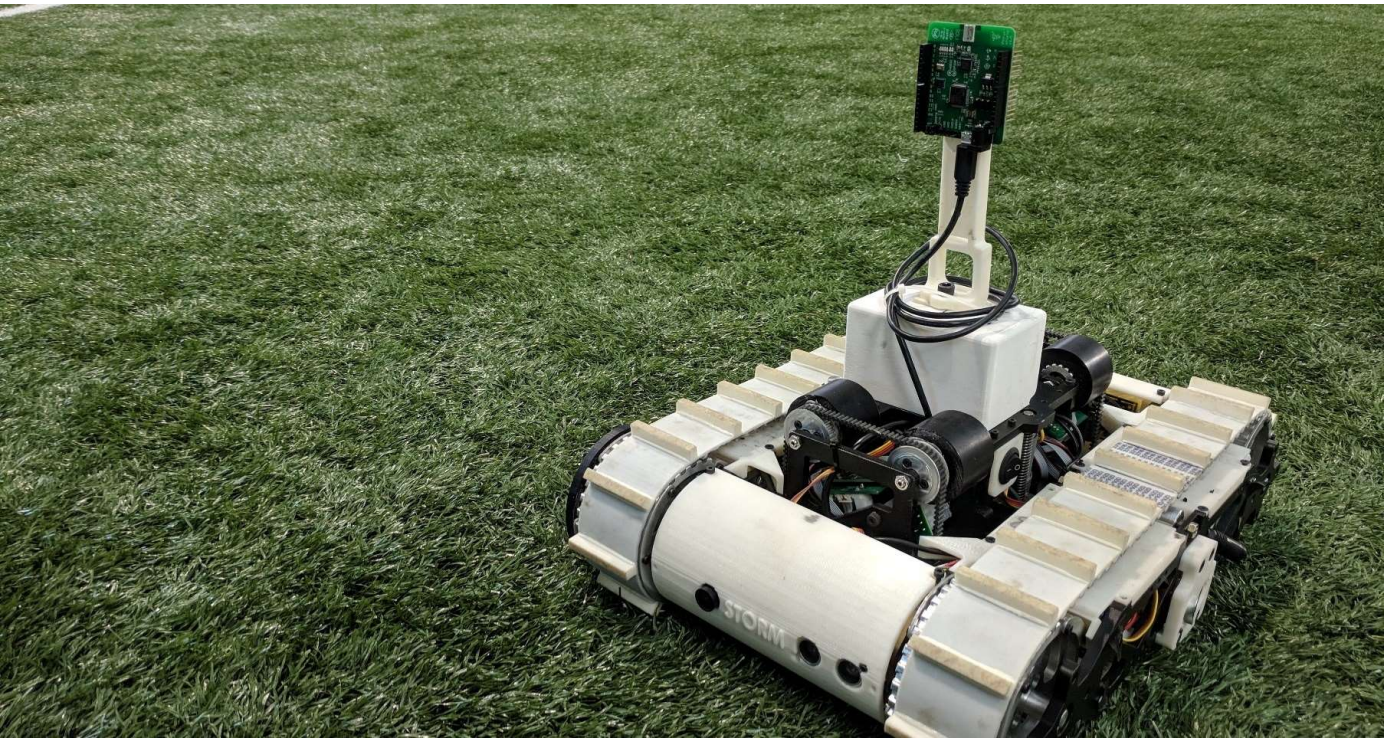


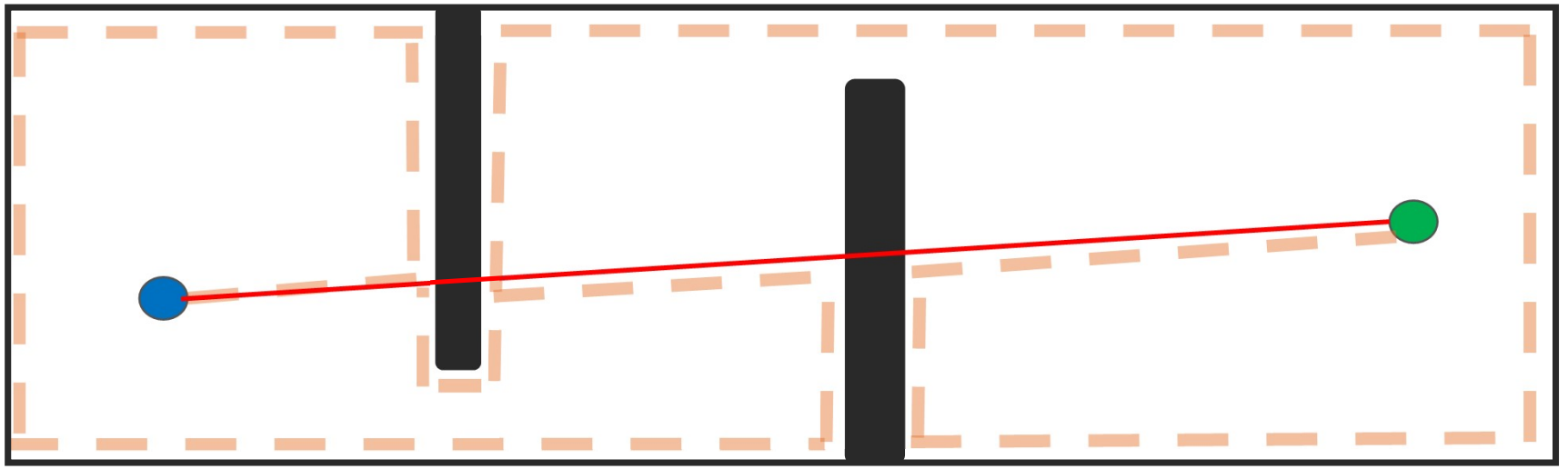
ED 5215
INTRODUCTION TO MOTION PLANNING

Instructor: Bijo Sebastian

MOTIVATION

Bug algorithms are easy to implement but, the solution by Bug-2 algorithm is not the shortest path

→ NOT rational

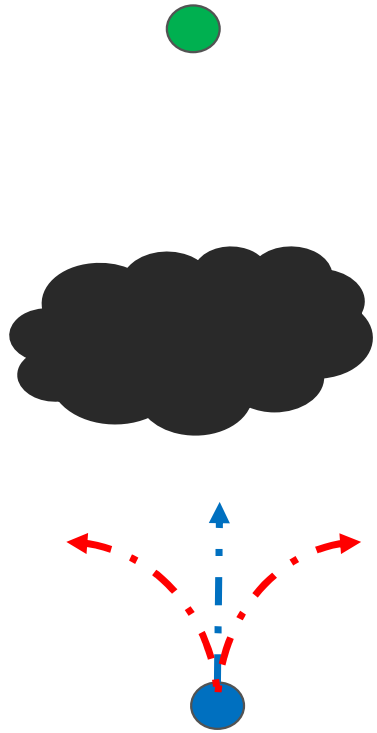


Question: What would be the shortest path and how do you find it, assuming a model of the world (map) is available ?

MOTIVATION

Reflex agents

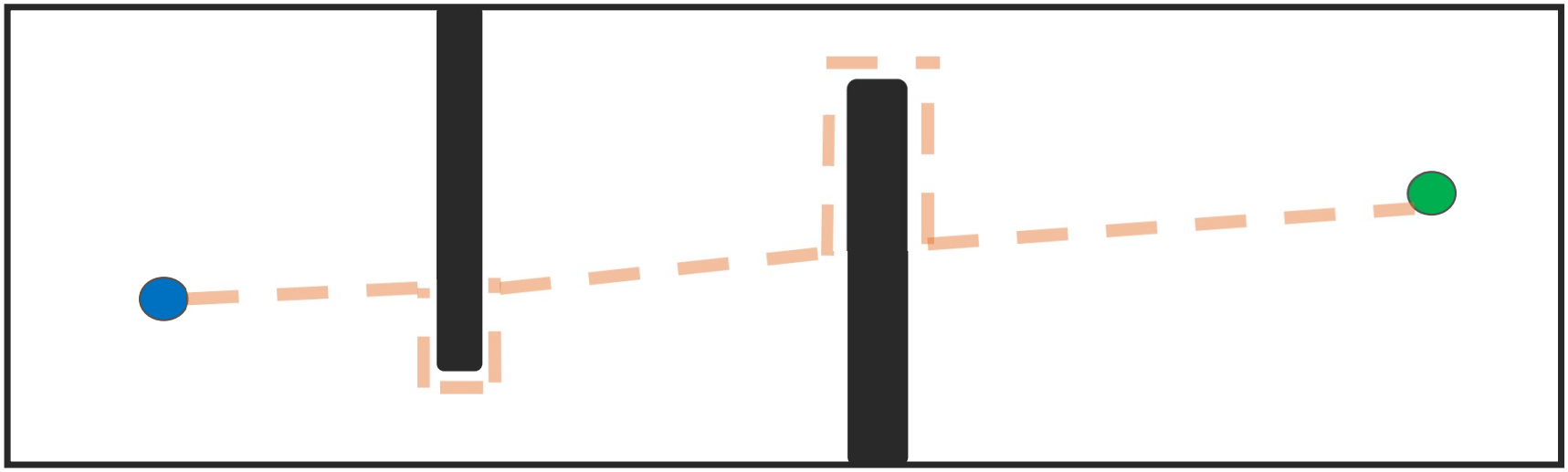
- Example Bug algorithms, robot with Go-to-goal and avoid-obstacle controllers
- Choose action based on current percept (and maybe memory)
- May have memory or a model of the world's current state
- Do not consider the future consequences of their actions
- Consider how the world IS (which could change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



MOTIVATION

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 be able to recognize when a goal is achieved
- 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

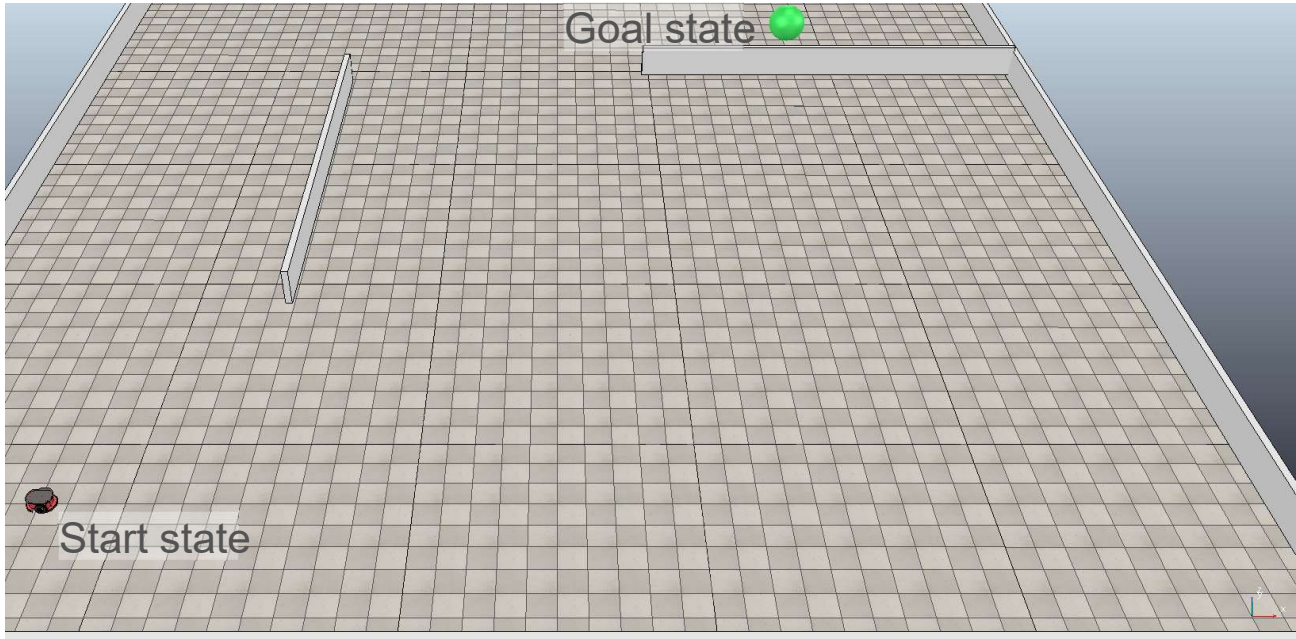
Uninformed search methods:

- 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

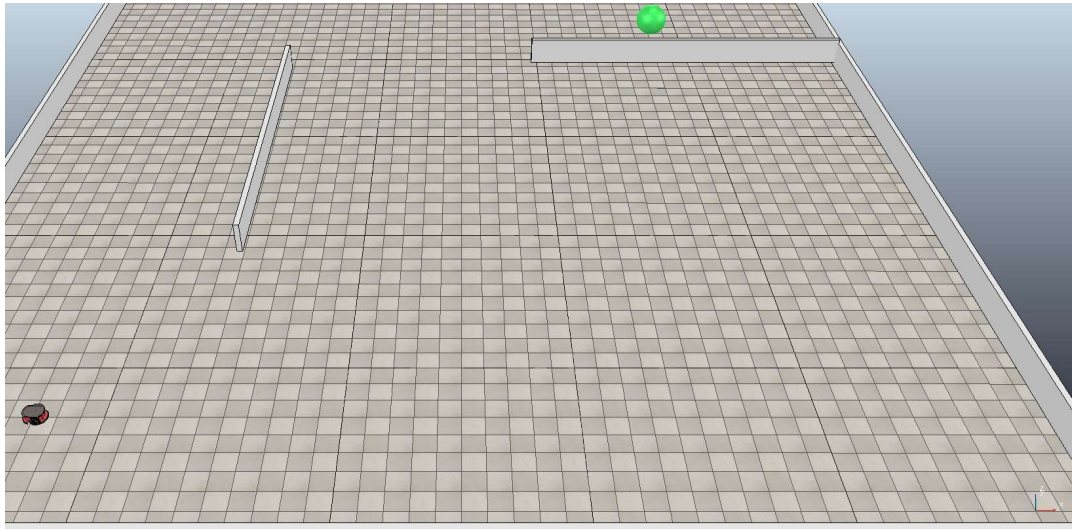
(x, y)

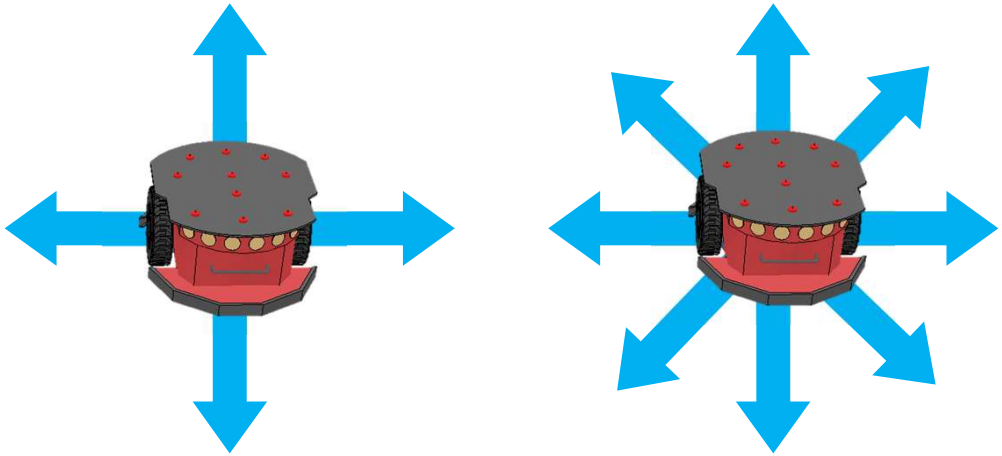
SEARCH PROBLEM

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



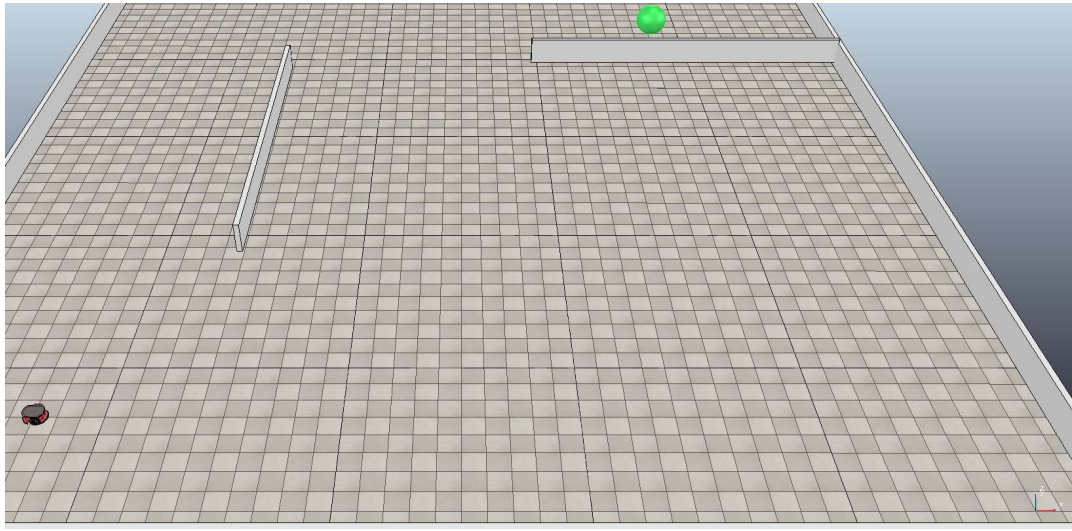


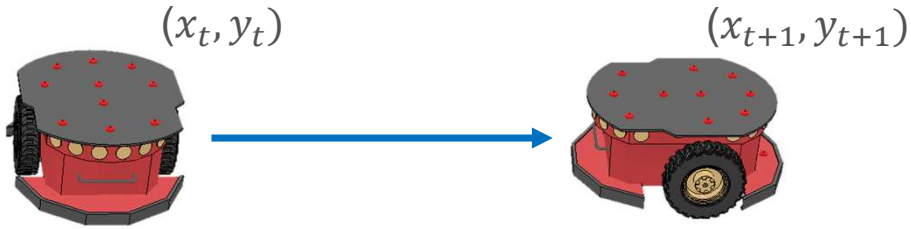
SEARCH PROBLEM

Elements of a search problem

3. State expansion function (successor function)

- 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)



