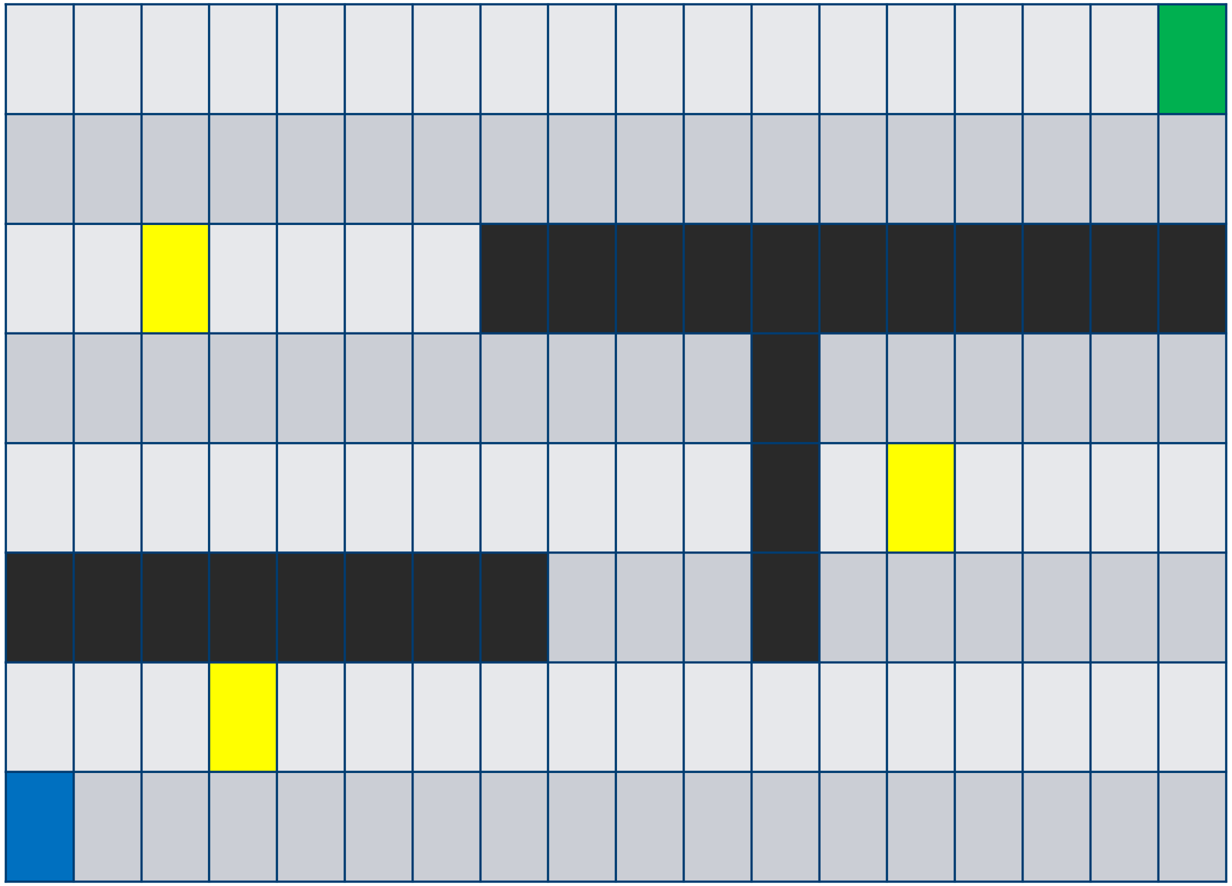
SEARCH PROBLEM



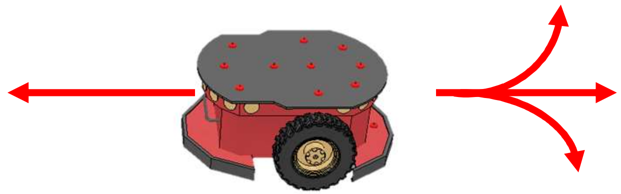
Question: The robot starts at the blue cell and want to get to the green cell, after visiting each of the yellow cell, along the shortest path. Assume the robot can move to four neighboring cells.

What would be the states of the robot that you need to keep track of?

SEARCH PROBLEM



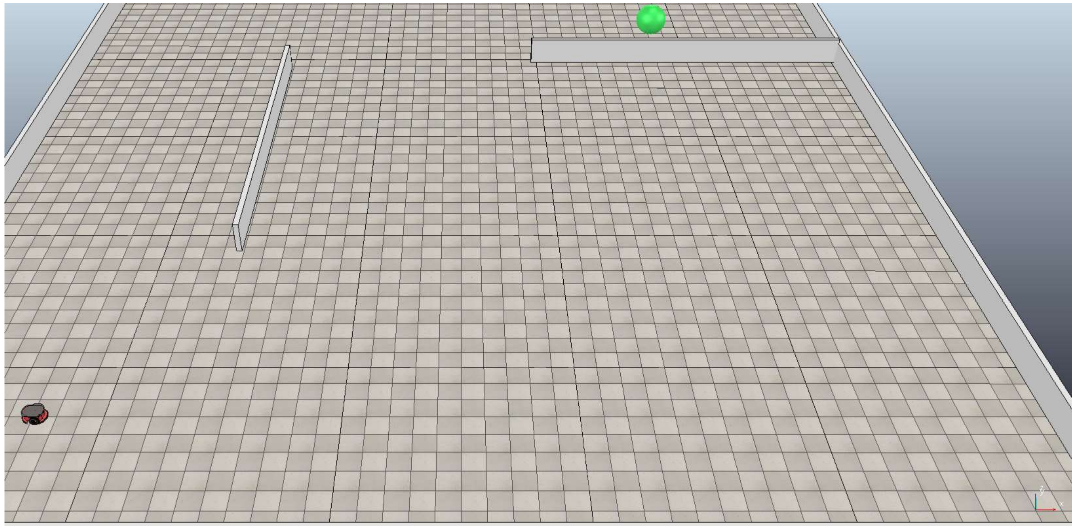
Question: In the previous problem, assume the robot motion is constrained as follows:



What would be the states of the robot that you need to keep track of?

SEARCH PROBLEM

Implementation



Problem formulation(specific to each application)

- States
- Actions
- State expansion
- Goal test

Application independent planning algorithm

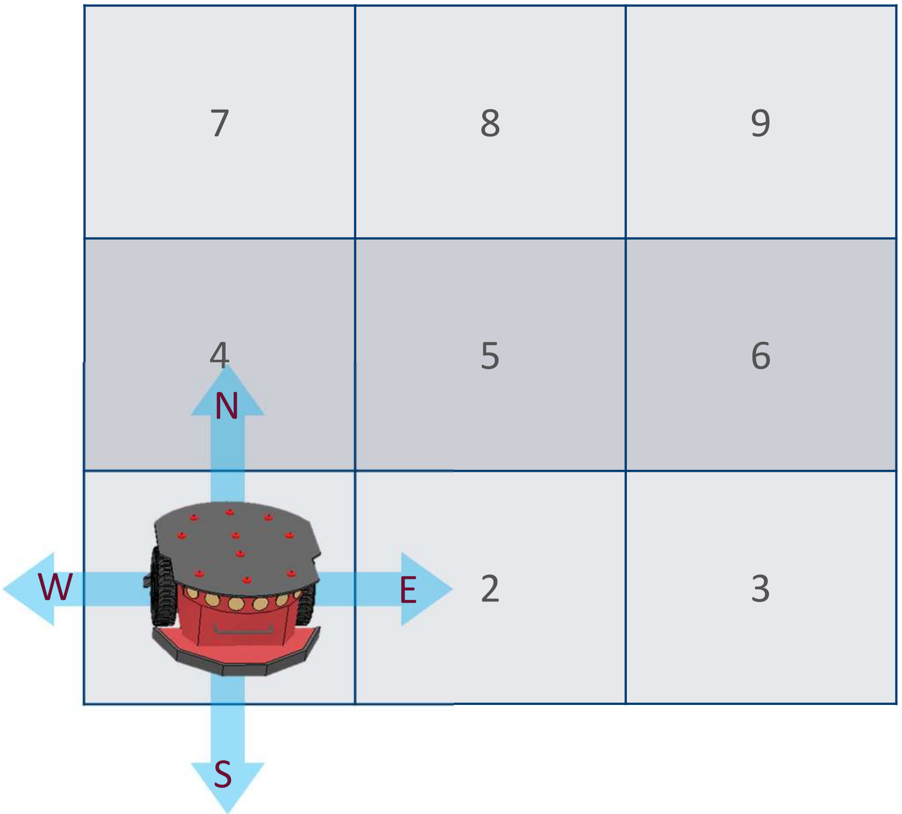
Library of search based or sampling-based planning algorithms (e.g., OMPL, Robotics Toolbox)

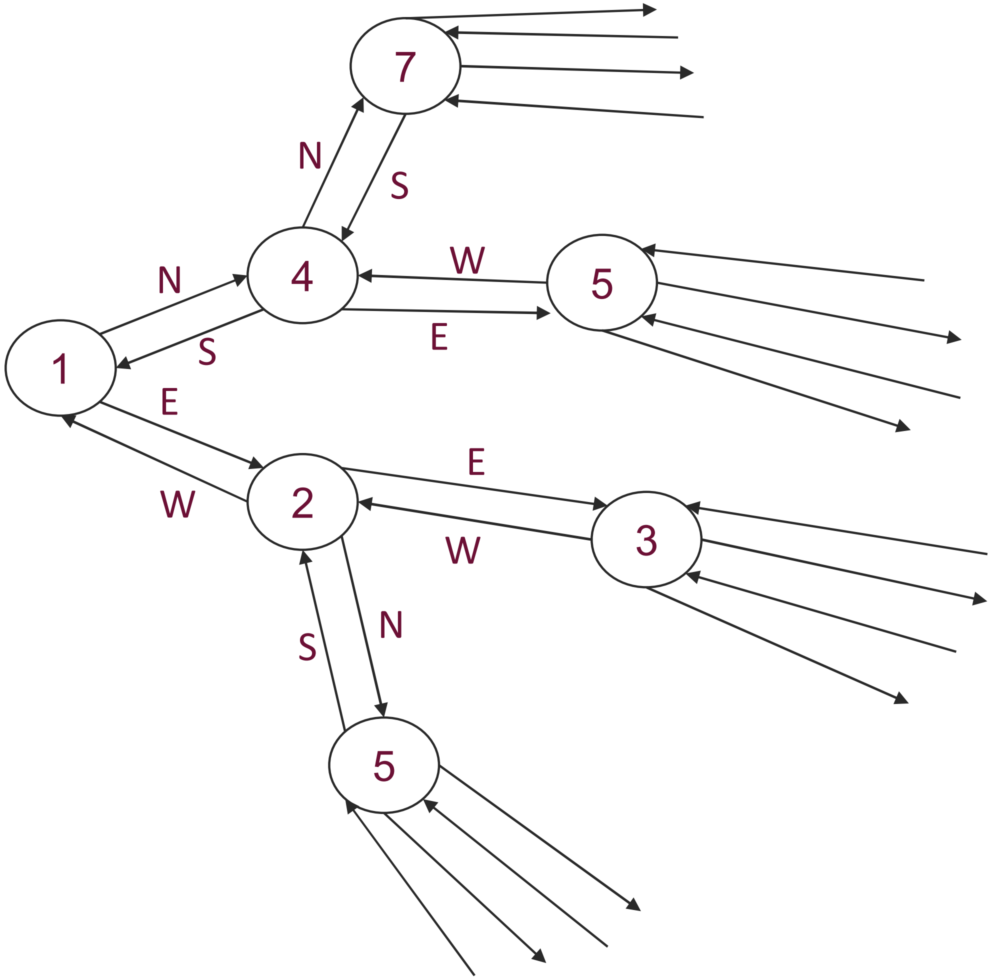
Solution

A sequence of actions (a plan) which takes the robot (agent) from start state to a goal state