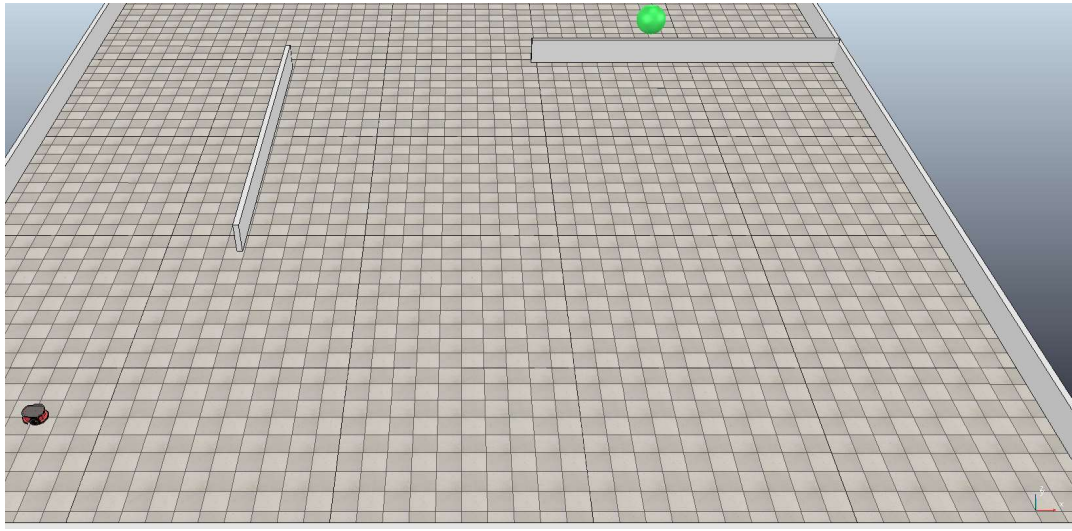


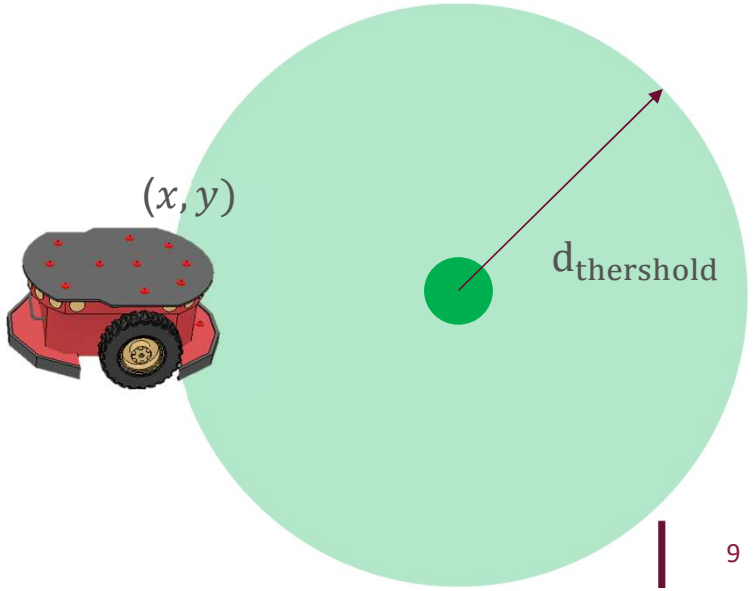
SEARCH PROBLEM

Elements of a search problem

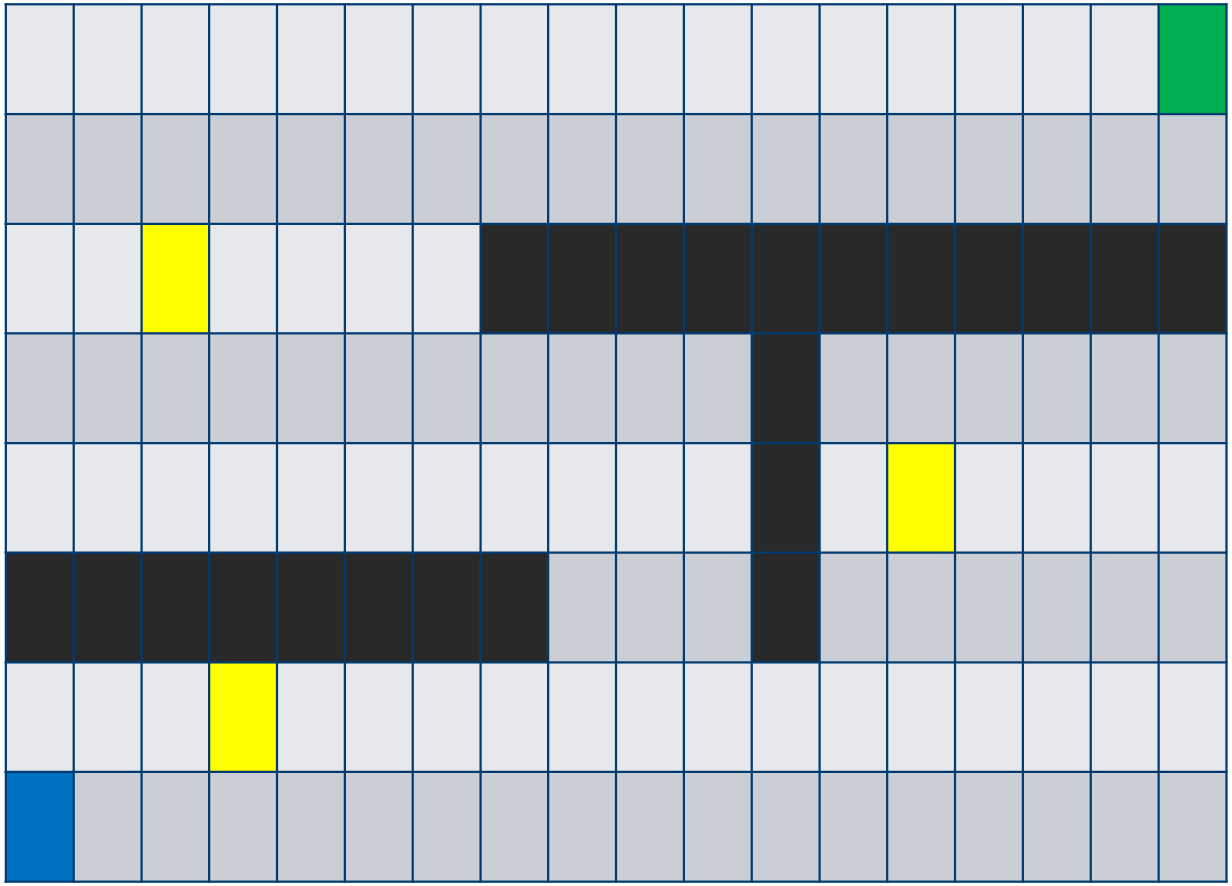
4. Goal test

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





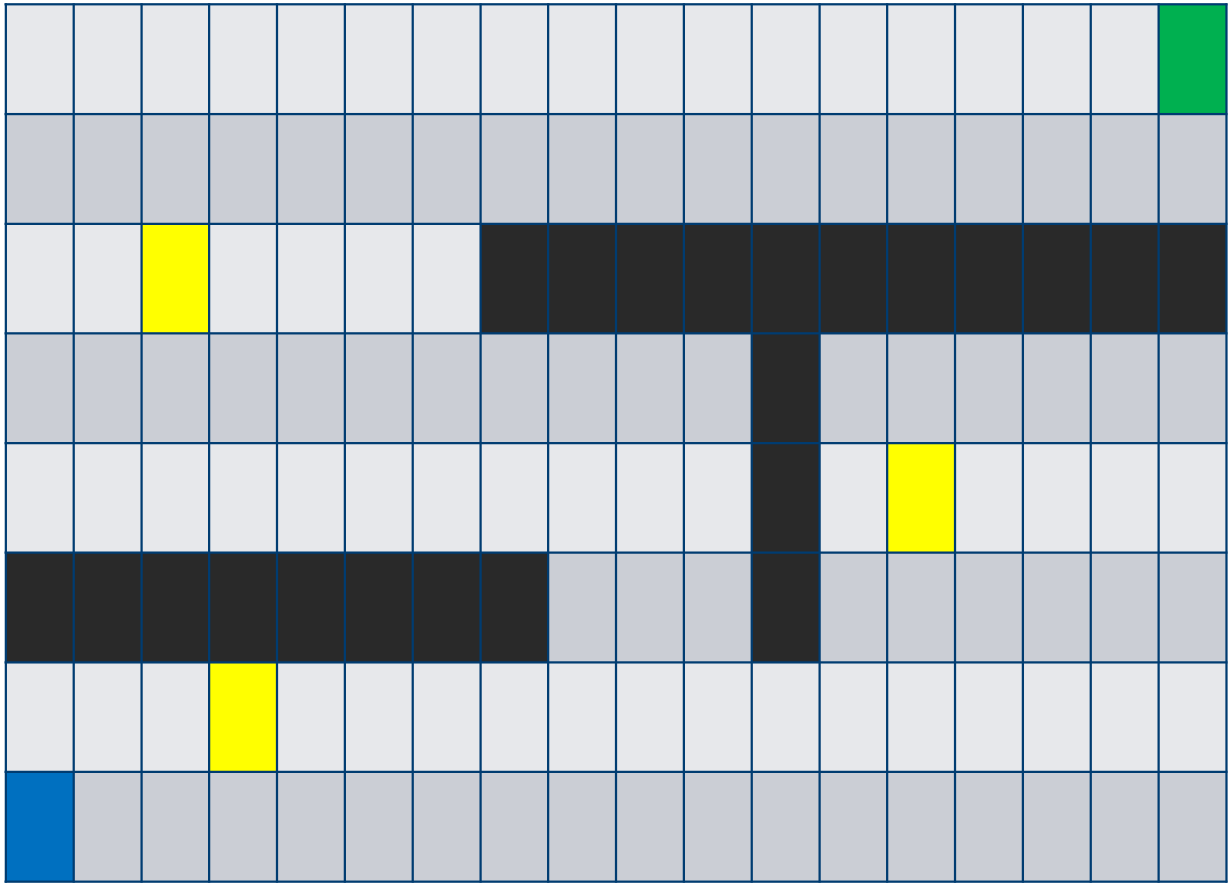
SEARCH PROBLEM



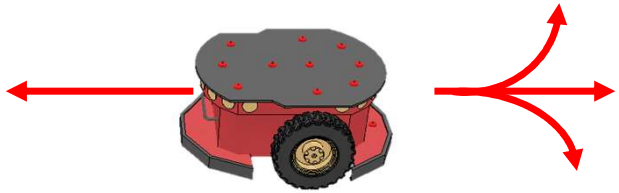
Question: The robot starts at the blue cell and want to get to the green cell, after visiting each of the yellow cell, along the shortest path. Assume the robot can move to four neighboring cells.

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

SEARCH PROBLEM



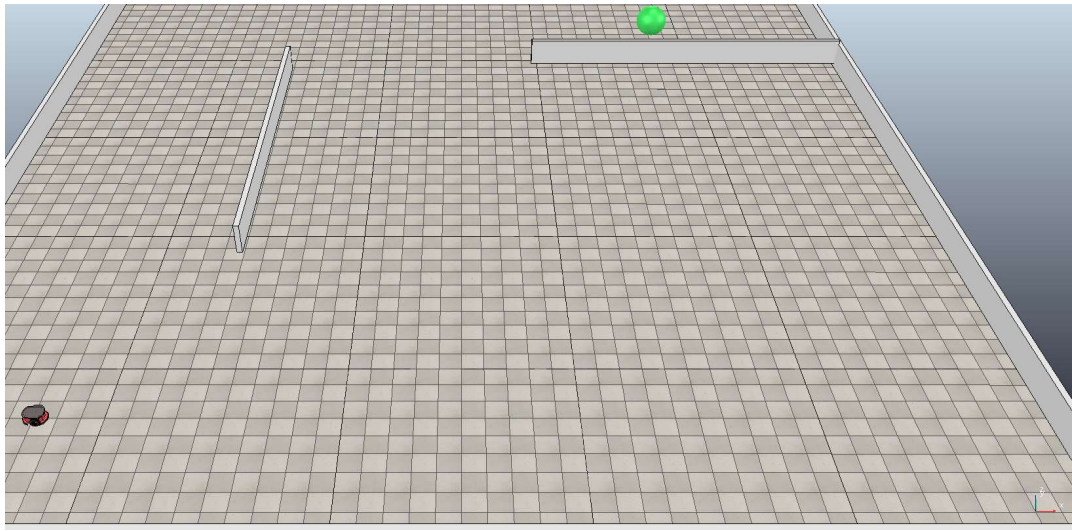
Question: In the previous problem, assume the robot motion is constrained as follows:



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

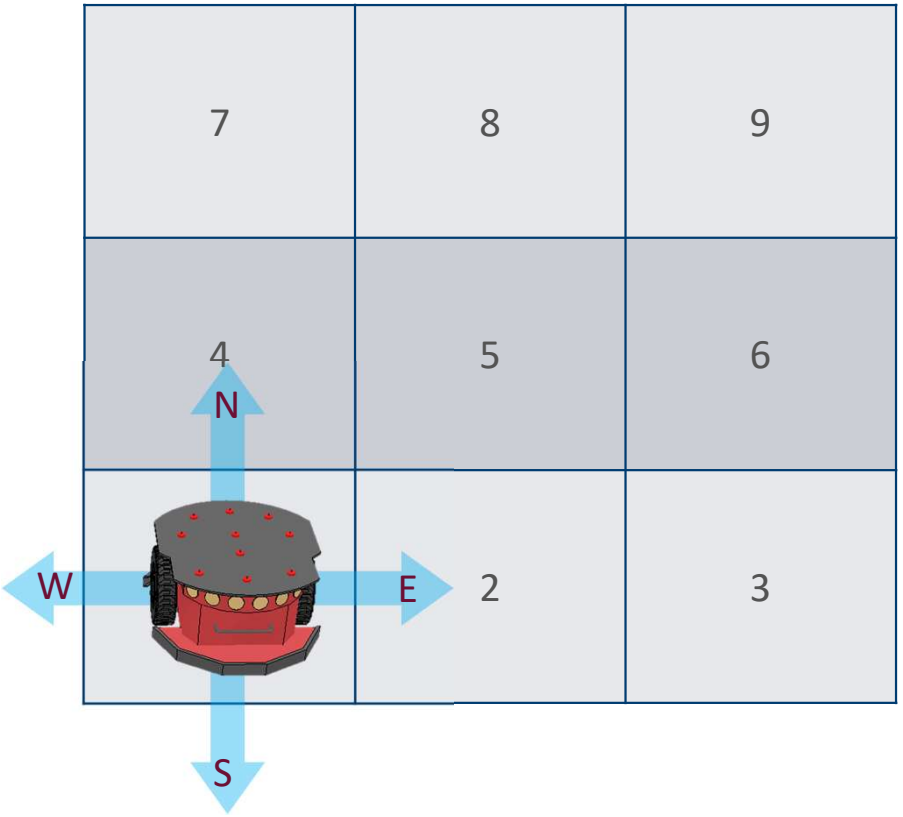
Library of search based or sampling-based planning algorithms (e.g., OMPL, Robotics Toolbox)