GRAPHS

State space graph

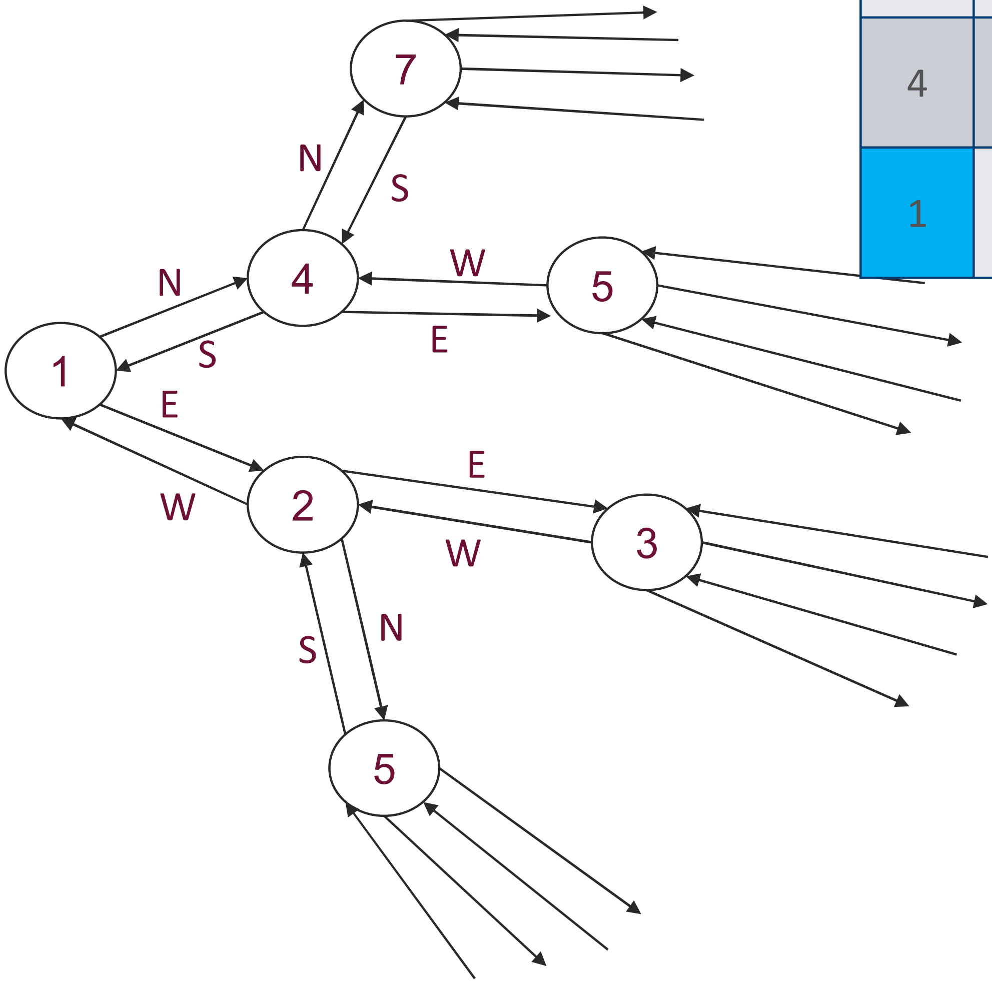




GRAPHS

State space graph: A mathematical representation of the state transitions

- Each node stores the relevant states of the robot
- Arcs represent actions leading to new states
- Goal nodes: Set of nodes satisfying the goal test (could be a single one)
- Each state node is unique



7	8	9
4	5	6
1	2	3

Question: Can we use graphs to solve search/planning problems?

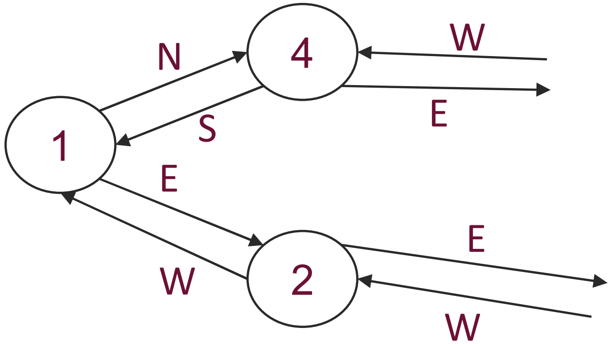
GRAPHS V/S TREES

7	8	9
4	5	6
1	2	3

State space graph:

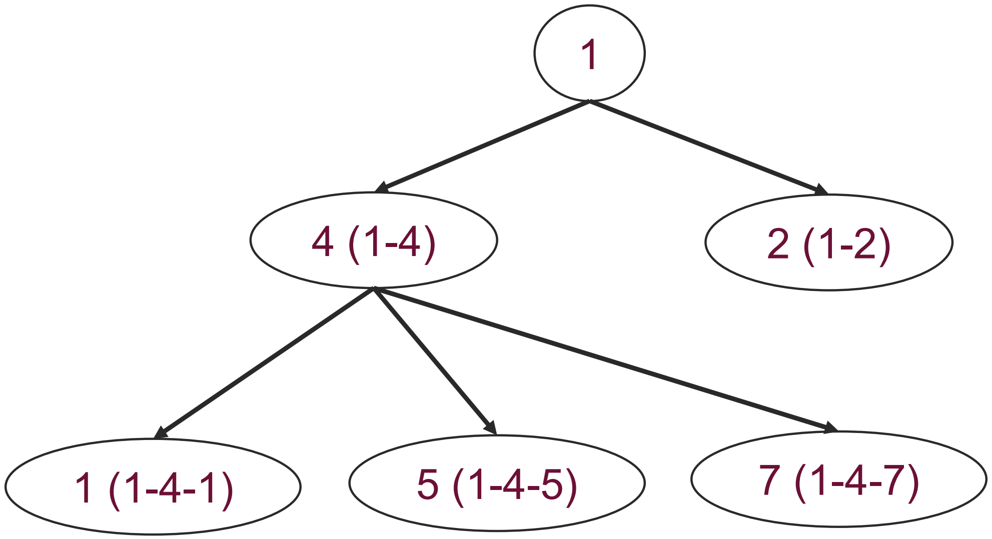
In a graph with doubly connected nodes, there are many ways to get to each node from start node

Question: What are the ways to get to node 4?



Search tree:

In a search tree, every node shows a state but also a plan on how to get to that state from the start node

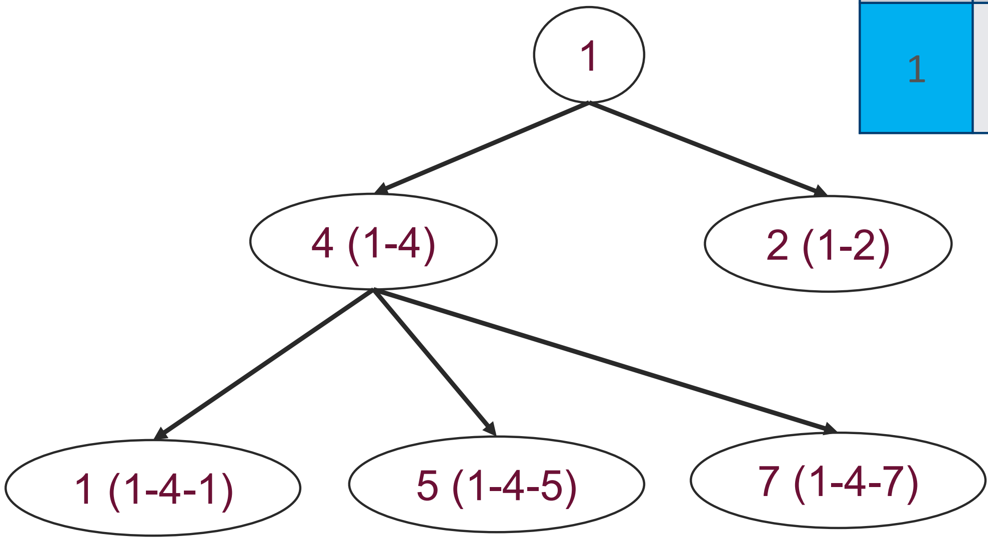


SEARCH TREE

7	8	9
4	5	6
1	2	3

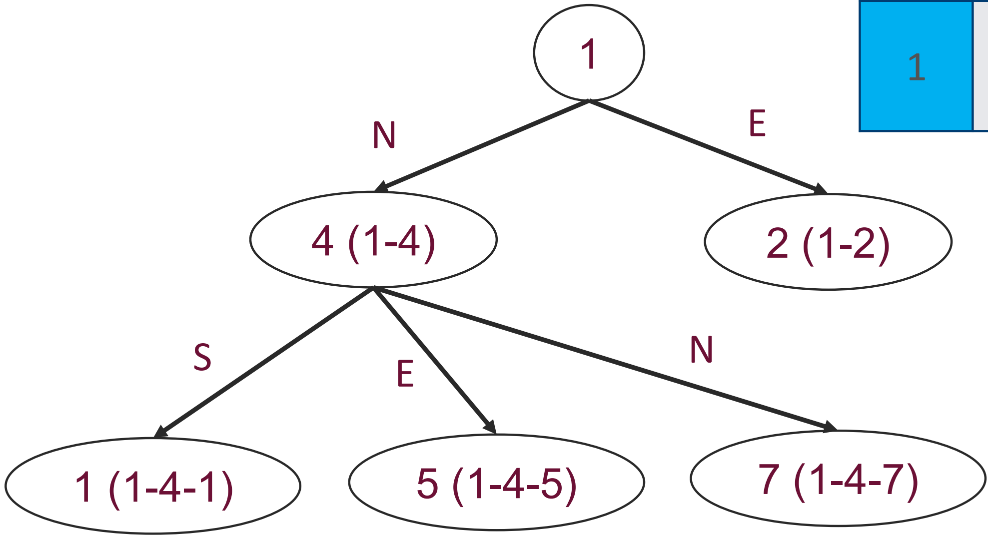
Question: Do we need to create/store the full tree in memory to solve the planning problem?