Solution

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

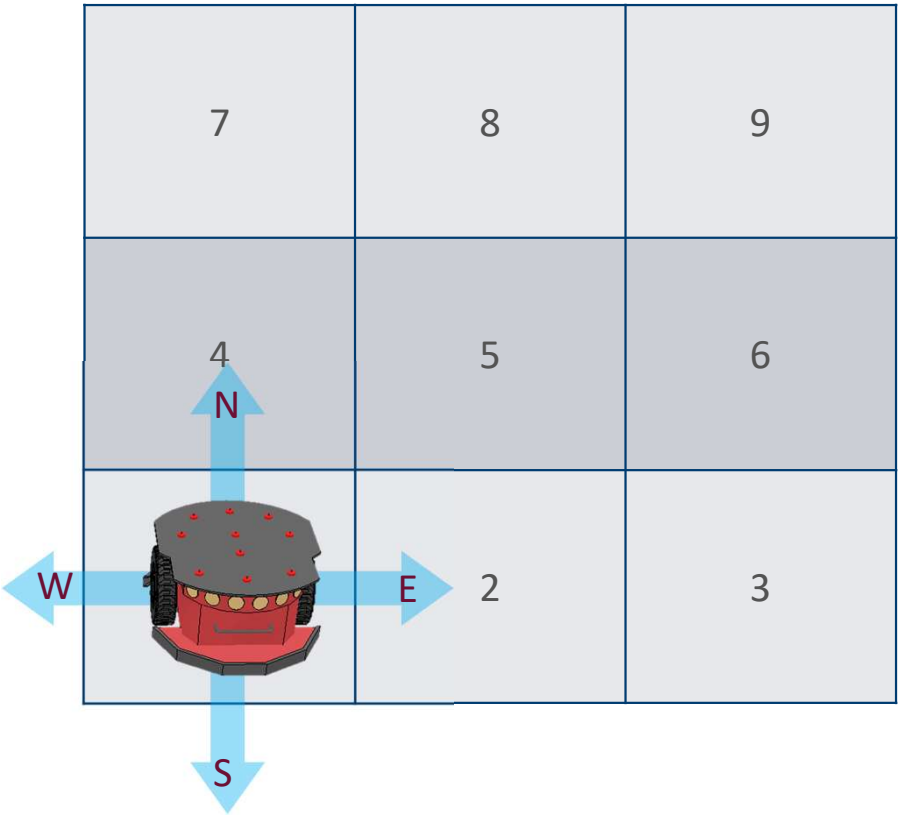
GRAPHS

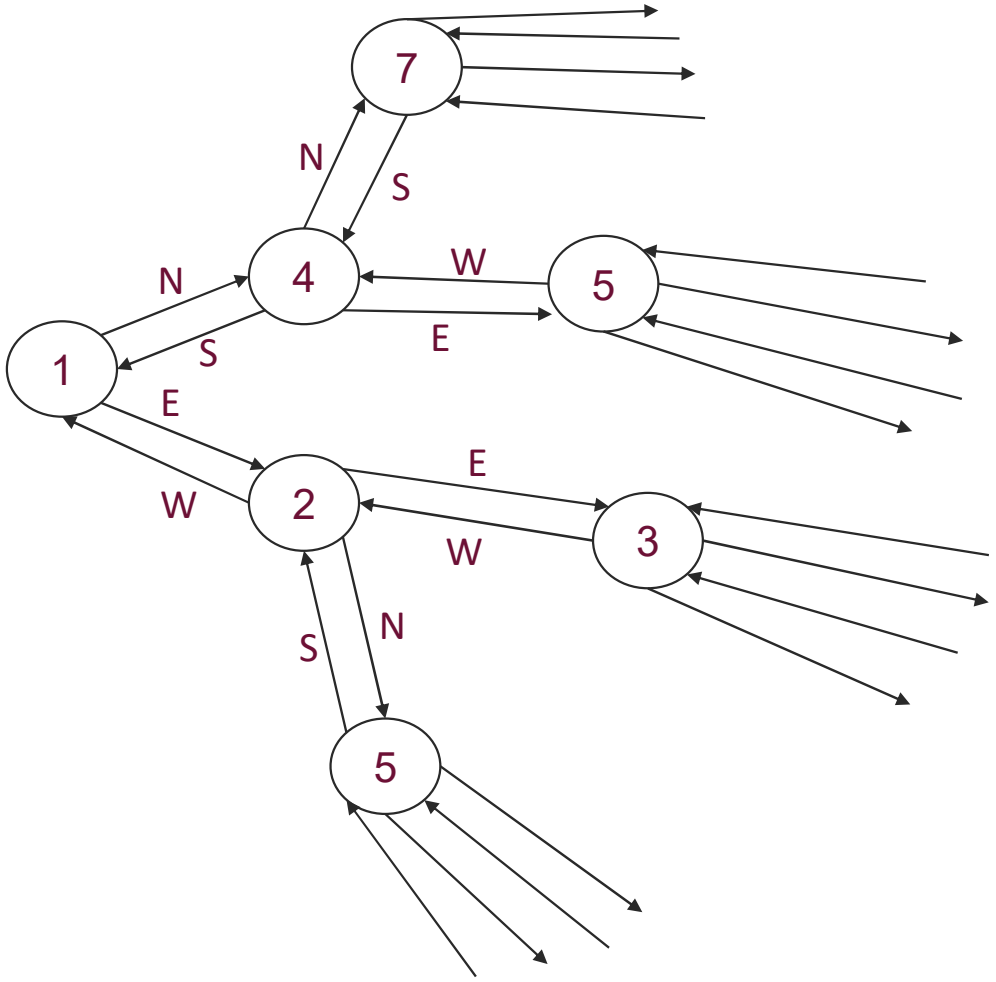
State space graph



GRAPHS

State space graph

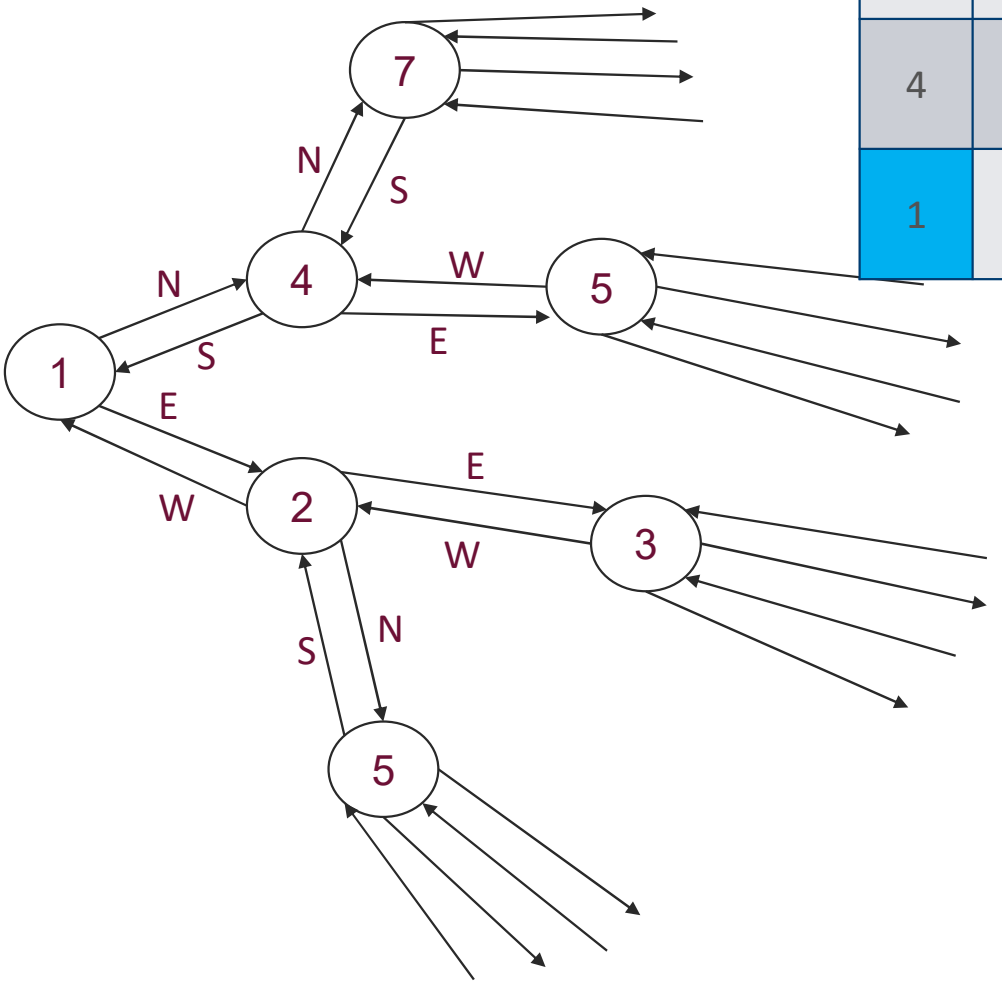




GRAPHS

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)



7	8	9
4	5	6
1	2	3

GRAPHS V/S TREES

7	8	9
4	5	6
1	2	3

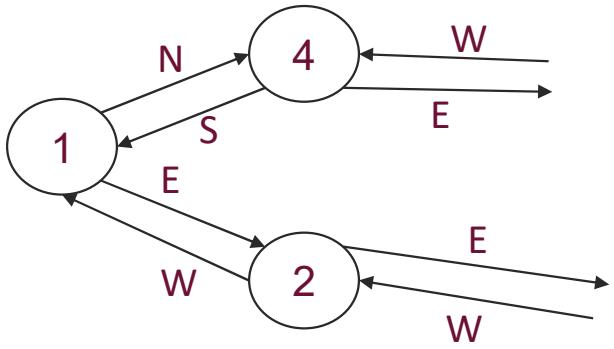
State space graph:

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

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

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



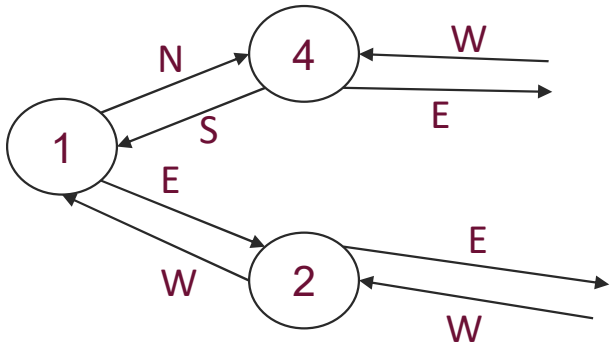
GRAPHS V/S TREES

7	8	9
4	5	6
1	2	3

State space graph:

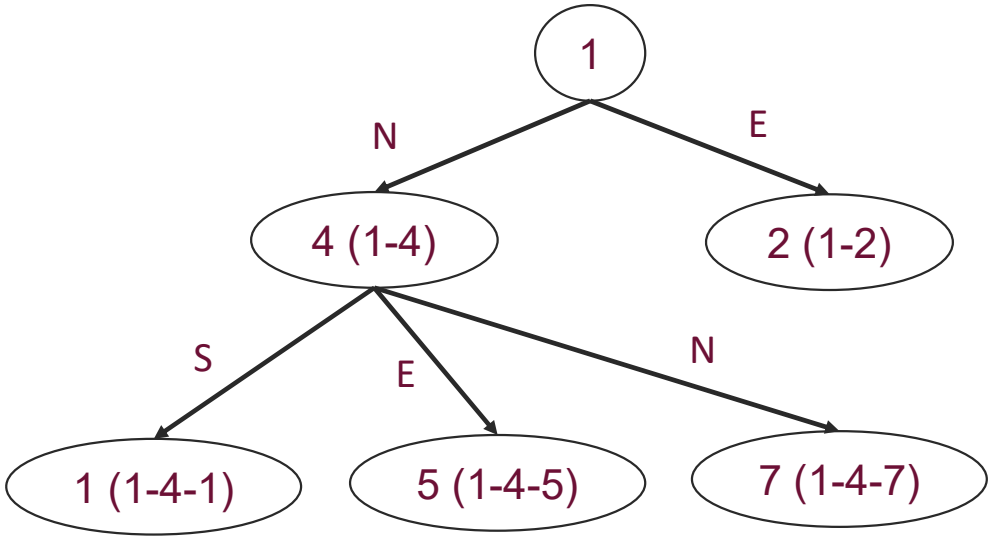
In a graph with doubly connected nodes, there are many ways to get to each node from start node

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



Search tree:

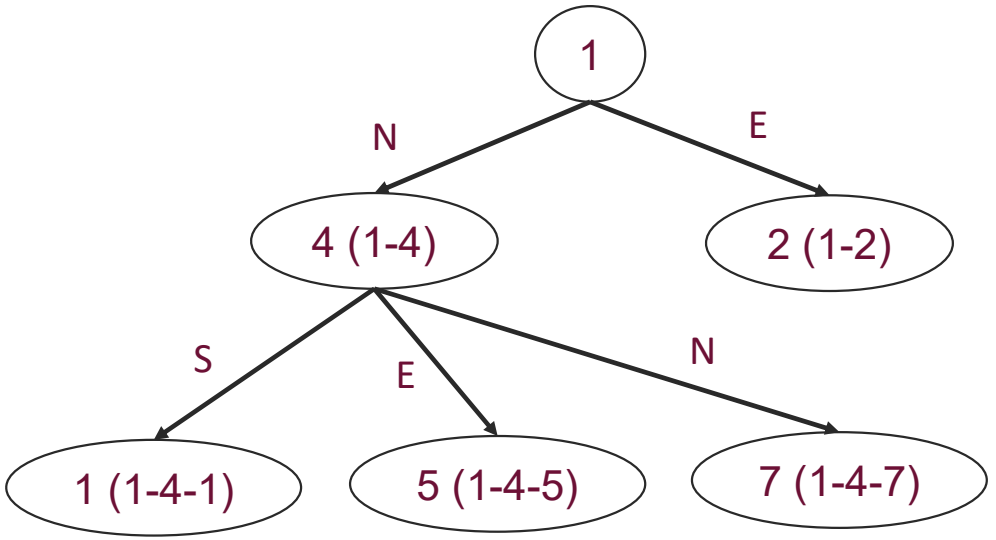
In a search tree, every node shows a state but also a plan on how to get to that state from the start node



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?

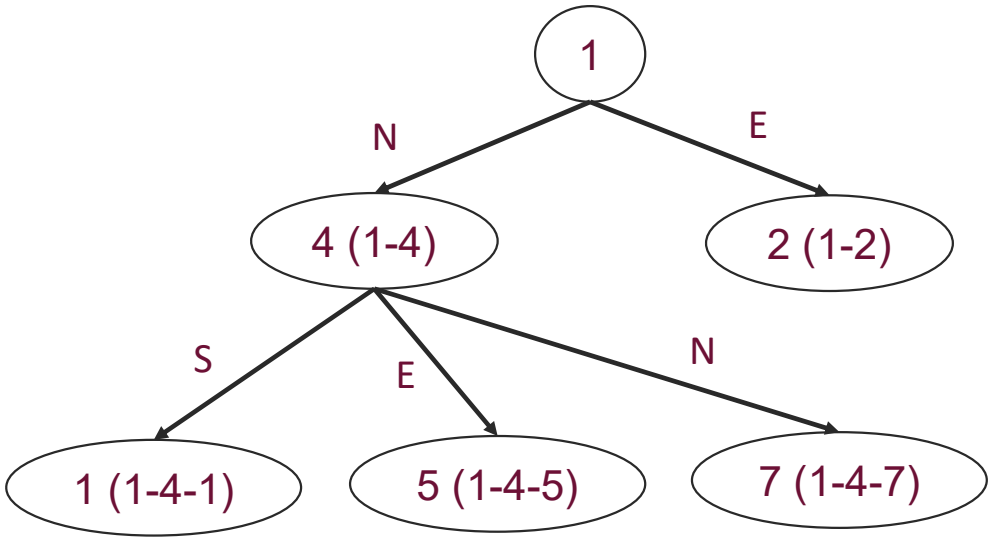


SEARCH TREE

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?

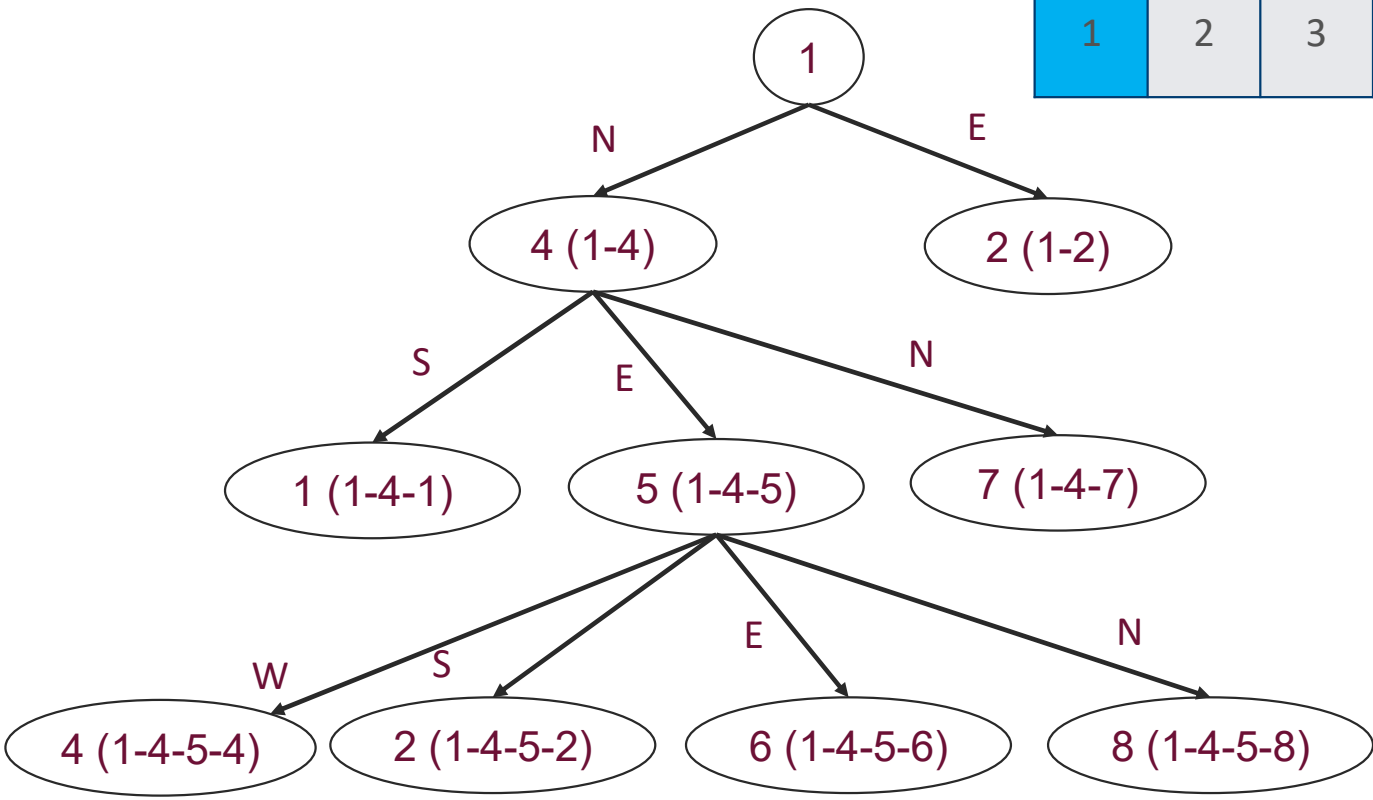
No, An efficient search algorithm must be able to come up with a plan to get to goal state by expanding minimum number of nodes/states



TREE SEARCH ALGORITHM

7	8	9
4	5	6
1	2	3

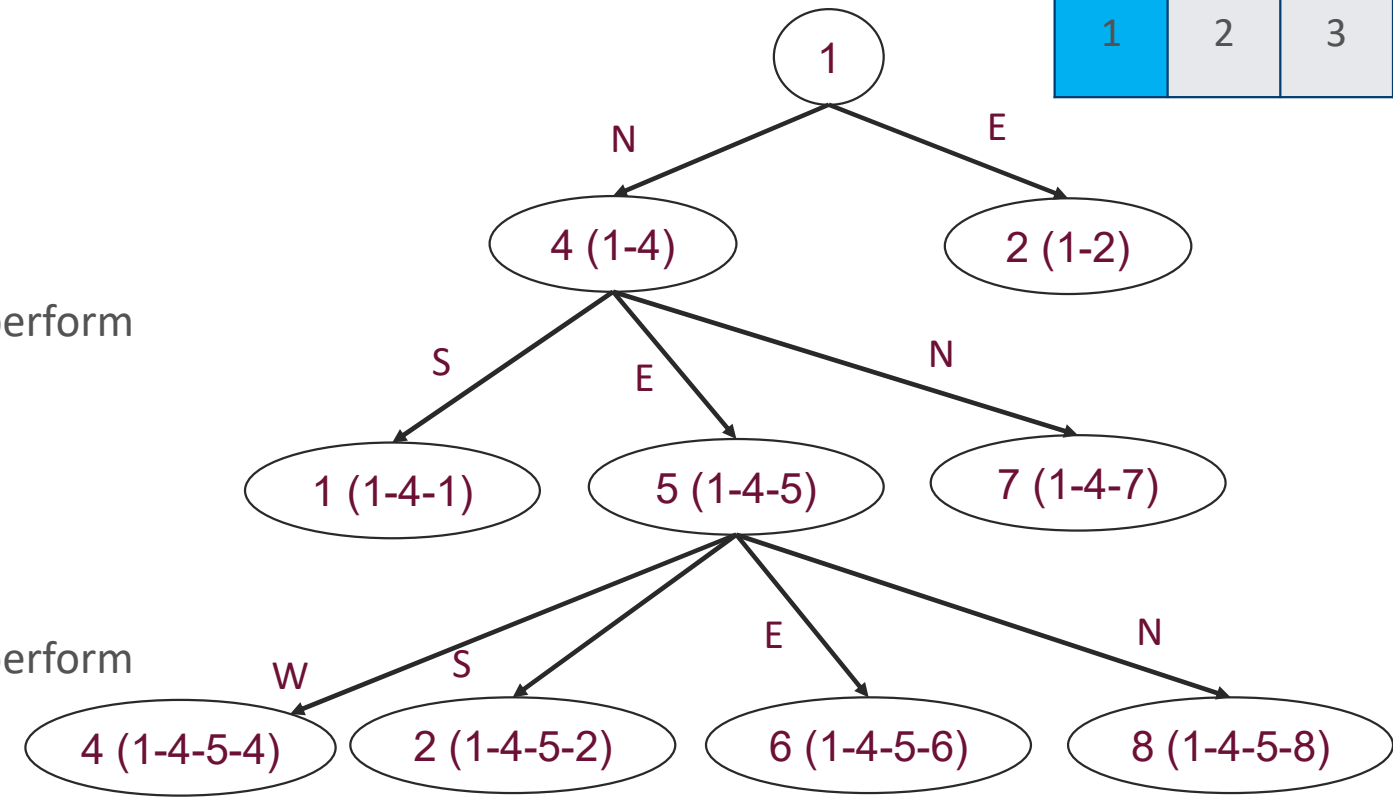
Question: Starting from 1 find a way to get to 8?



TREE SEARCH ALGORITHM

- Begin with start state in Fringe: { 1 }
- Remove a state(1) from Fringe and perform all possible actions to get to new states
- Add new states to Fringe: { 4, 2}
- Remove a state(4) from Fringe and perform all possible actions to get to new states
- Add new states to Fringe: {2,5,7,1}
- Remove a state from Fringe(5) and perform all possible actions to get to new states
- Add new states to Fringe: {2,7,1,1,2,6,8}

7	8	9
4	5	6
1	2	3

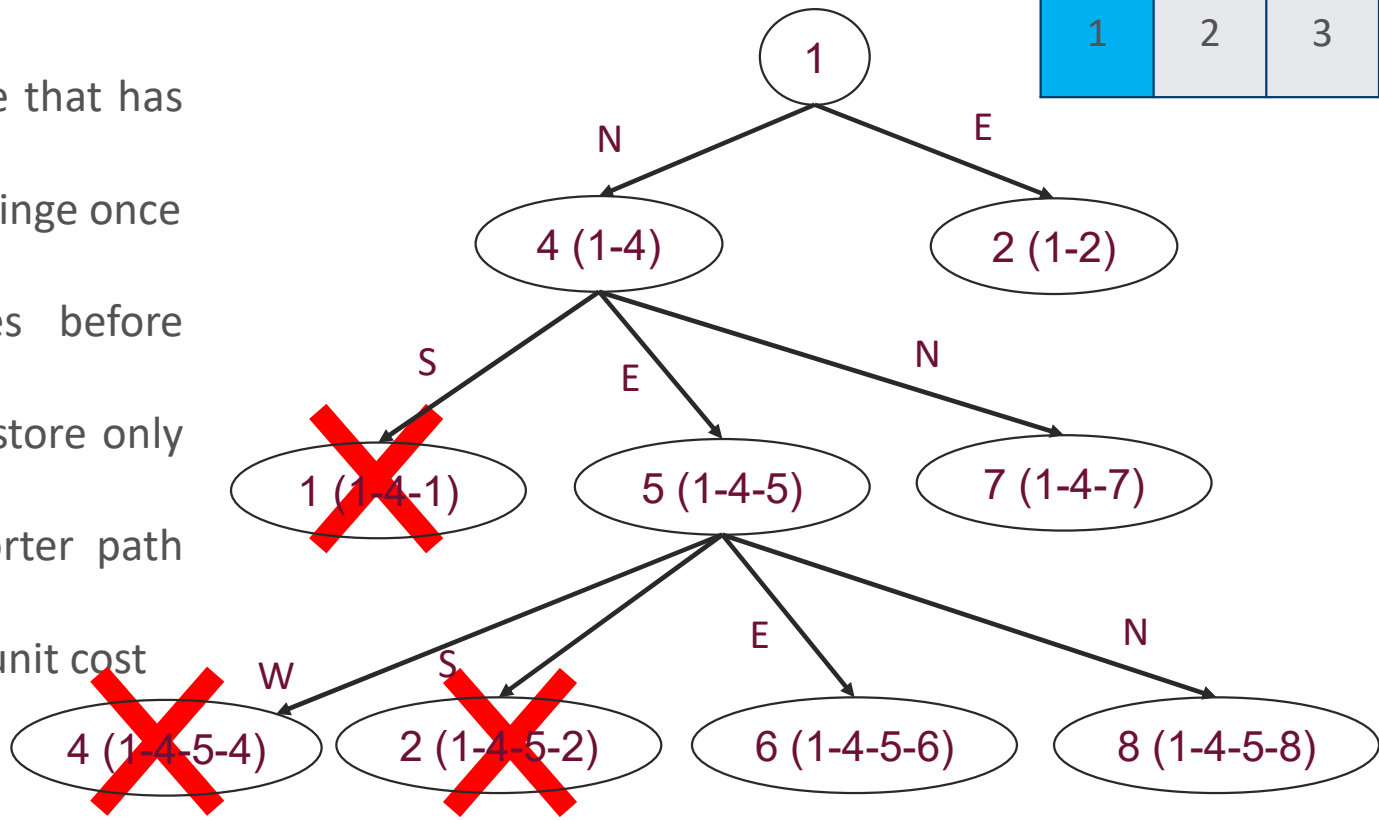


TREE SEARCH ALGORITHM

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 exists, 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-FORMAL

Pseudo code:

function **TREE-SEARCH**(problem, strategy) returns a solution, or failure
    create empty fringe and closed set
    add the initial state of problem to fringe
    loop do
        if there are no candidates for expansion in the fringe then return failure
        pop a node for expansion from fringe according to strategy
        if the popped node is goal state, then return the corresponding solution
        else
        add the popped node to closed set
        expand the popped node by performing all possible actions
        for each resulting child node:
            add the child node to the fringe
            except:
                if child node exists in closed set
                if duplicate child node exists in fringe:
                    store the one that can be reached via shorter path from start node

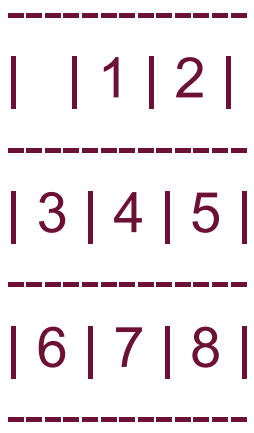
Based on the strategy used in choosing node/state from fringe, we have different planning algorithms like depth first , breadth first, uniform cost, Dijkstra

EIGHT PUZZLE

Eight puzzle link:
https://www.helpfulgames.com/subjects/brain-training/sliding-puzzle.html

Question: Can we formulate Eight puzzle as search problem, that allows you to solve it in minimum number of moves?

What should be the states of the search problem?
What are the actions?
How do you formulate a state expansion function?
What is the goal check?

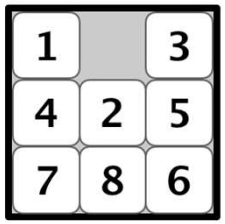


Solved puzzle, note the blank position before 1

EIGHT PUZZLE

Can we formulate Eight puzzle as search problem, that allows you to solve it in minimum number of moves?
Yes, see demonstration

What should be the states of the search problem?
String representing board:



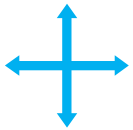
board

string : 1, _, 3, 4, 2, 5, 7, 8, 6

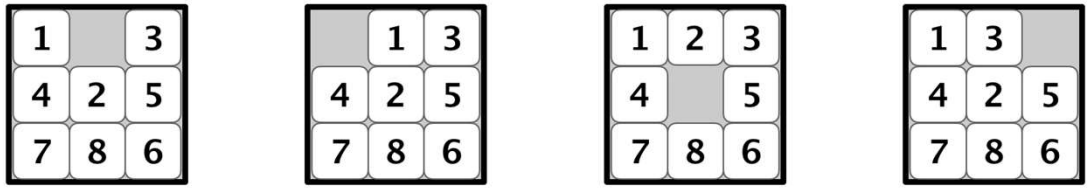
What is the goal check?
String comparison with string : _,1, 2, 3, 4, 5, 6, 7, 8

EIGHT PUZZLE

What are the actions?
Move the empty tile in four directions, within limits of the board



Depending on the location of the blank square, a board can have 2, 3, or 4 neighbors.



How do you formulate a state expansion function?

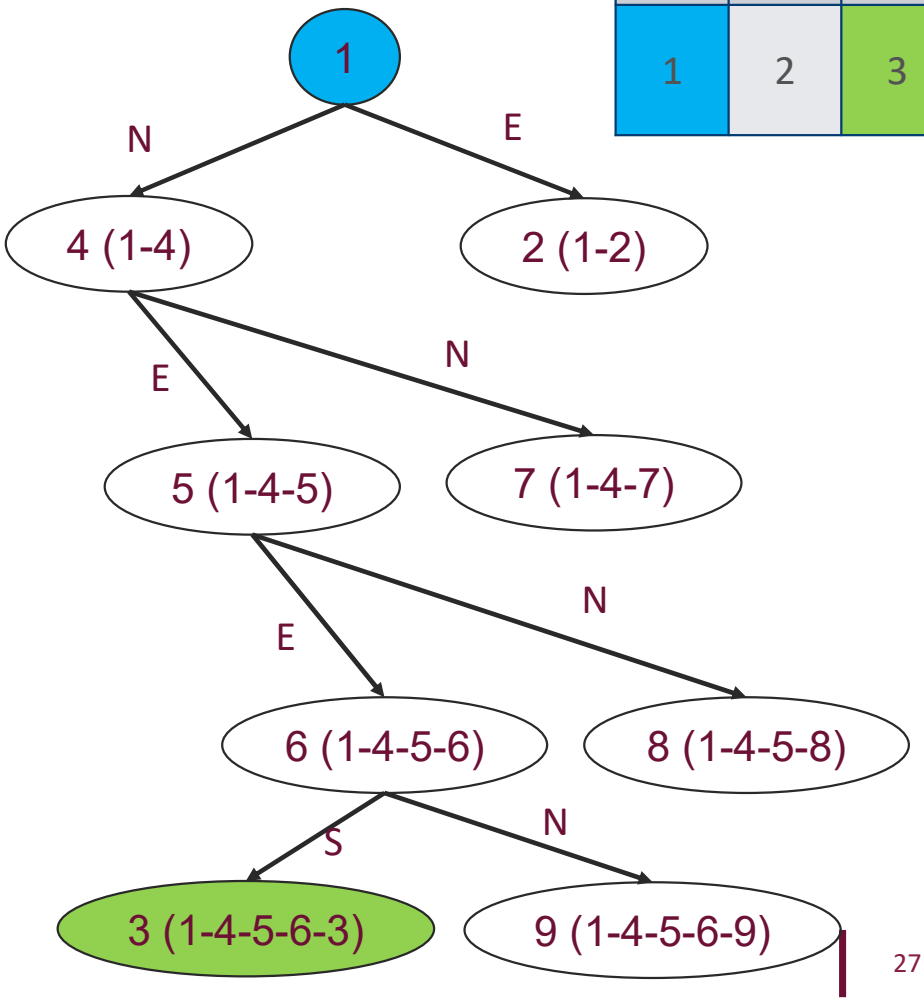
- Store the row and column coordinates of the blank space
- Based on the action chosen (up, down left or right) find the new row and column coordinates of blank space
- Swap blank space with the number currently in the new row and column coordinates

DEPTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Question: Starting from 1 find a way to get to 3?