No, An efficient search algorithm must be able to come up with a plan to get to goal state by expanding minimum number of nodes/states



TREE SEARCH ALGORITHM

- Begin with start state in Fringe: { 1 }
- Remove a state from Fringe and perform all possible actions to get to new states
- Add new states to Fringe: { 4, 2}
- Remove a state from Fringe and perform all possible actions to get to new states
- Remove a state from Fringe and perform all possible actions to get to new states
- Add new states to Fringe: {2,5,7}

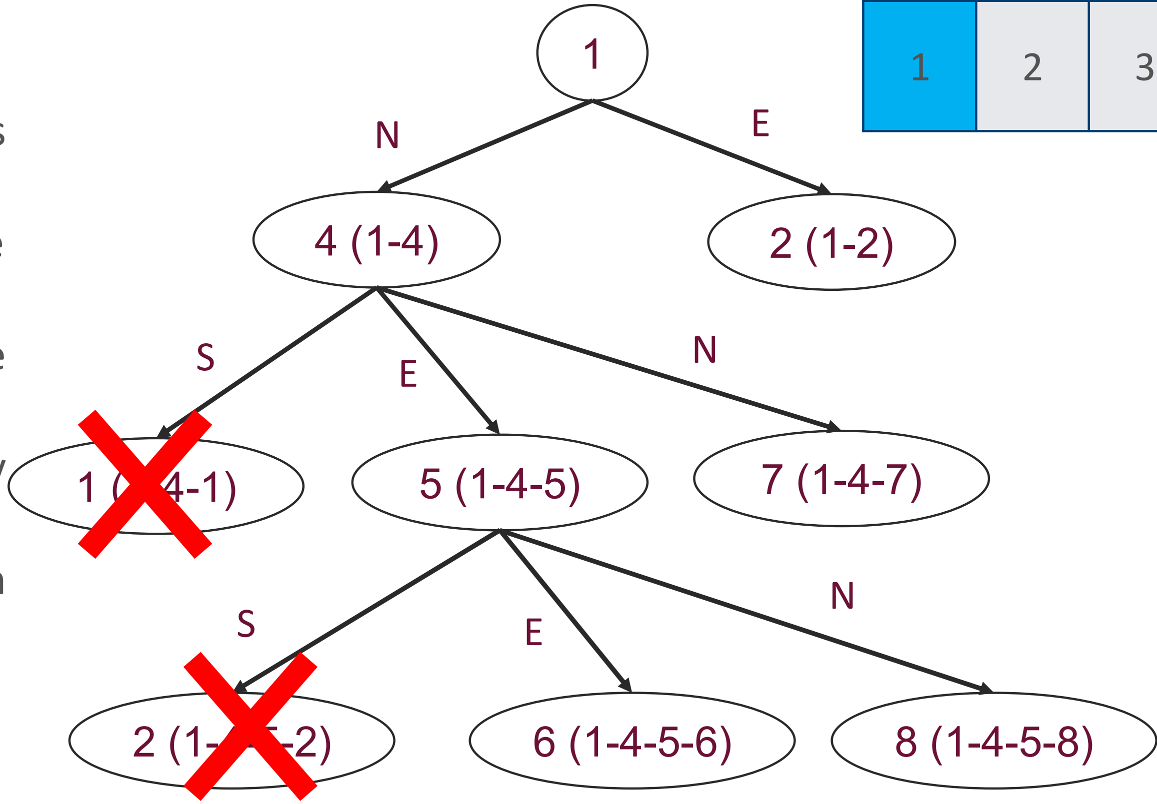


TREE SEARCH ALGORITHM

Two things that are specific requirements from a planning perspective

- To avoid infinite loops do not add state that has already been removed (popped) from fringe once
- Explicitly check fringe for duplicates before adding nodes to it. If duplicate exist store only the one that can be reached via shorter path from start node, assuming each step is unit cost

7	8	9
4	5	6
1	2	3



TREE SEARCH ALGORITHM

7	8	9
4	5	6
1	2	3

Pseudo code:

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    Initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
        except:
            states that has already been removed (popped) from fringe once
            If duplicate state:
                Store the one that can be reached via shorter path from start node