Strategy: Expand the deepest node first

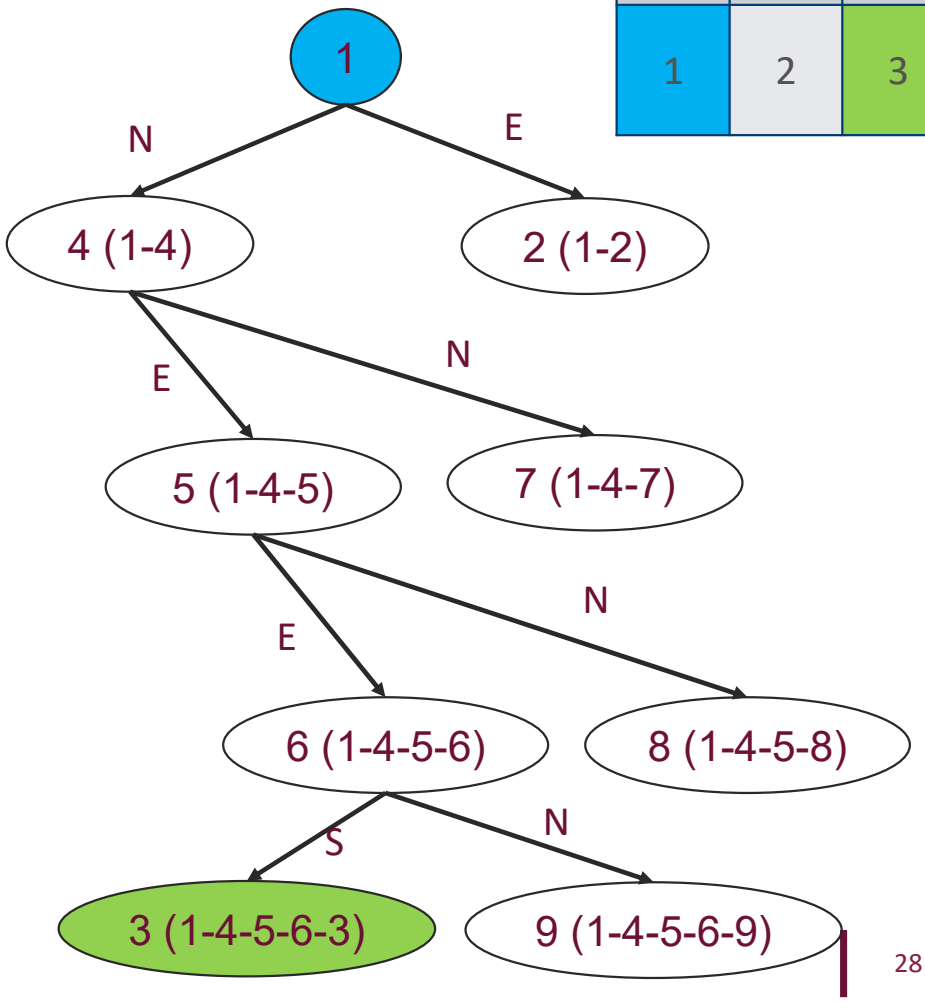


DEPTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Question: Starting from 1 find a way to get to 3?

Strategy: Expand the deepest node first



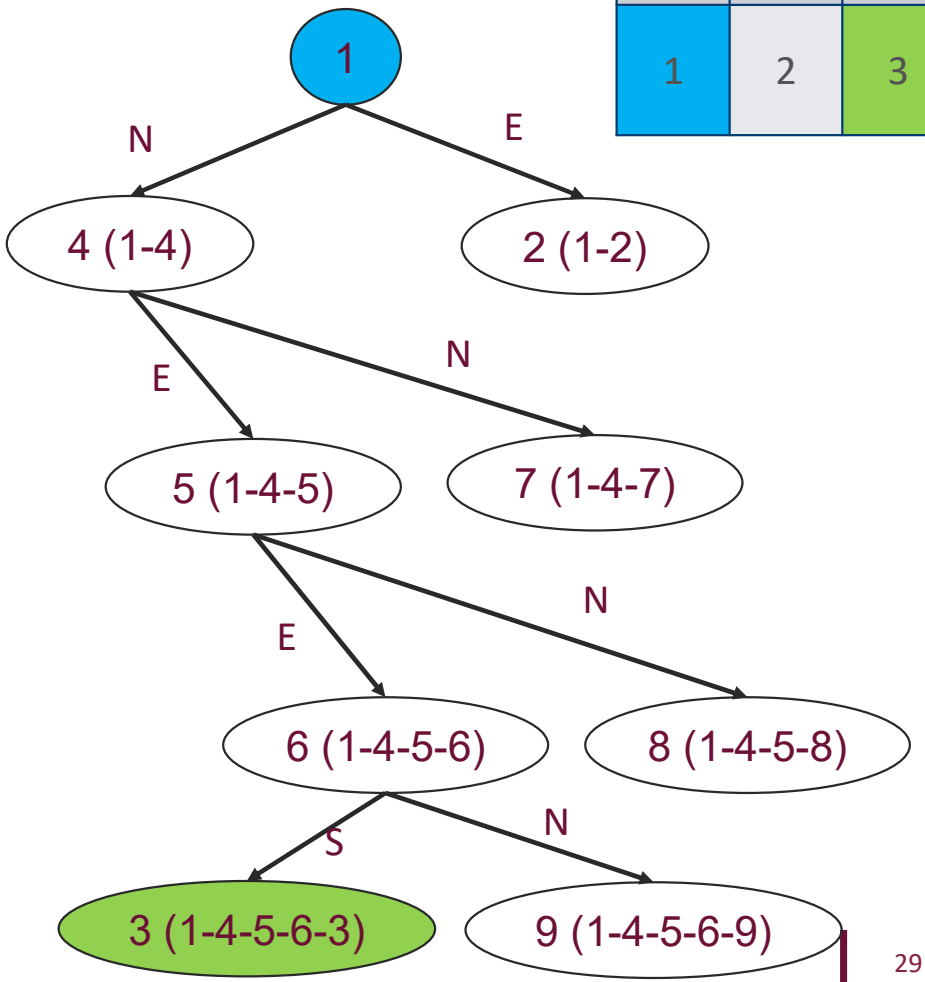
DEPTH FIRST SEARCH

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

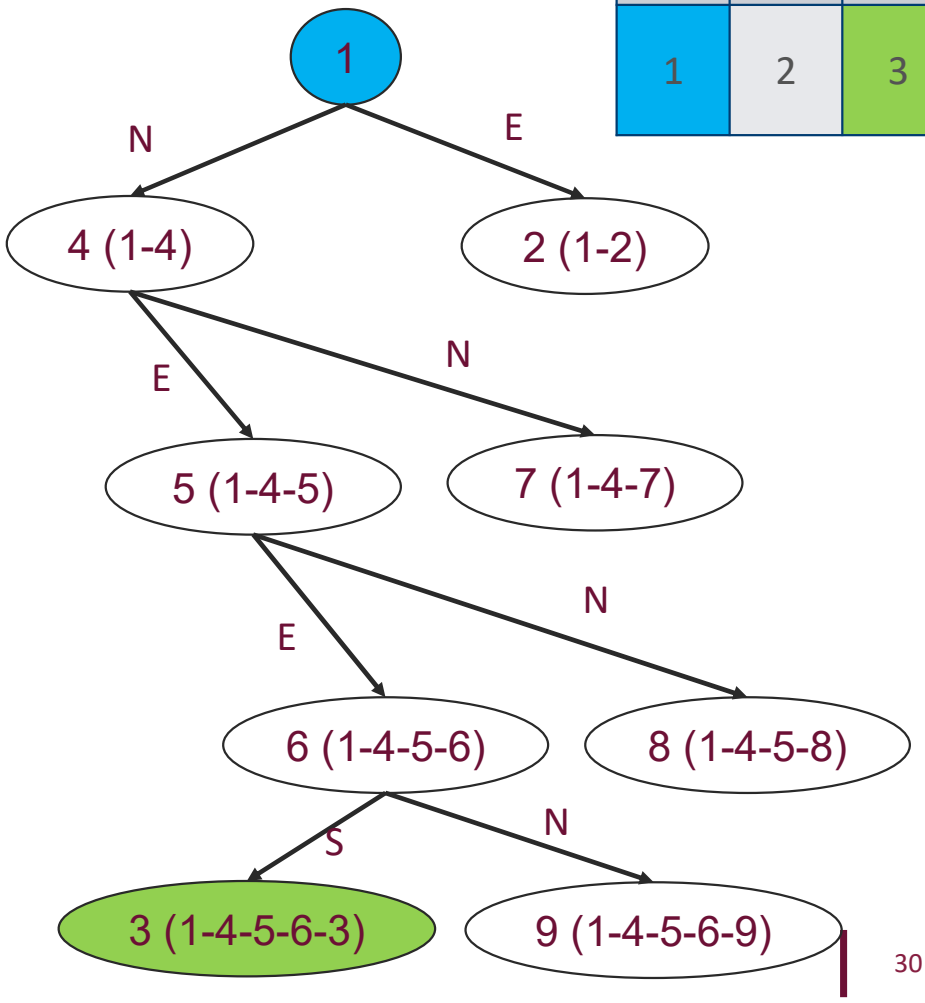


DEPTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Questions:

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



DEPTH FIRST SEARCH

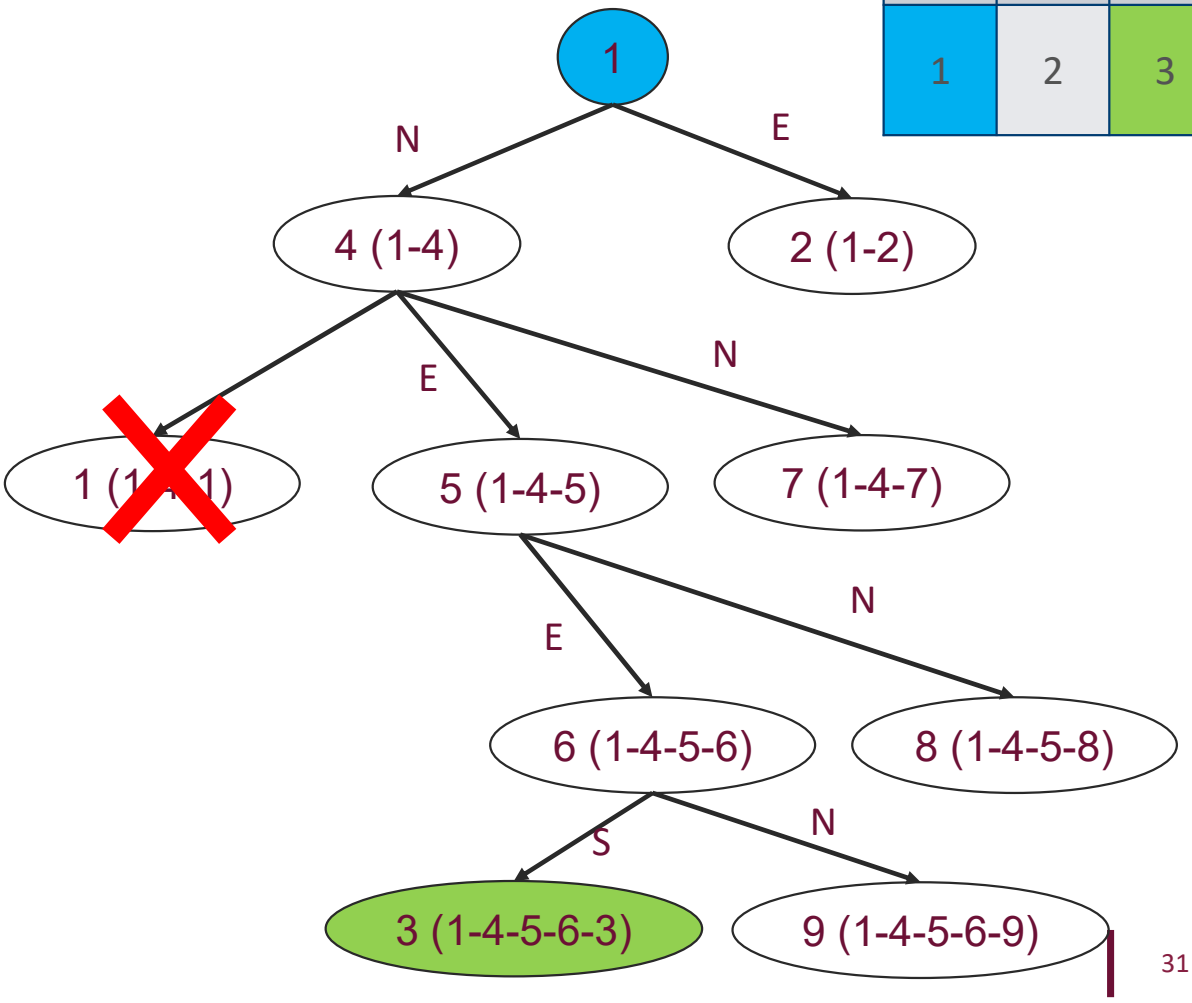
7	8	9
4	5	6
1	2	3

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



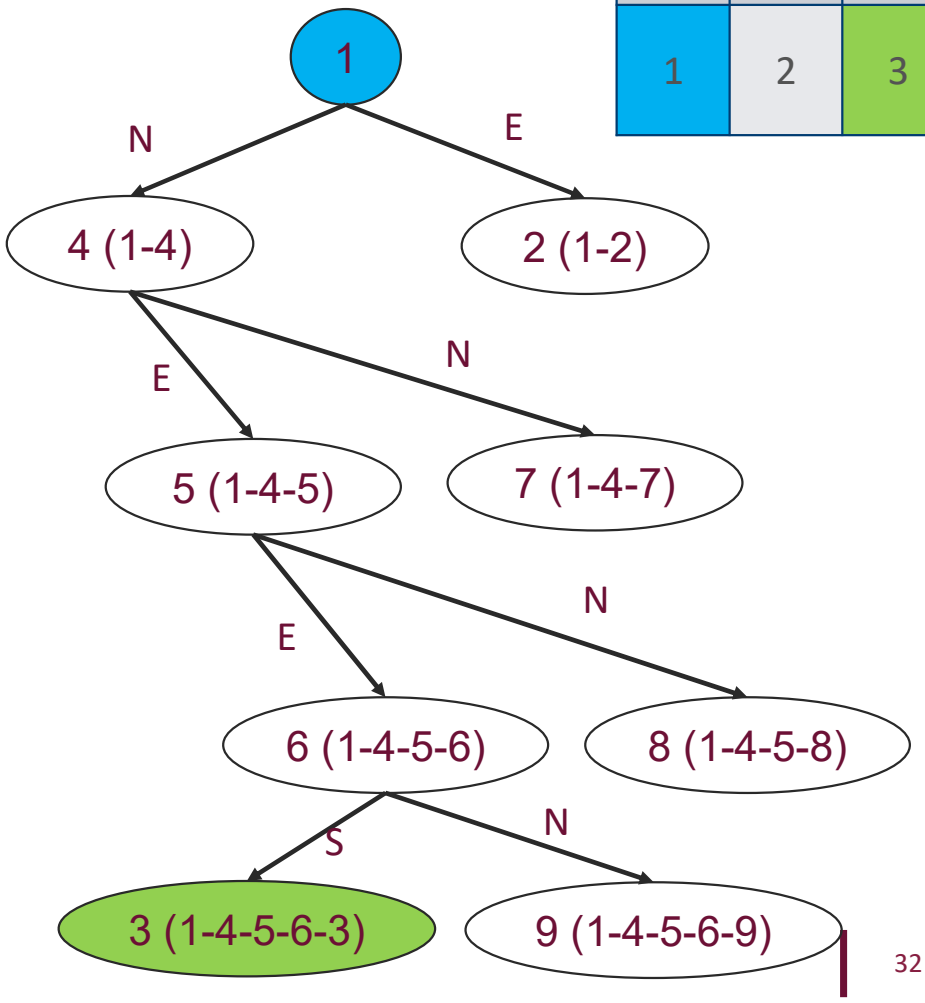
DEPTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

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

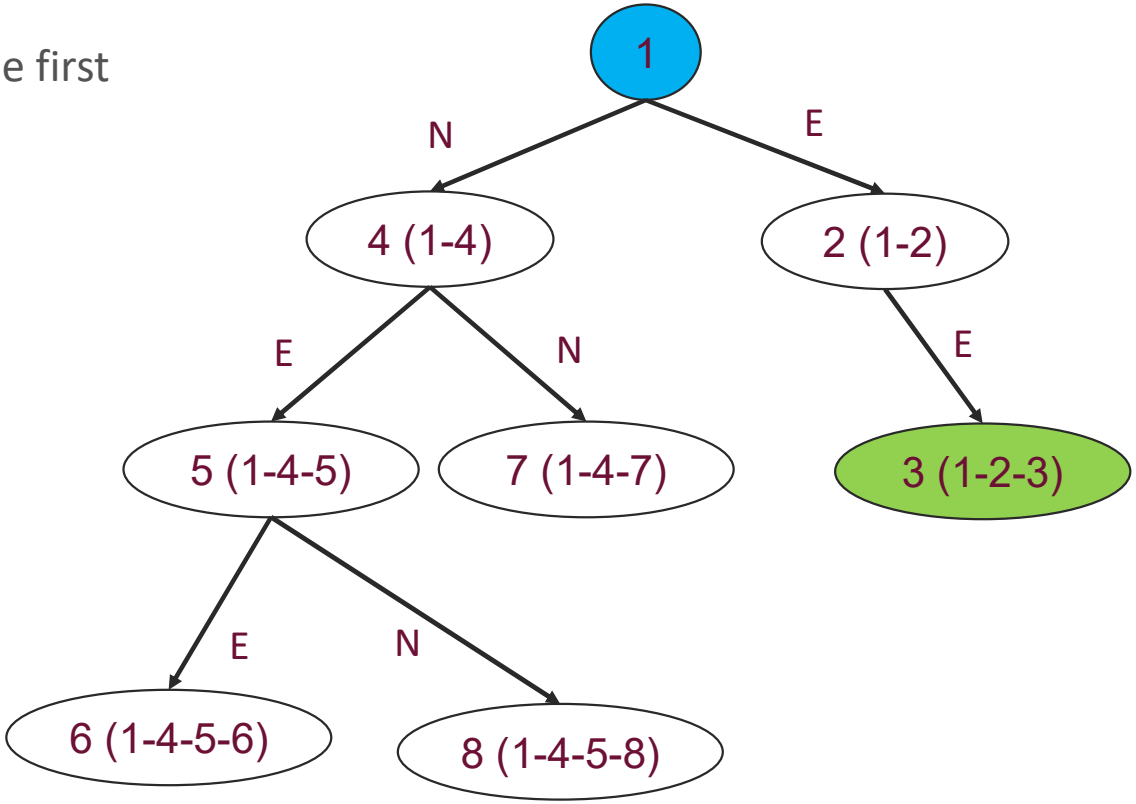


BREADTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Question: Starting from 1 find a way to get to 3?