        end
```

Based on the strategy used in choosing node/state from fringe, we have different planning algorithms like breadth first, depth first etc.

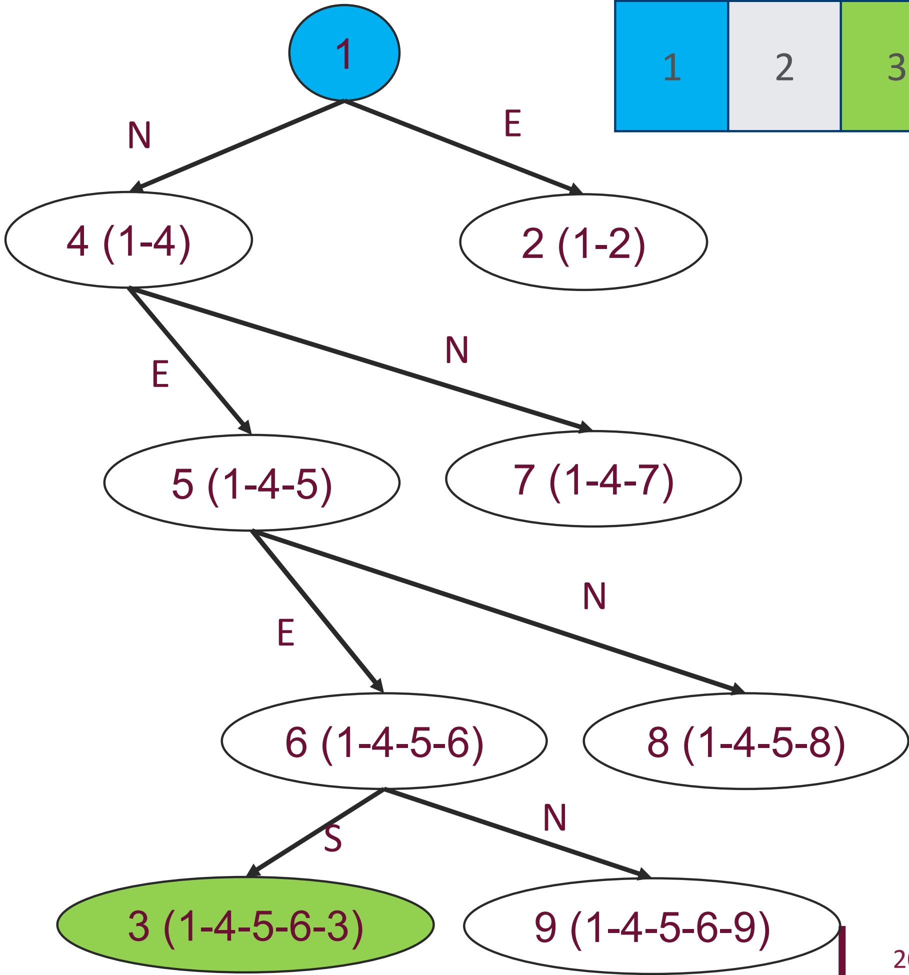
DEPTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

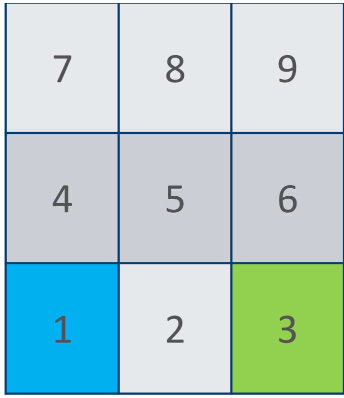
Strategy: Expand the deepest node first

- Fringe { 1 } → pop 1
- Fringe: { 2,4} → pop 4
- Fringe: {2,5,7} → pop 5
- Fringe: {2,7,6,8} → pop 6
- Fringe: {2,7,8,3,9} → pop 9
- Goal state reached

Path is 1(N)-4(E)-5(E)-6(S)-3

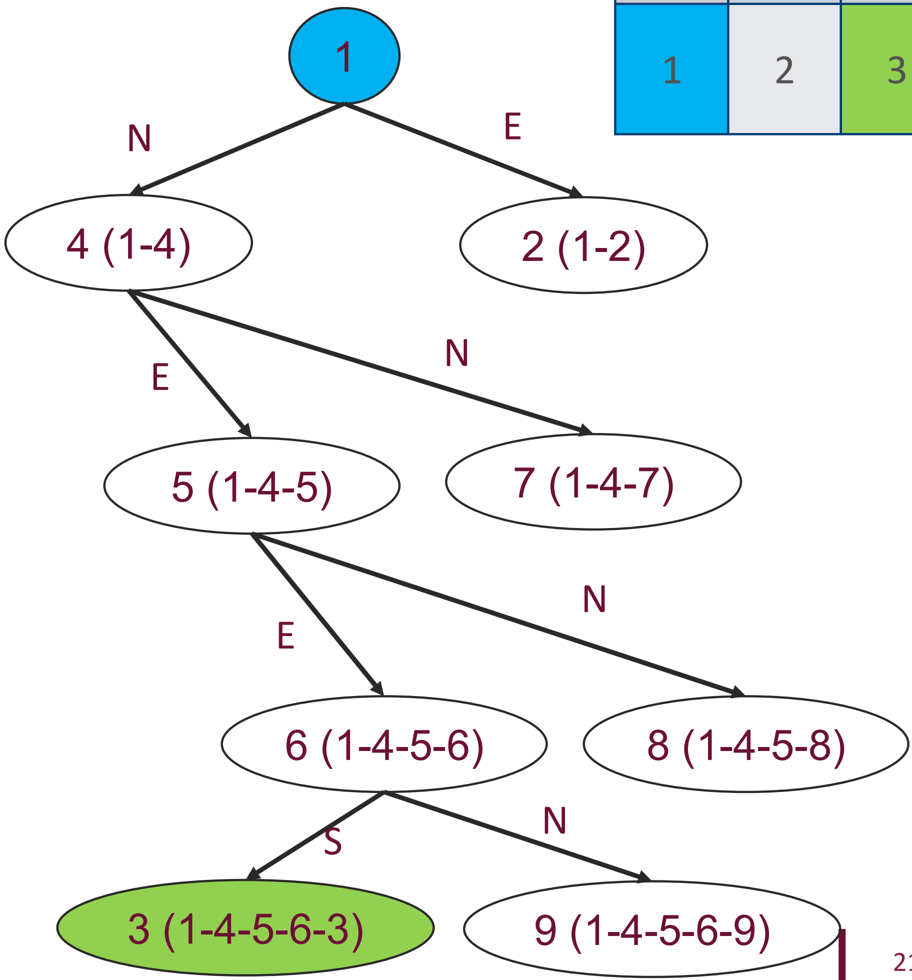


DEPTH FIRST SEARCH



Questions:

- Is Depth first complete?
- Is Depth first optimal?



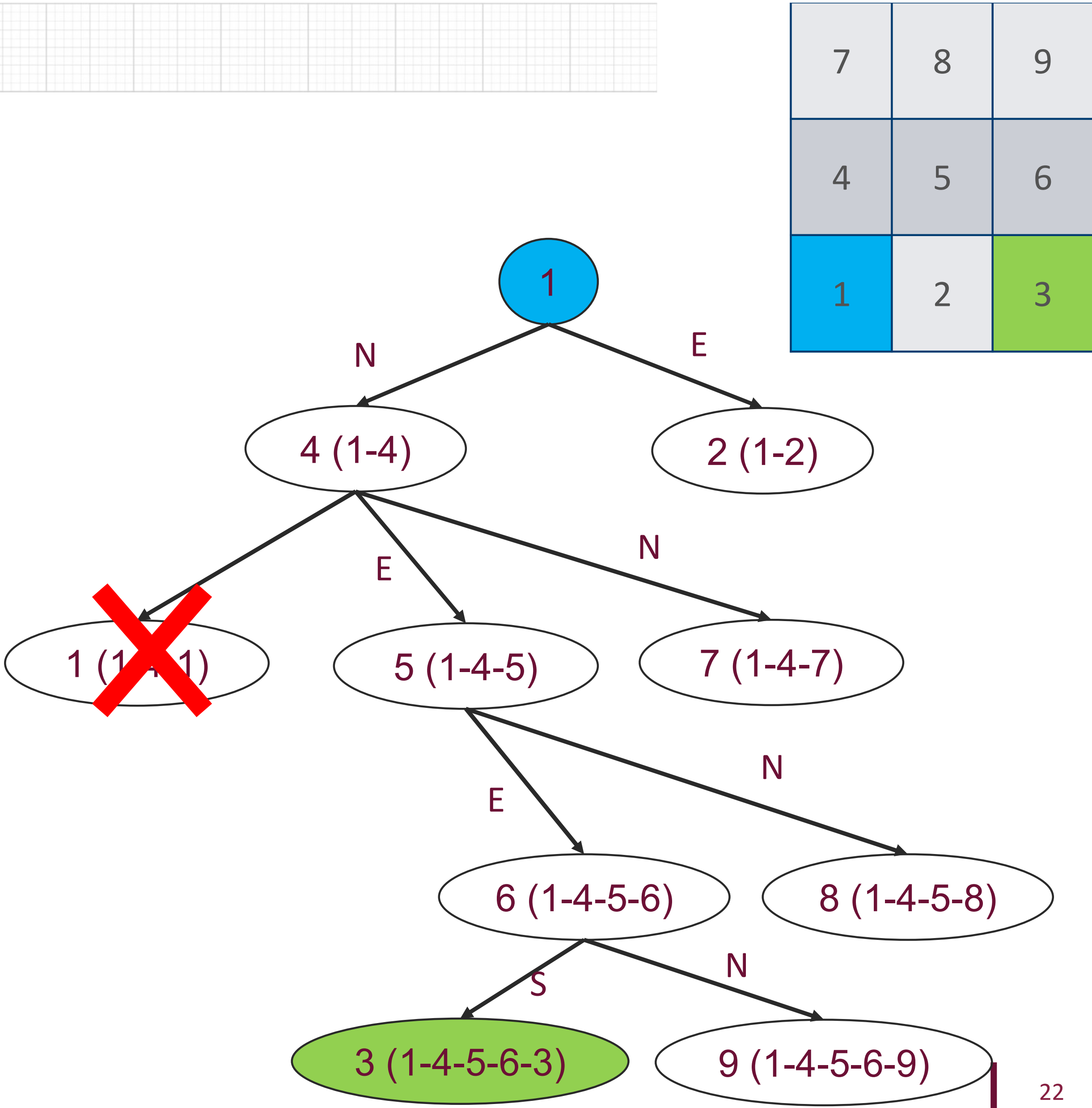
DEPTH FIRST SEARCH

Questions:

- Is Depth first complete?

Yes, provided we ensure we do not add state that has already been removed (popped) from fringe once

If not, this could lead to infinite loops

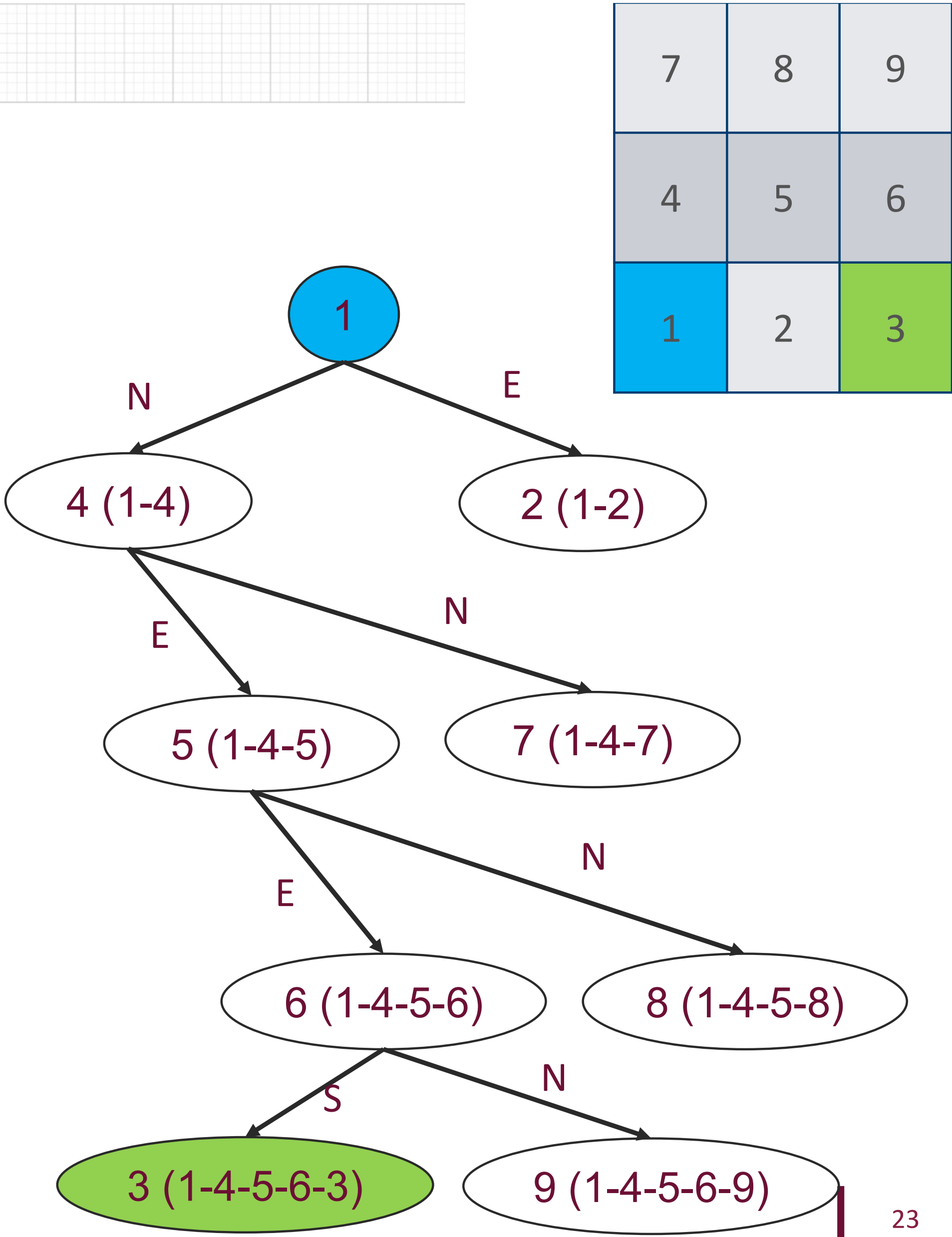


DEPTH FIRST SEARCH

Questions:

- Is Depth first optimal?

No, path 1(E)-2(E)-3 would have been shorter assuming unit cost for each step. But we never expanded that



BREADTH FIRST SEARCH

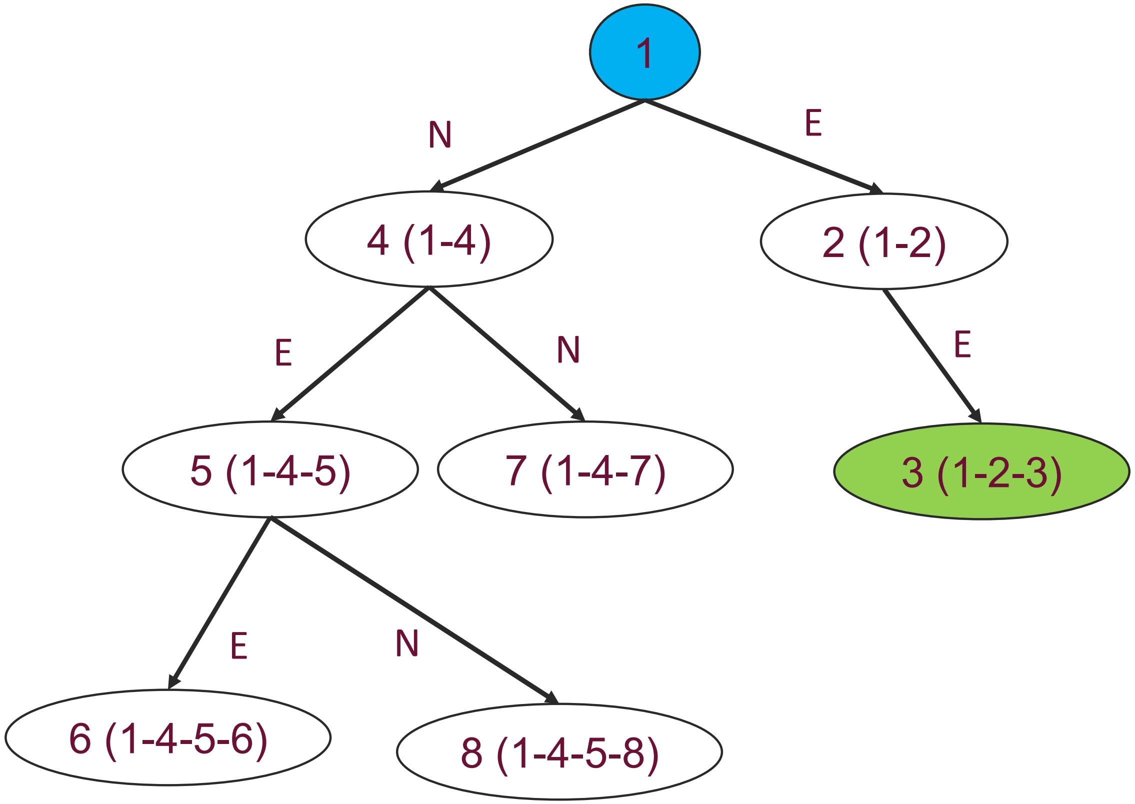
7	8	9
4	5	6
1	2	3

Strategy: Expand the shallowest node first

- Fringe { 1 } → pop 1
- Fringe: { 2,4} → pop 4
- Fringe: {2,5,7} → pop 2
- Fringe: {5,7,3} → pop 5
- Fringe: {7,3,6,8} → pop 7
- Fringe: {3,6,8} → pop 3

Goal state reached

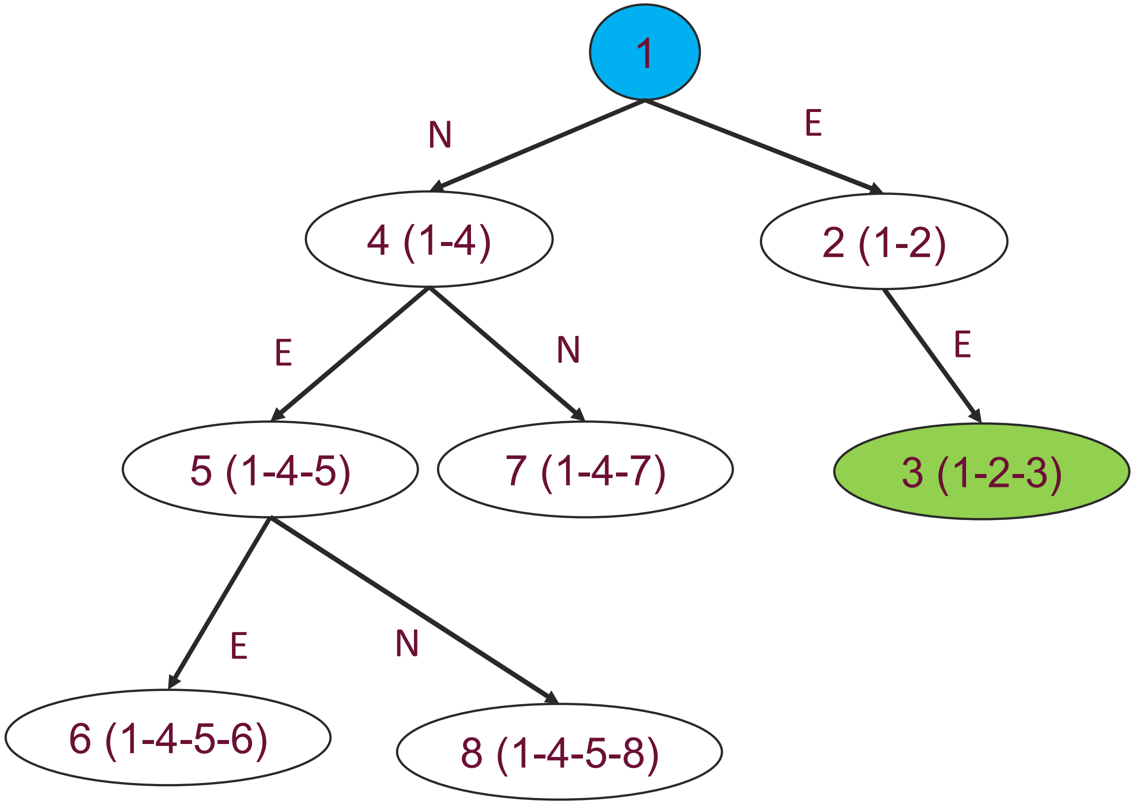
Path is 1(E)-2(E)-3



BREADTH FIRST SEARCH

Questions:

- Is Breadth first complete?
- Is Breadth first optimal?



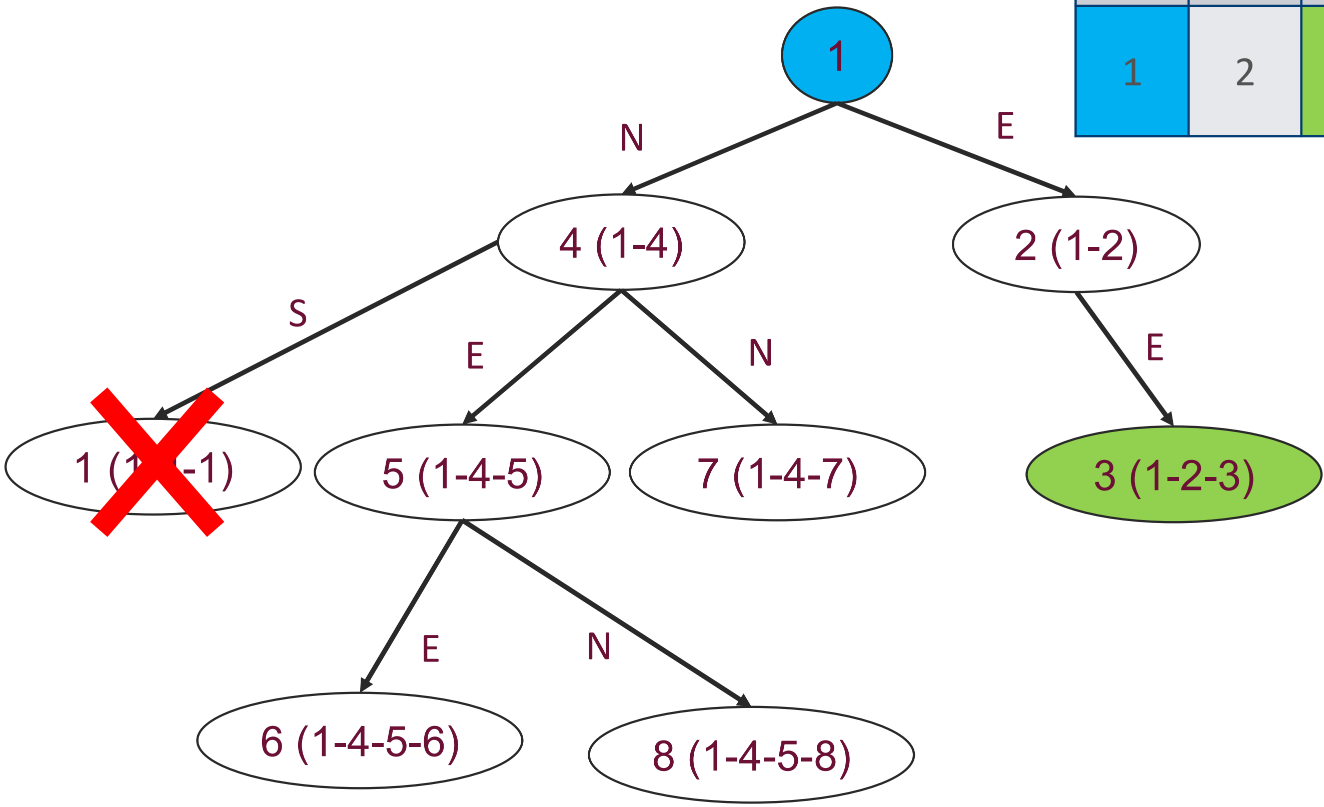
BREADTH FIRST SEARCH

Questions:

- Is Breadth first complete?