Strategy: Expand the shallowest node first



BREADTH FIRST SEARCH

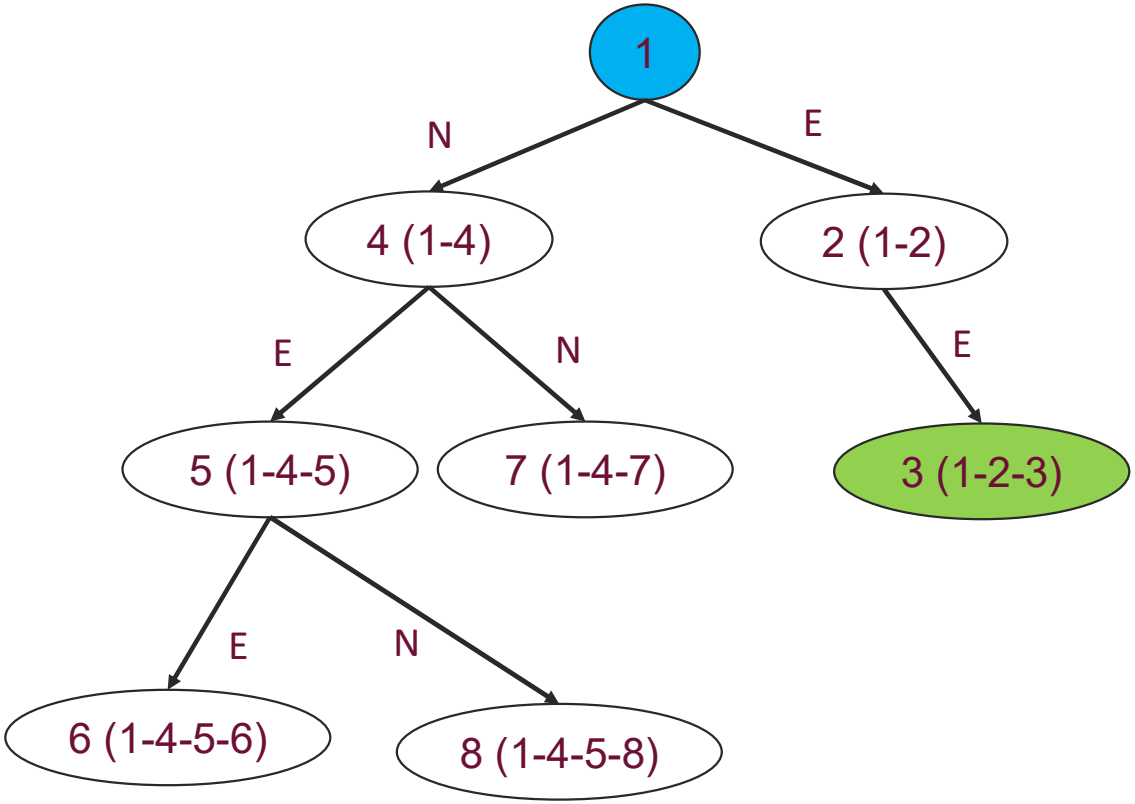
7	8	9
4	5	6
1	2	3

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

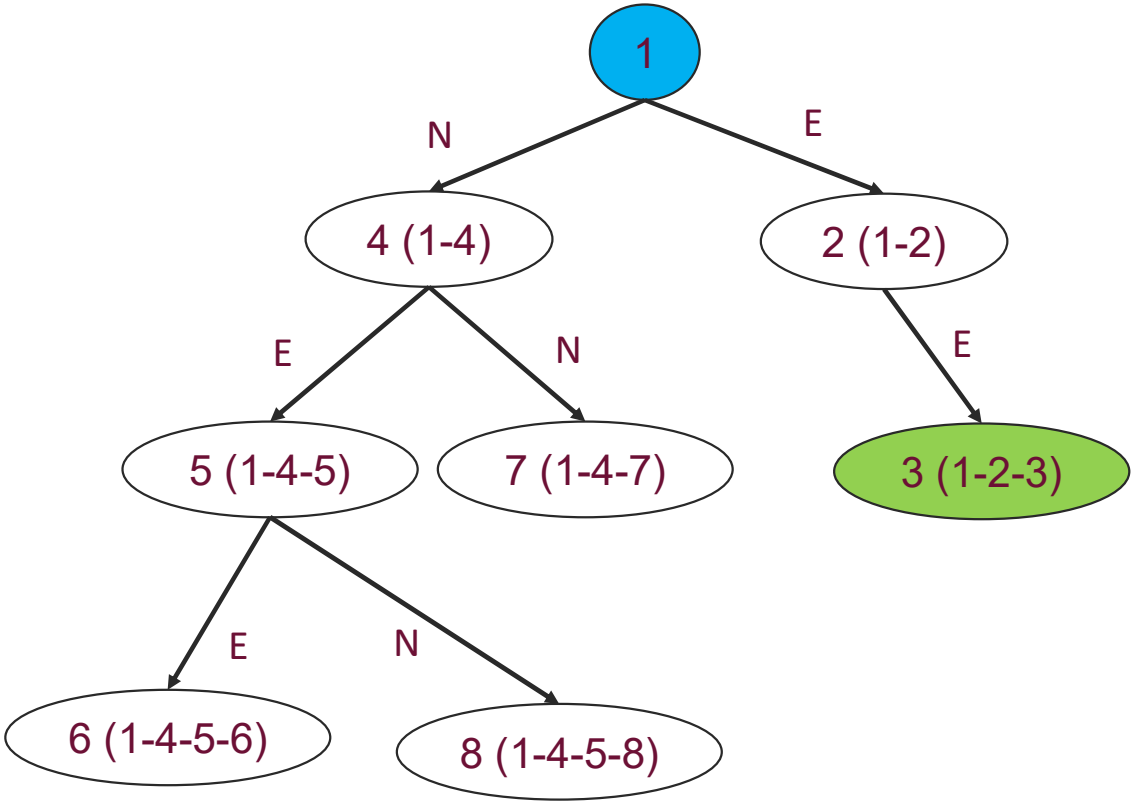


BREADTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Questions:

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



BREADTH FIRST SEARCH

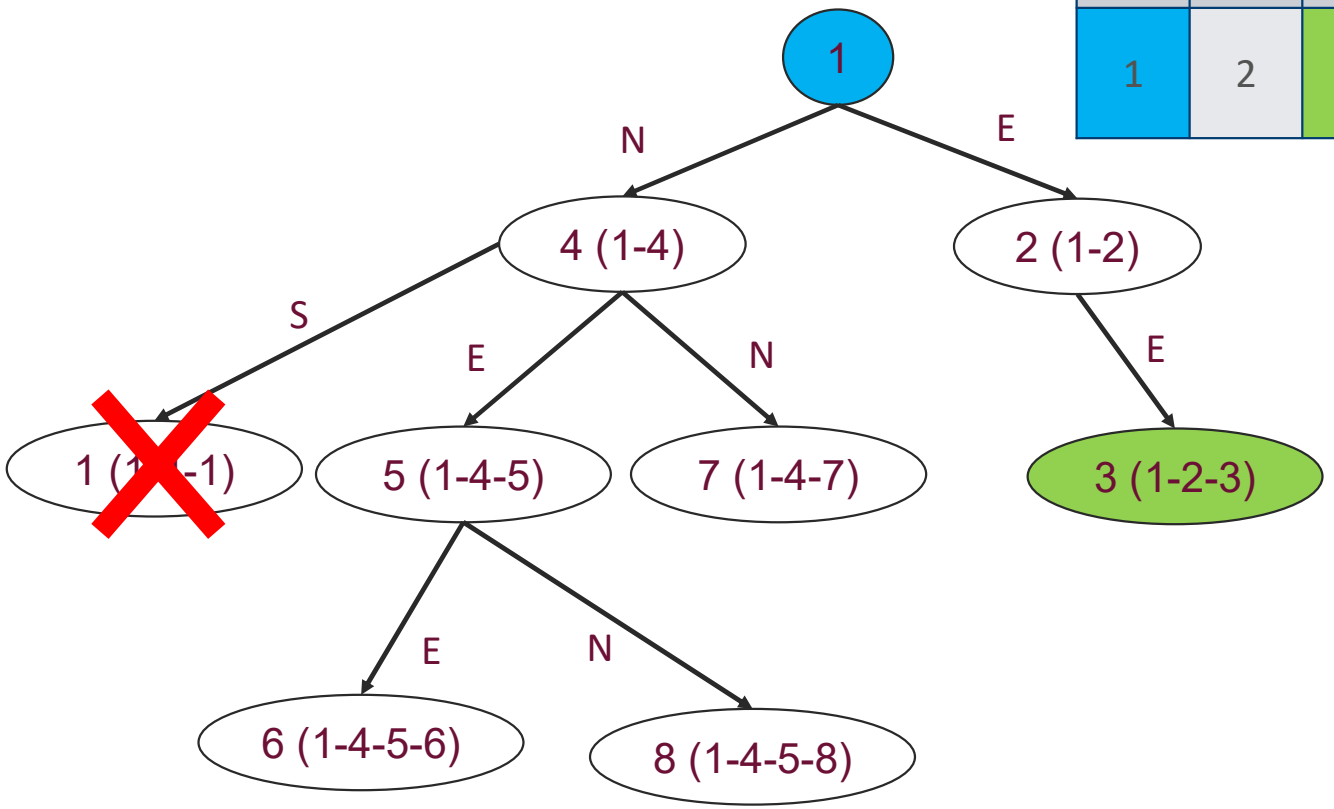
7	8	9
4	5	6
1	2	3

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



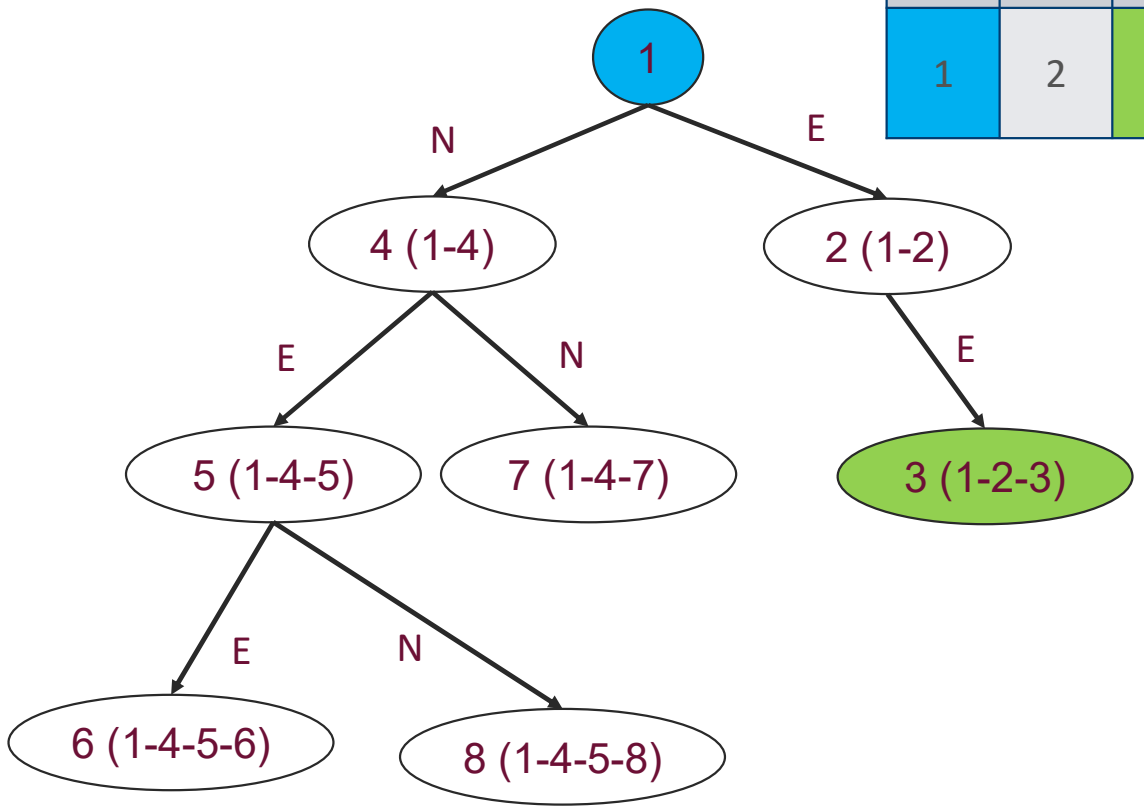
BREADTH FIRST SEARCH

7	8	9
4	5	6
1	2	3

Questions:

- Is Breadth first optimal?

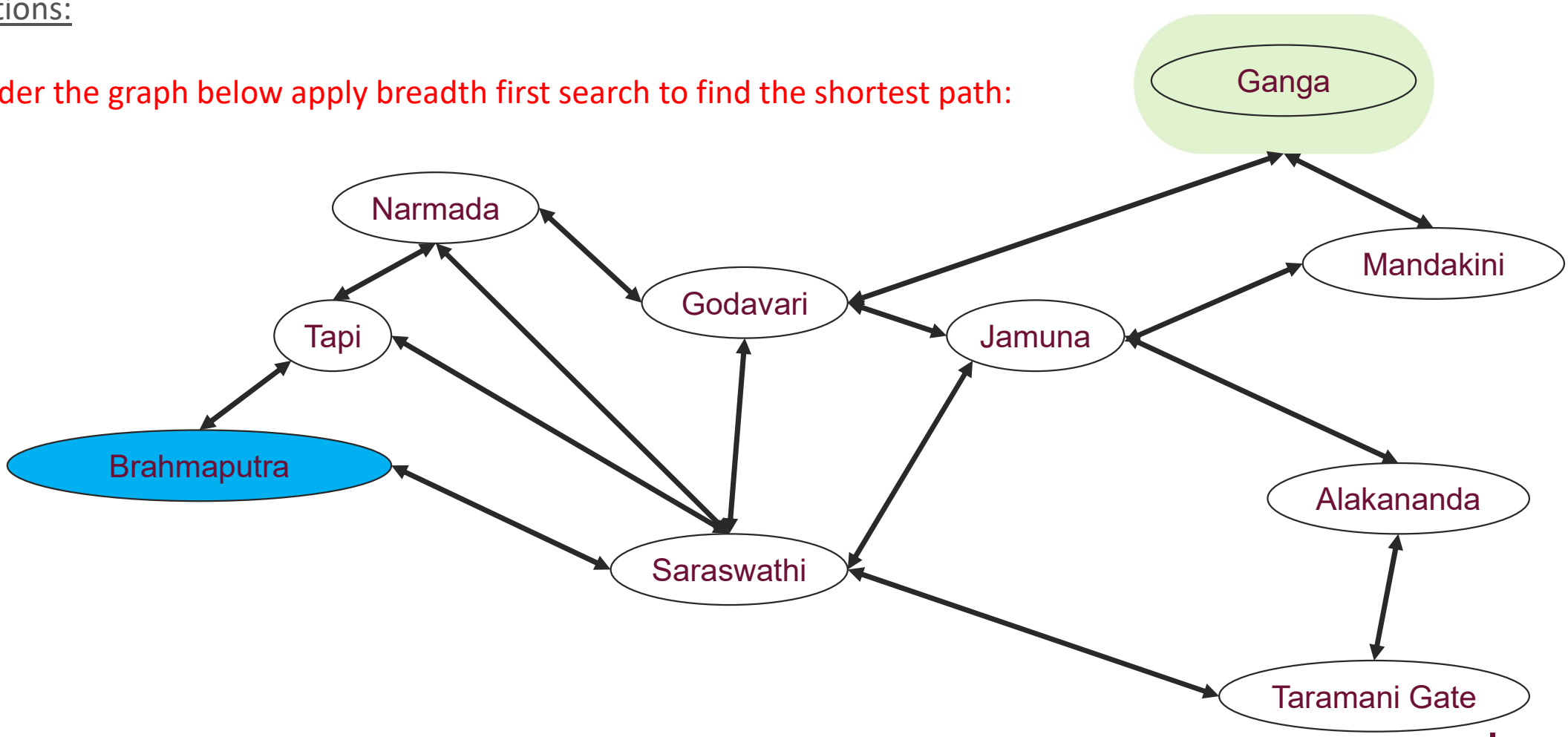
Yes, since we expand nodes at the same level first the path to goal will be optimal, under the assumption each step is unit cost