Yes, provided we ensure we do not add state that has already been removed (popped) from fringe once

If not, this could lead to infinite loops

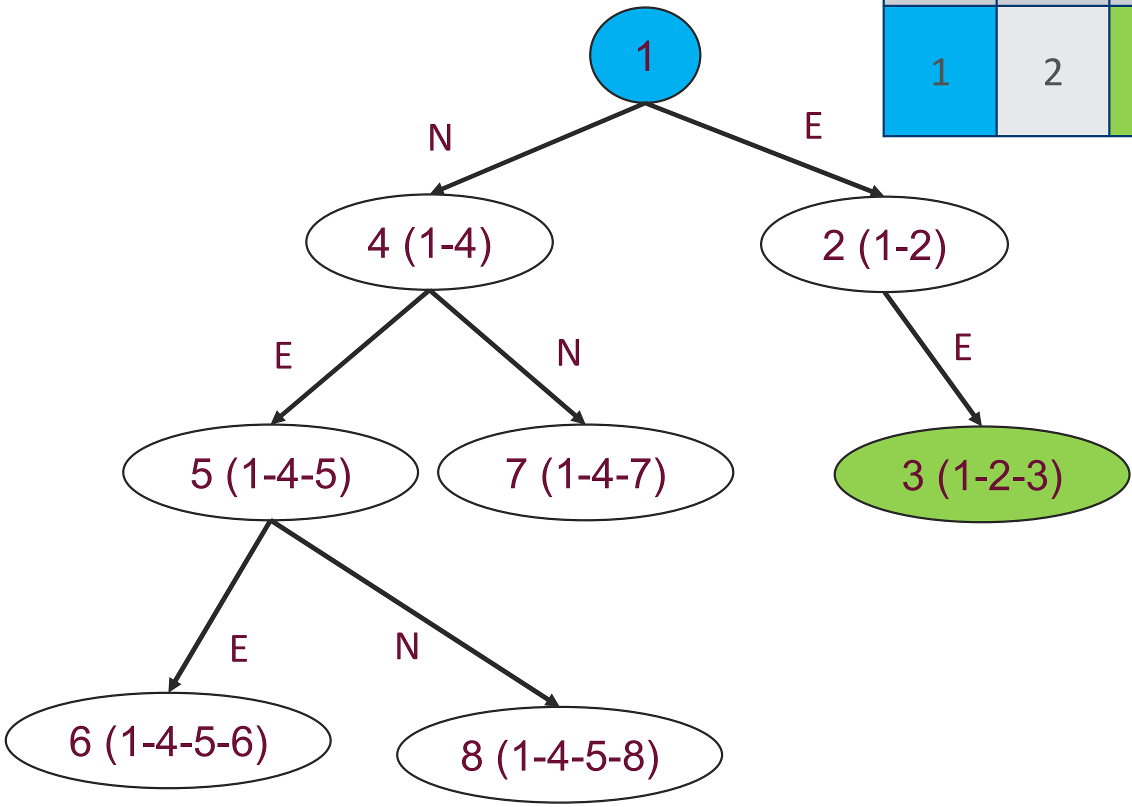


BREADTH FIRST SEARCH

Questions:

- Is Breadth first optimal?

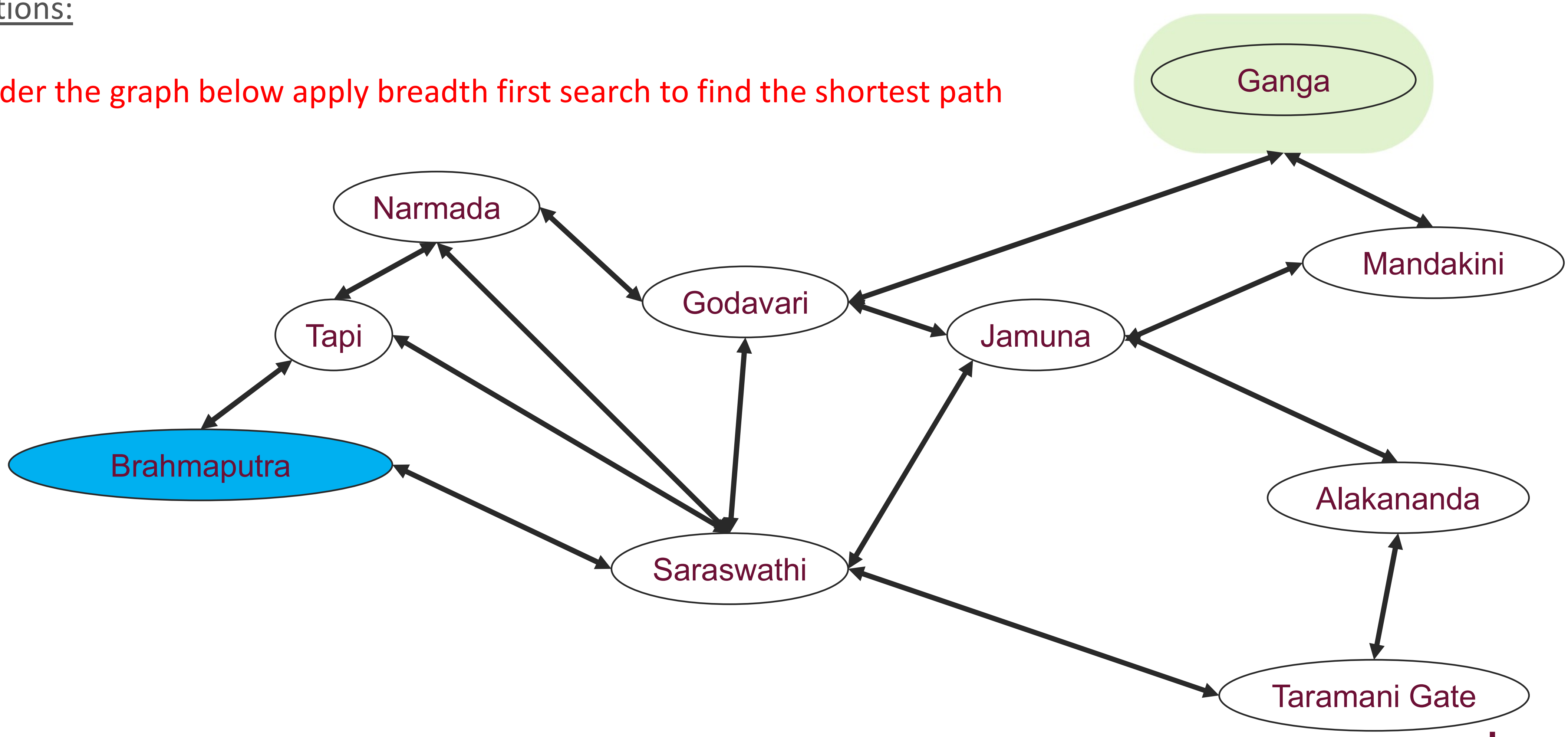
Yes, since we expand nodes at the same level first the path to goal will be optimal, under the assumption each step is unit cost



BREADTH FIRST SEARCH

Questions:

Consider the graph below apply breadth first search to find the shortest path



BREADTH FIRST SEARCH

Strategy: Expand the shallowest node first

Fringe (B0): pop B0

Fringe (T1,S1): pop T1

Fringe (S1, N2): pop S1

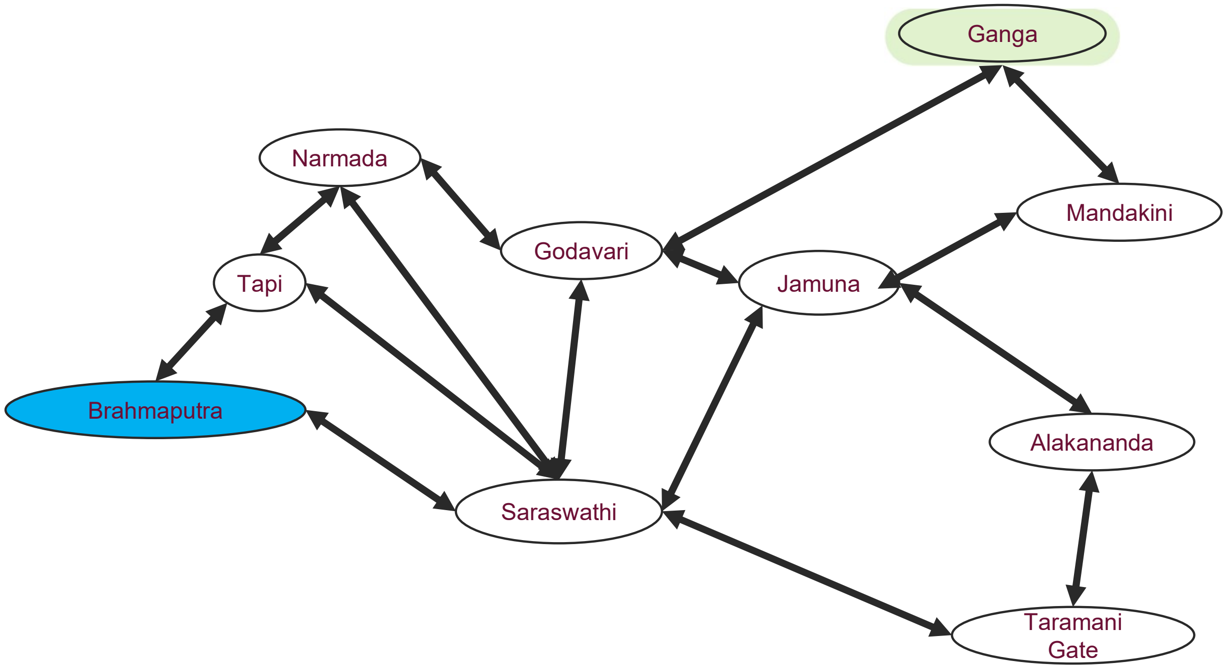
Fringe (N2, G2, J2, TG2): pop N2

Fringe (G2, J2, TG2): pop G2

Fringe (J2, TG2, Ganga3): pop J2

Fringe (TG2, Ganga3, M3, A3): pop TG2

Fringe (Ganga3, M3, A3): pop Ganga3 → goal: solution Brahmaputra, Saraswathi, Godavari, Ganga



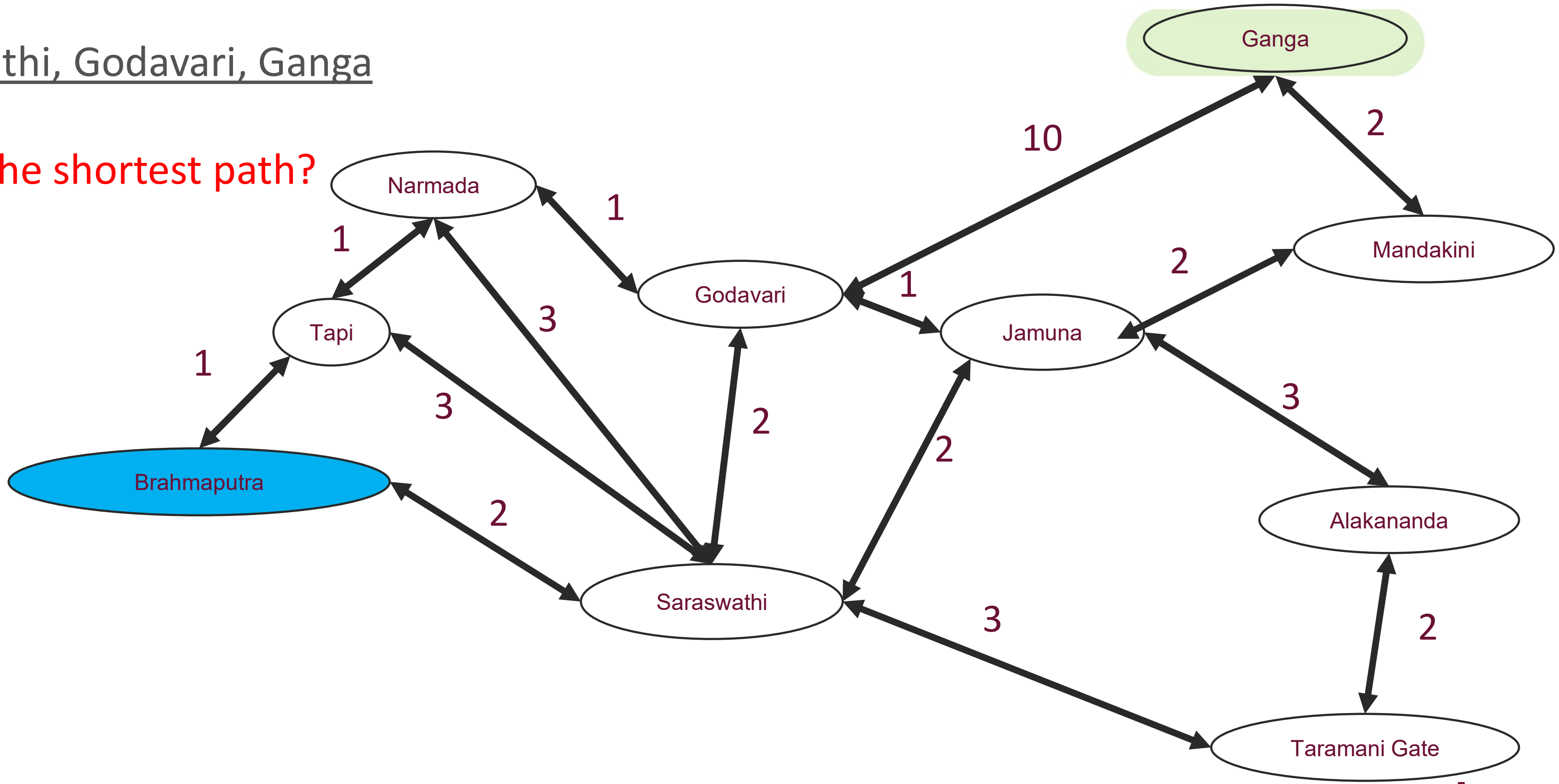
BREADTH FIRST SEARCH

Questions:

Consider the graph below apply breadth first search to find the shortest path

Solution Brahmaputra, Saraswathi, Godavari, Ganga

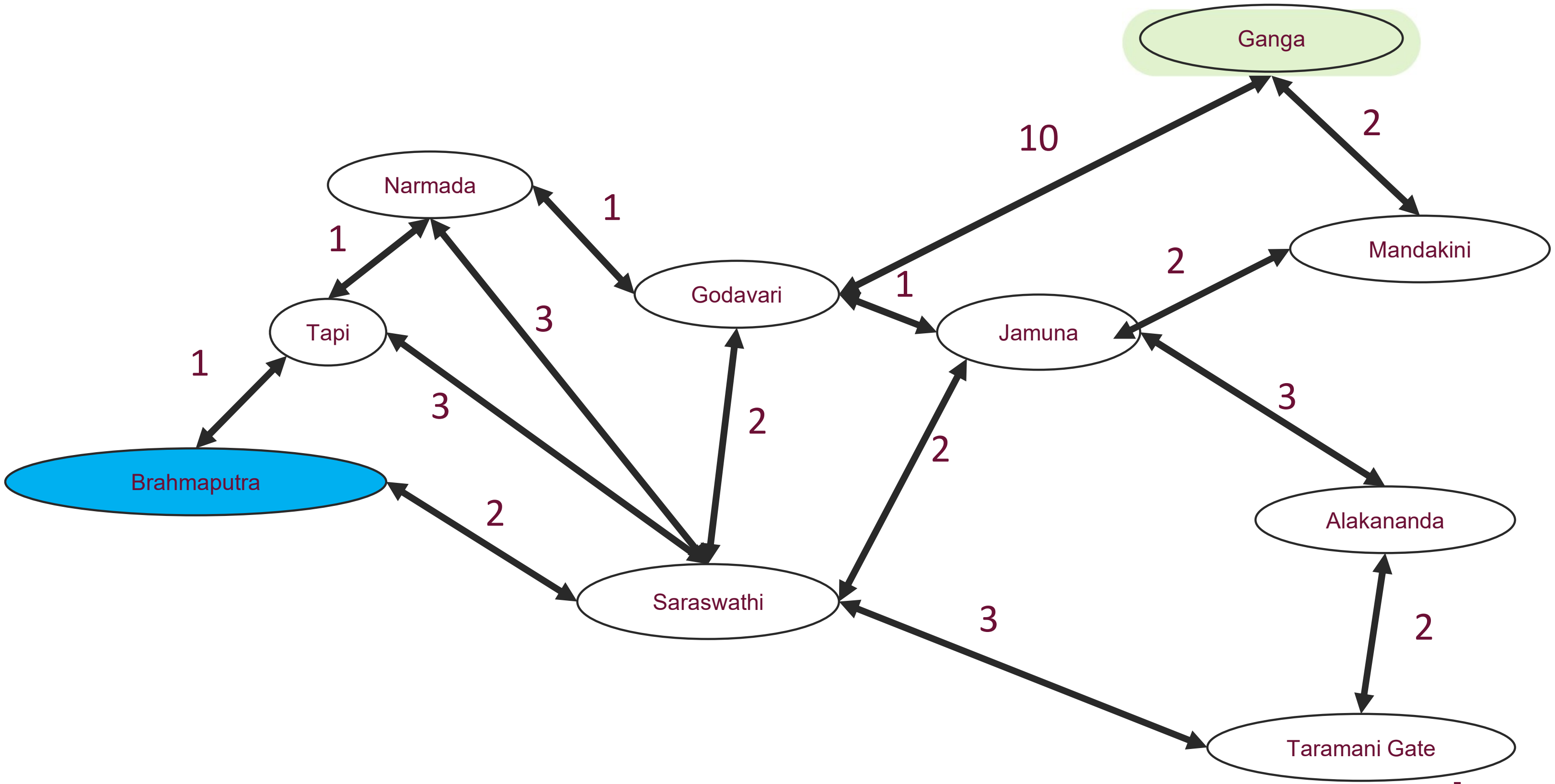
With arc costs added, is it still the shortest path?



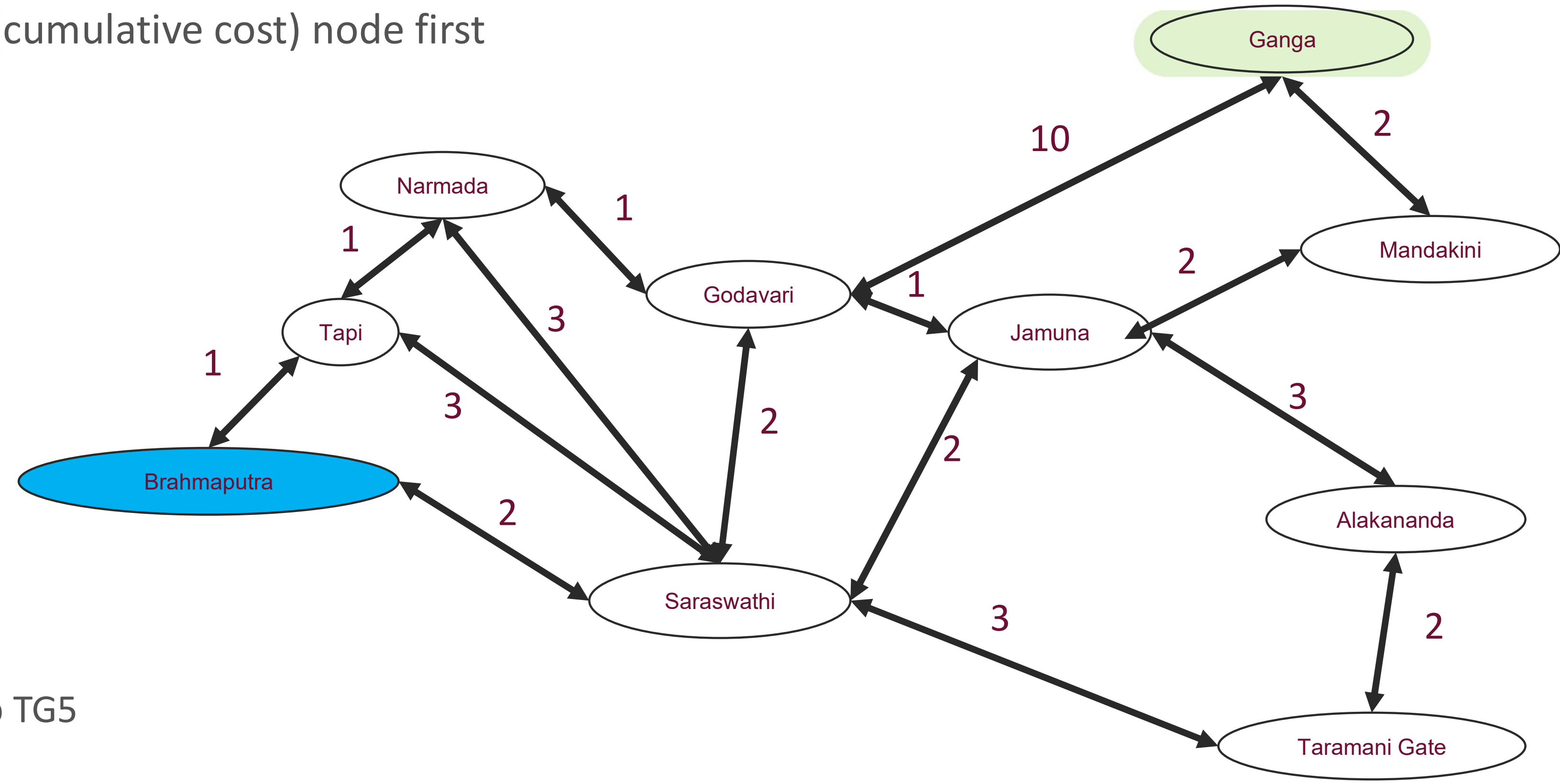
UNIFORM COST SEARCH

Limitation of Breadth first search: Cannot accommodate non uniform edge/action cost

Solution: Prioritize nodes to expand based on cumulative cost → Uniform Cost search



UNIFORM COST SEARCH



Strategy: Expand the cheapest (cumulative cost) node first

Fringe (B0): pop B0

Fringe (T1,S2): pop T1

Fringe (S2, N2): pop S2

Fringe (N2, G4, J4, TG5): pop N2