BREADTH FIRST SEARCH

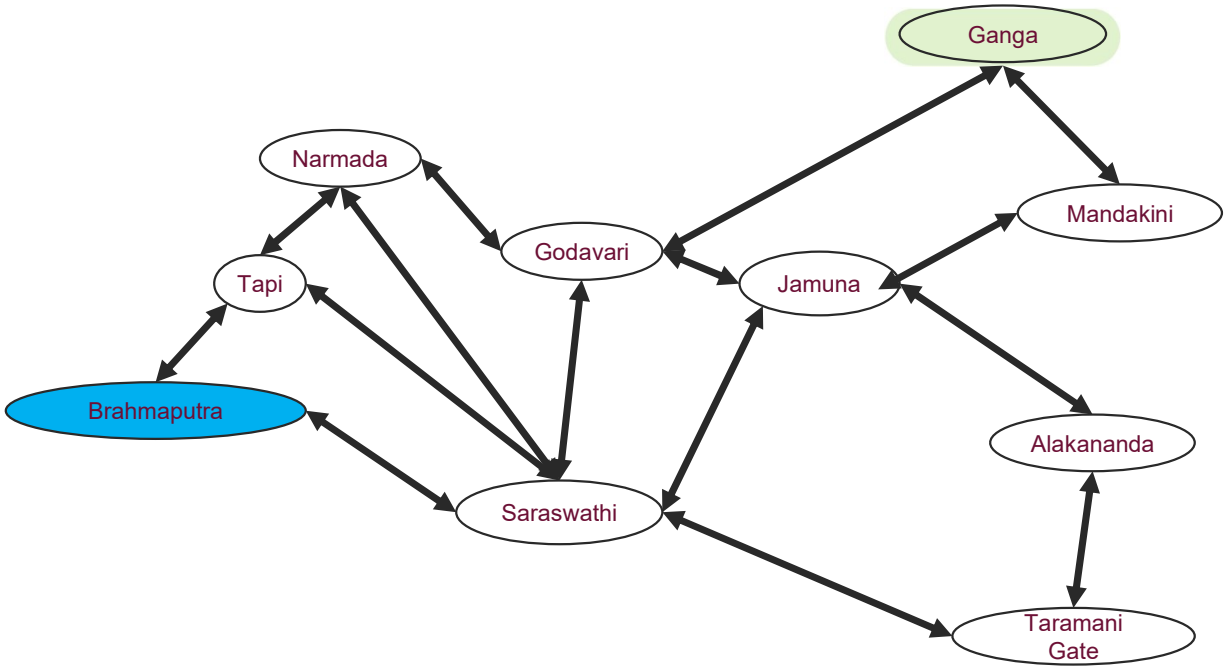
Questions:

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



BREADTH FIRST SEARCH

Strategy: Expand the shallowest node first



BREADTH FIRST SEARCH

Strategy: Expand the shallowest node first

Fringe (B0): pop B0

Fringe (T1,S1): pop T1

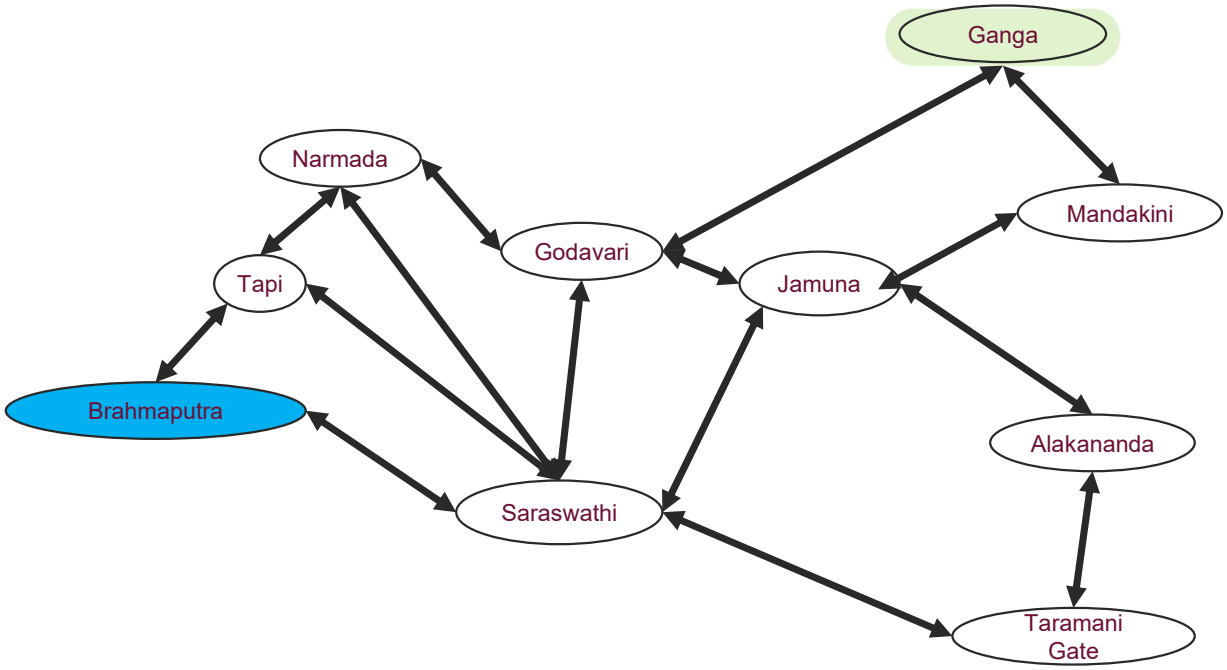
Fringe (S1, N2): pop S1

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

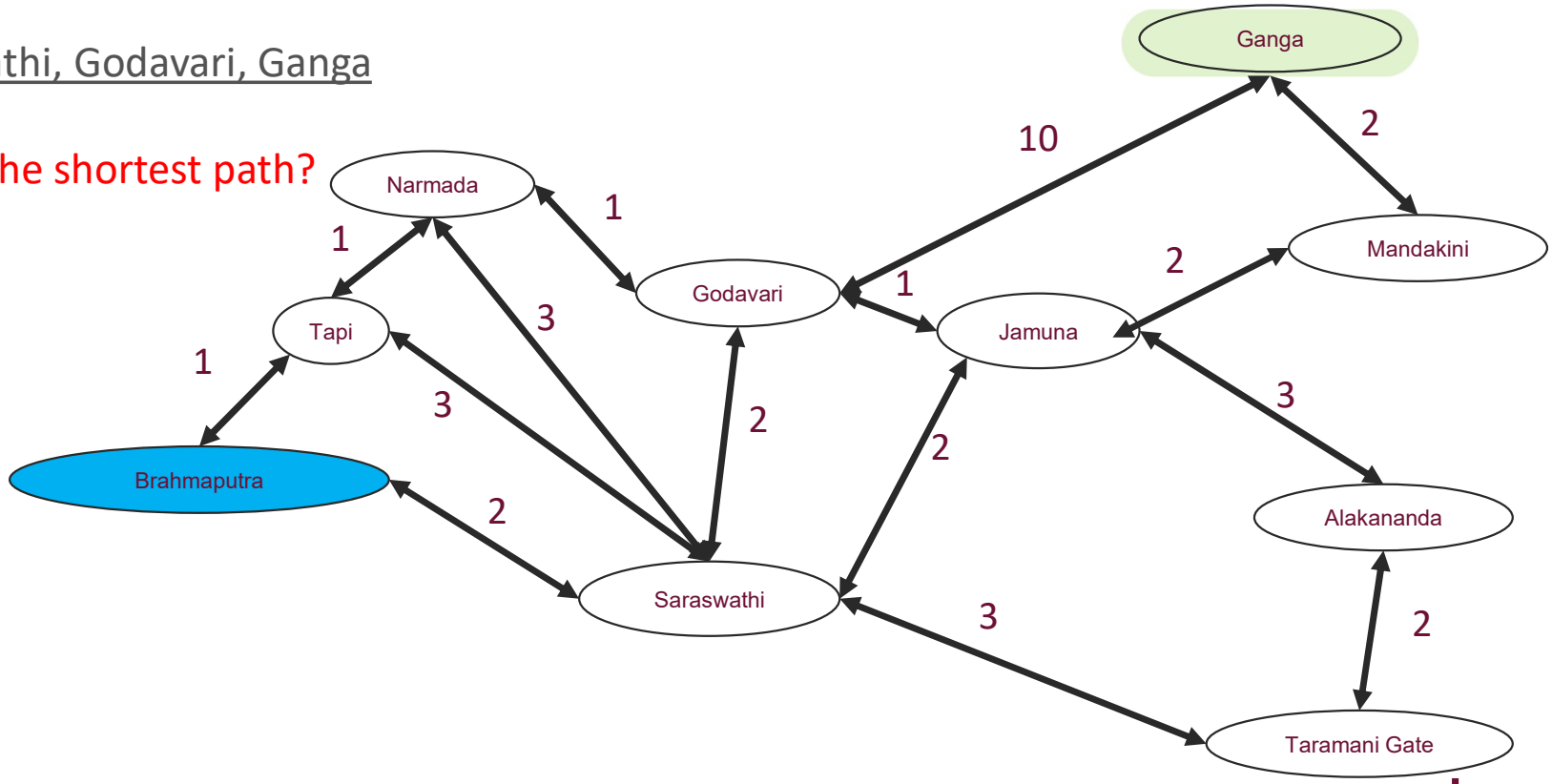
BREADTH FIRST SEARCH

Questions:

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

Solution Brahmaputra, Saraswathi, Godavari, Ganga

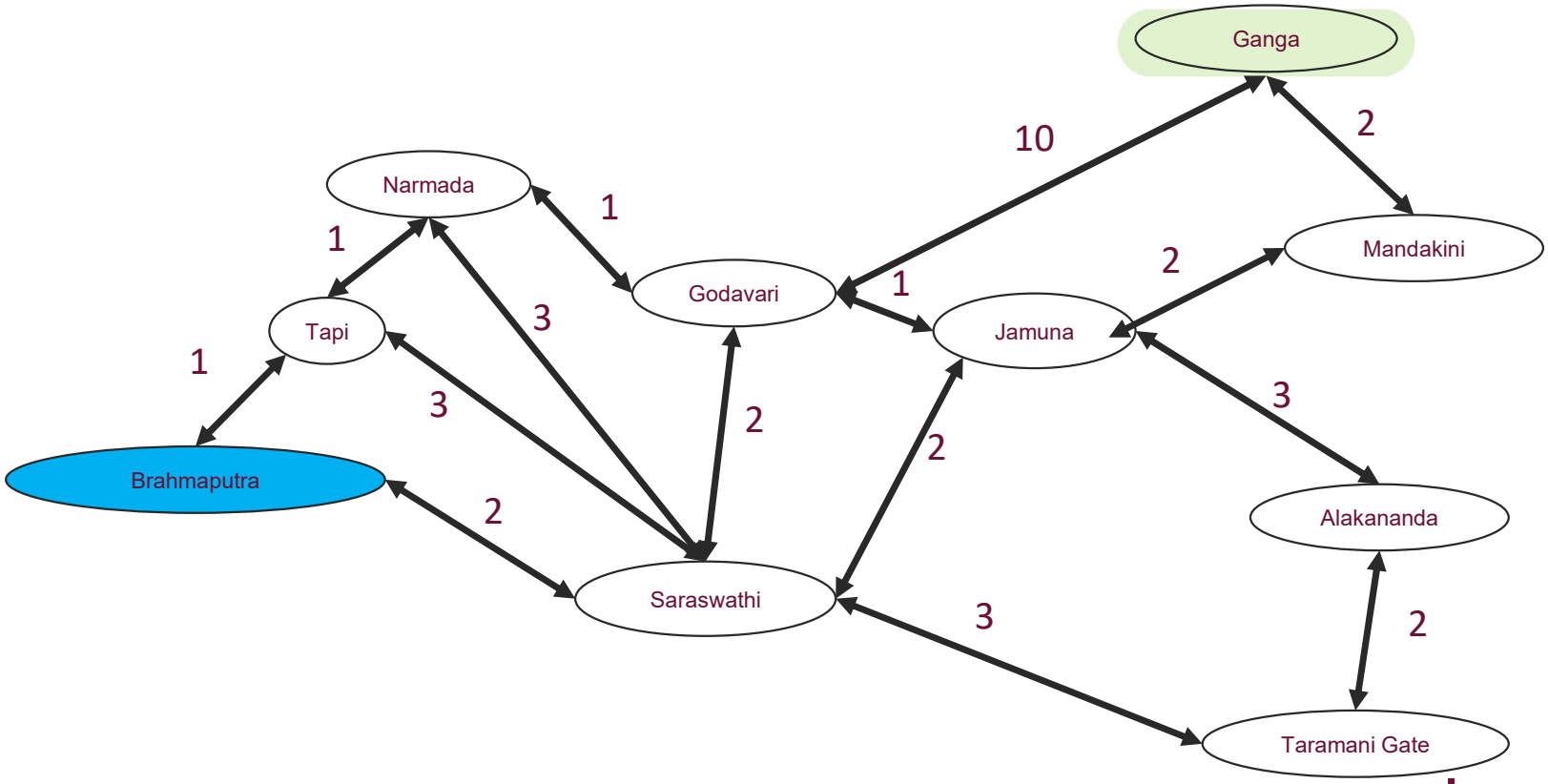
With arc costs added, is it still the shortest path?



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



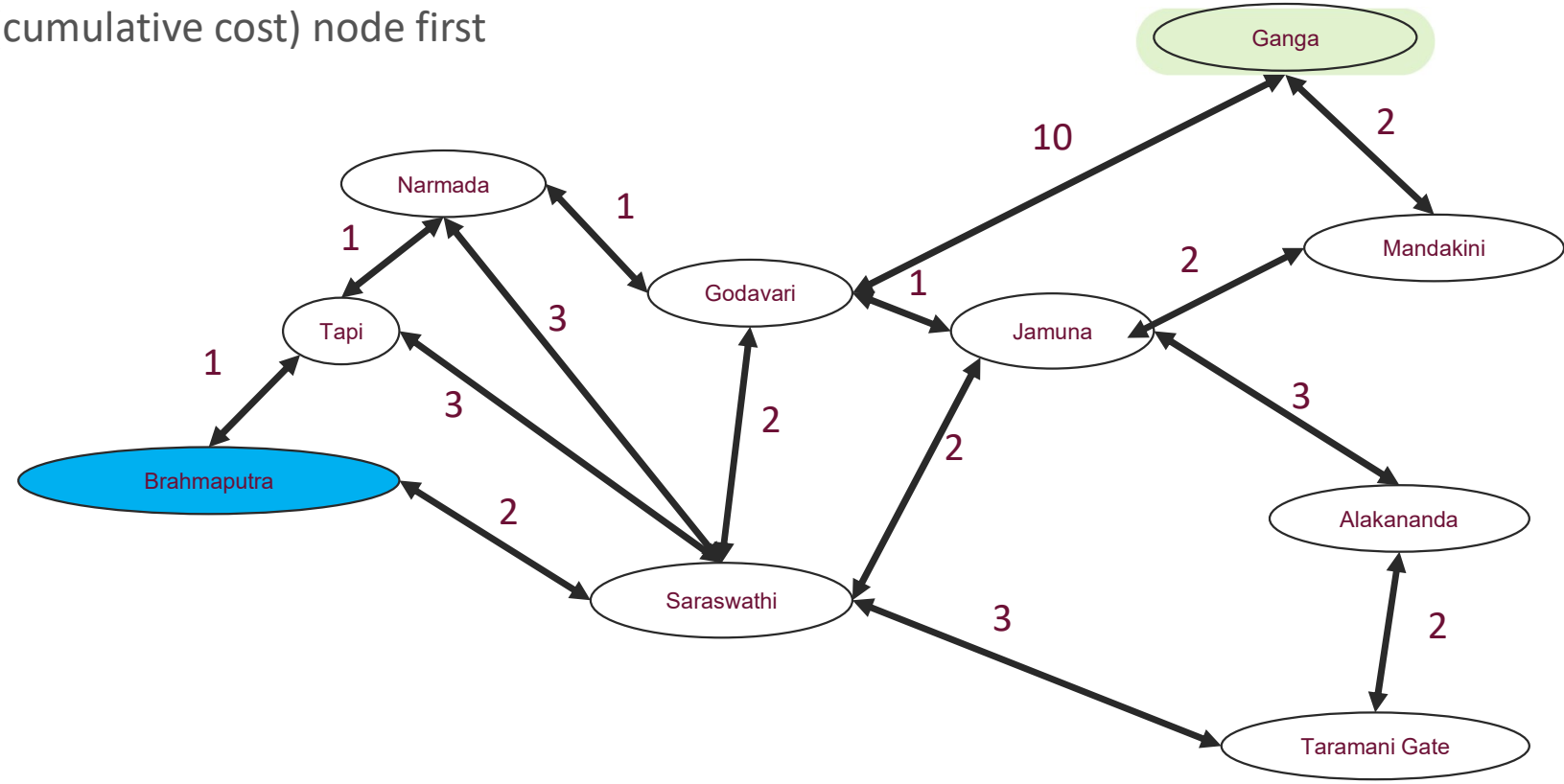
UNIFORM COST SEARCH

Pseudo code:

```
function Uniform Cost search ( problem) returns a solution, or failure
        create empty fringe and closed set
        add the initial state of problem to fringe with a cost of 0
        loop do
                if there are no candidates for expansion in the fringe then return failure
                pop node with least cost from fringe
                if the popped node is goal state, then return the corresponding solution
                else
                add the popped node to closed set
                expand the popped node by performing all possible actions
                for each resulting child node:
                        compute cost = popped node cost + cost of action to get to child
                        add the child node to the fringe
                        except:
                                if child node exists in closed set
                                if duplicate child node exists in fringe:
                                        store the one with lower cost
```