Fringe (G3, J4, TG5): pop G3

Fringe (J4, TG5, Ganga13): pop J4

Fringe (TG5, M6, A7,Ganga13): pop TG5

Fringe (M6, A7,Ganga13): pop M6

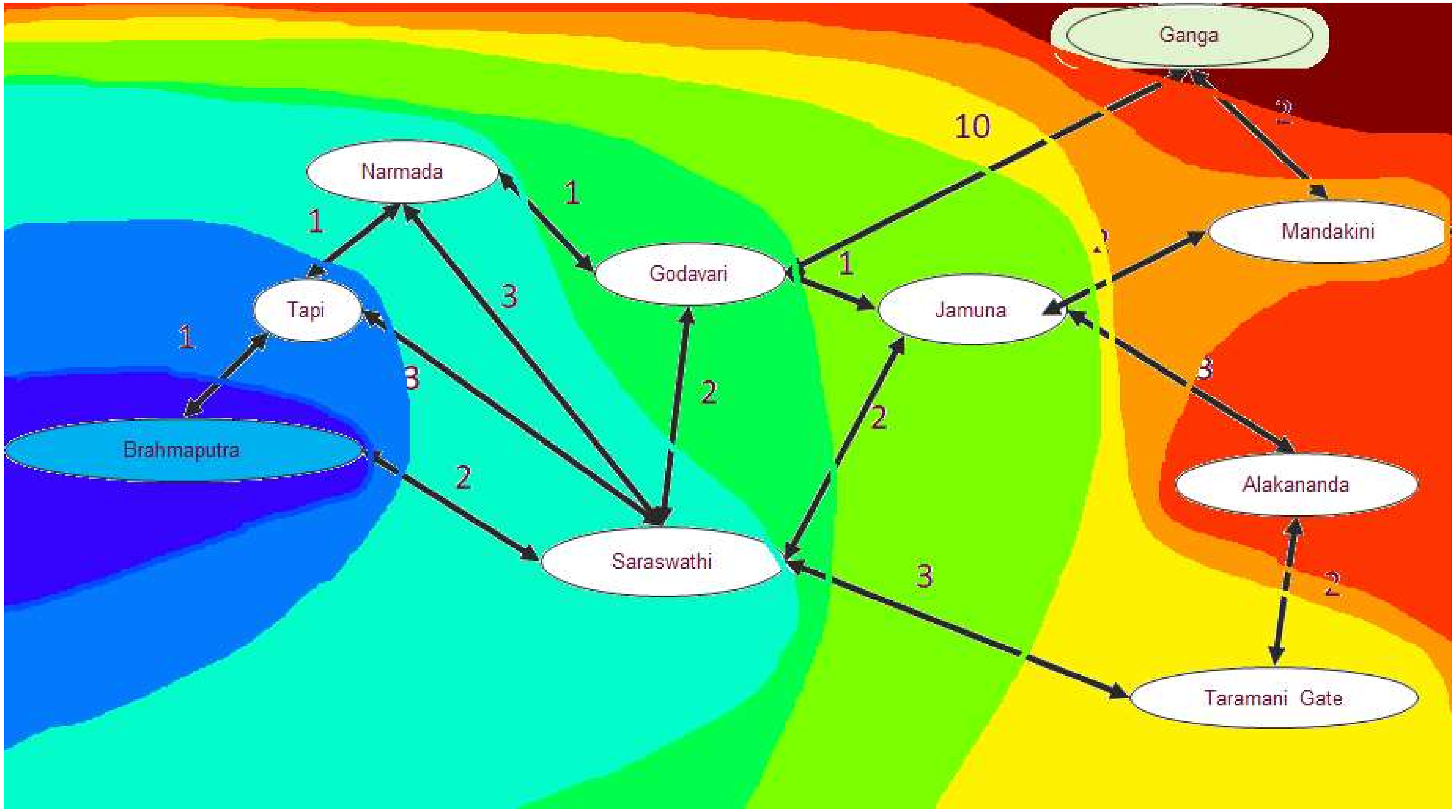
Fringe (A7,Ganga8): pop A7

Fringe(Ganga8):pop Ganga8 → goal: solution Brahmaputra, Saraswathi, Jamuna, Mandakini, Ganga

UNIFORM COST SEARCH

Solution: Brahmaputra, Saraswathi, Jamuna, Mandakini, Ganga

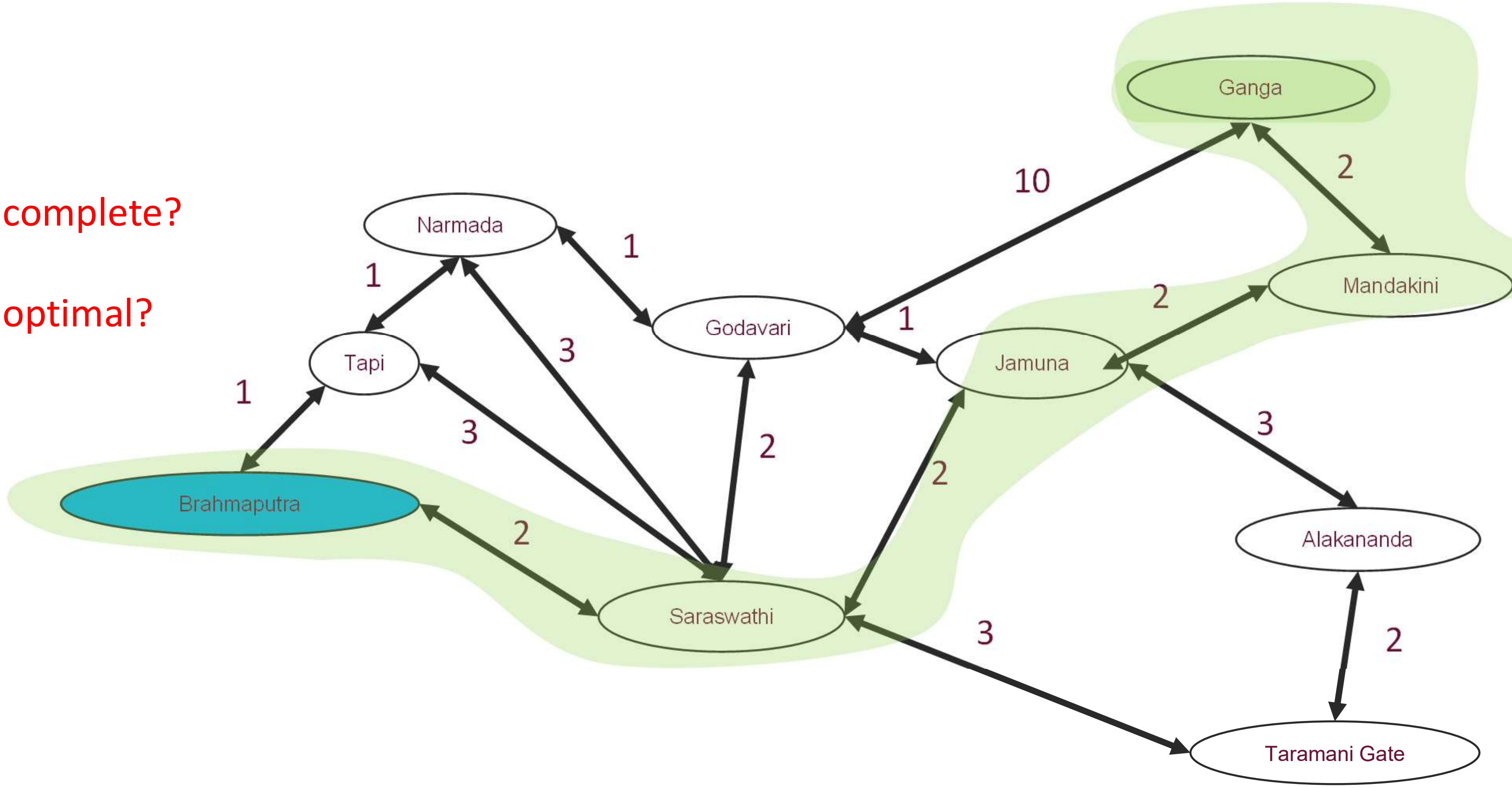
Cost Contours:



UNIFORM COST SEARCH

Questions:

- Is Uniform Cost Search complete?
- Is Uniform Cost Search optimal?

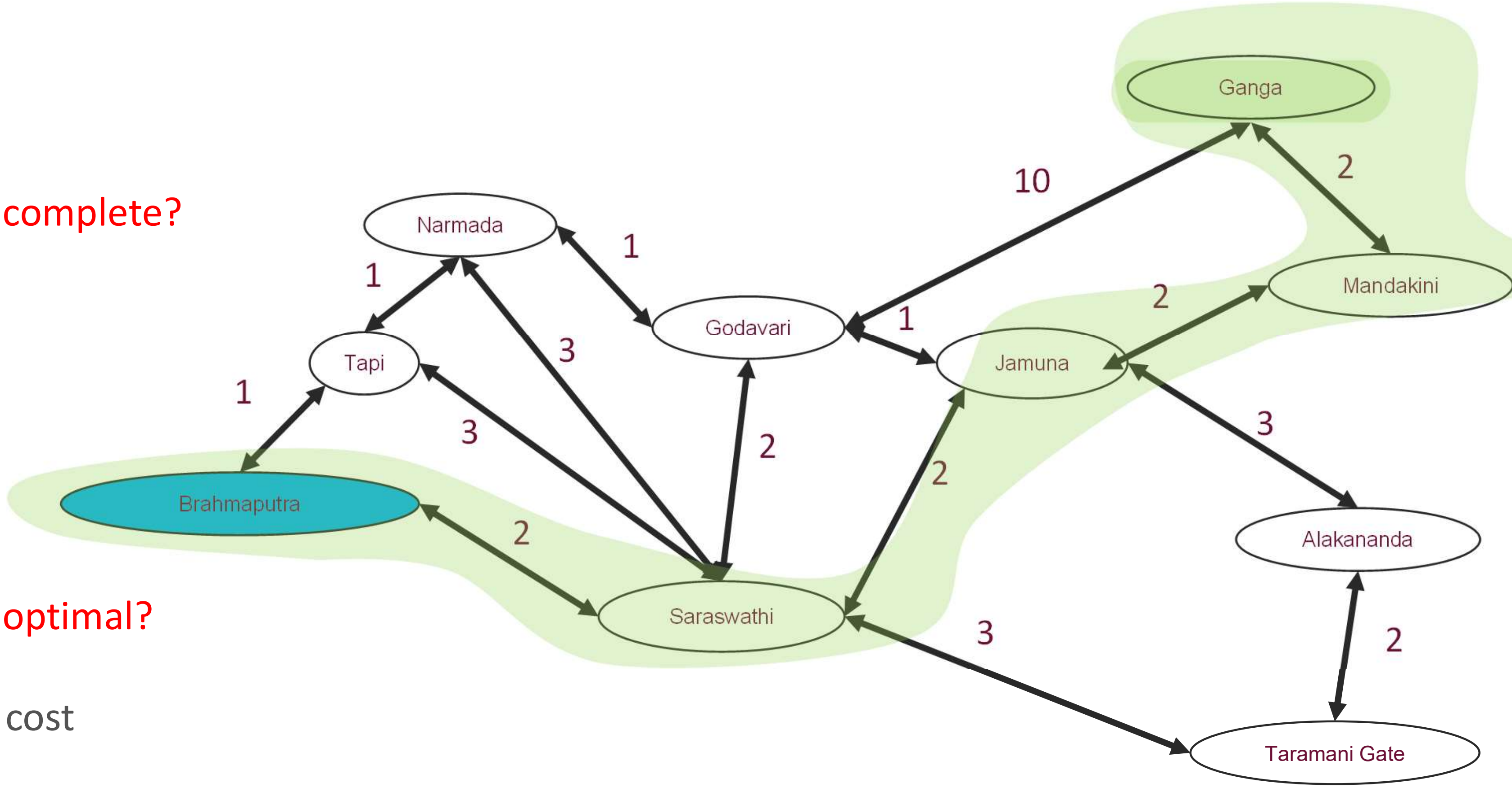


UNIFORM COST SEARCH

Questions:

- Is Uniform Cost Search complete?

Yes



- Is Uniform Cost Search optimal?

Yes, optimal in cumulative cost

Question: Is this the best, what could be the disadvantage of UCS, what can we do improve?