UNIFORM COST SEARCH

Strategy: Expand the cheapest (cumulative cost) node first



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

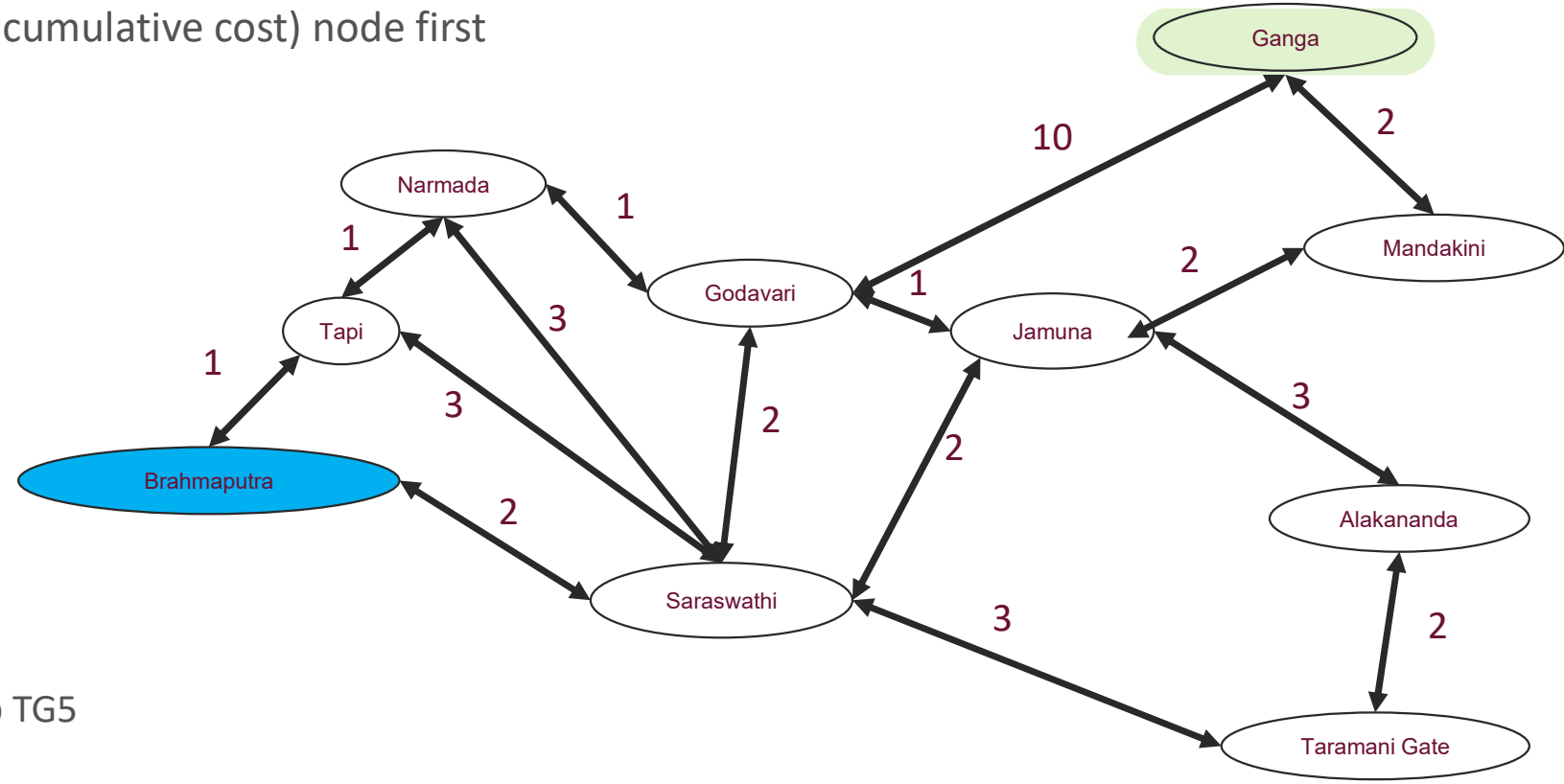
Fringe (J4, TG5, Ganga13): pop J4

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

Fringe (M6, A7,Ganga13): pop M6

Fringe (A7,Ganga8): pop A7

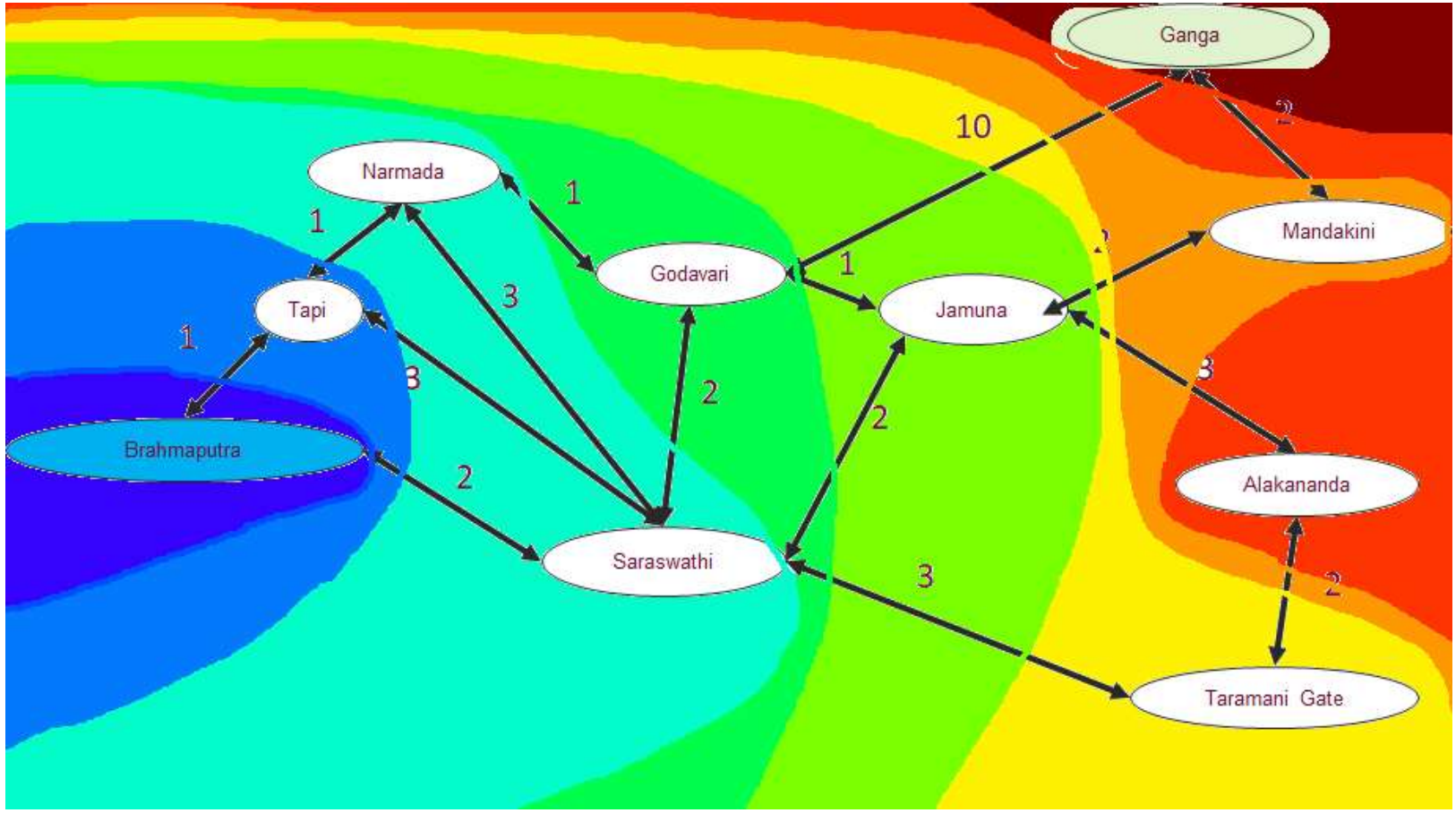
Fringe(Ganga8):pop Ganga8 → goal: solution Brahmaputra, Saraswathi, Jamuna, Mandakini, Ganga



UNIFORM COST SEARCH

Solution: Brahmaputra, Saraswathi, Jamuna, Mandakini, Ganga

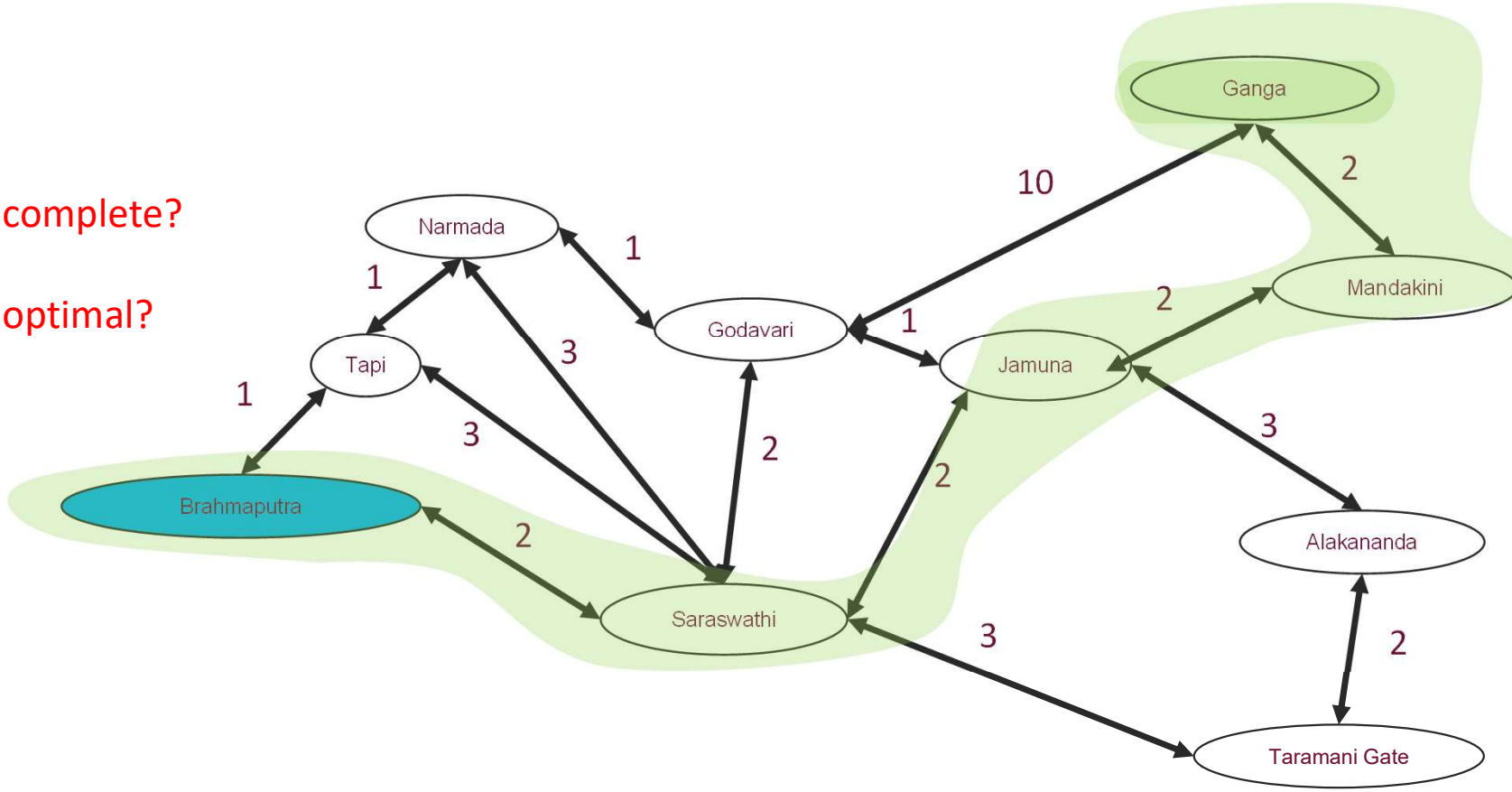
Cost Contours:



UNIFORM COST SEARCH

Questions:

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

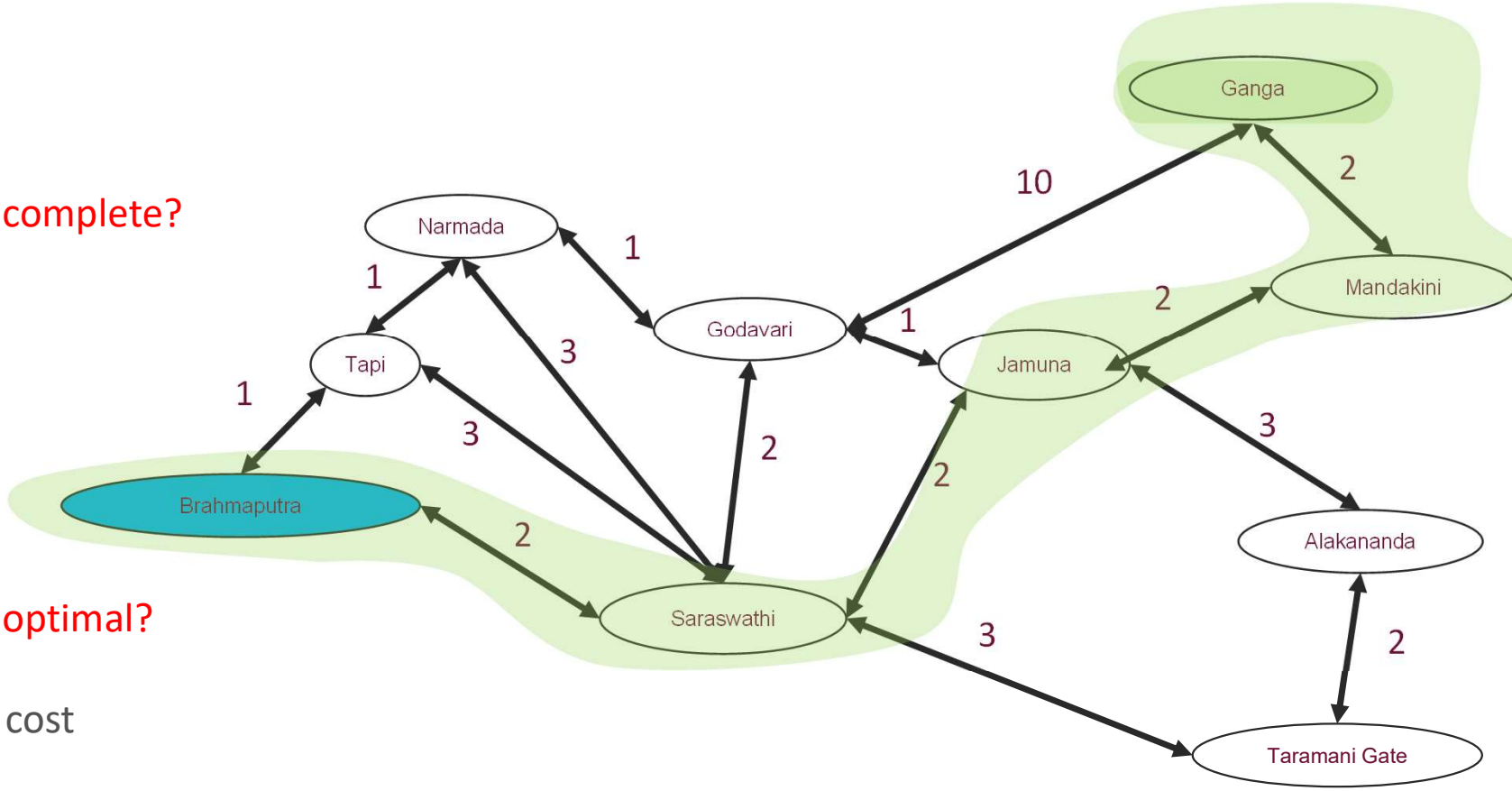


UNIFORM COST SEARCH

Questions:

- Is Uniform Cost Search complete?

Yes



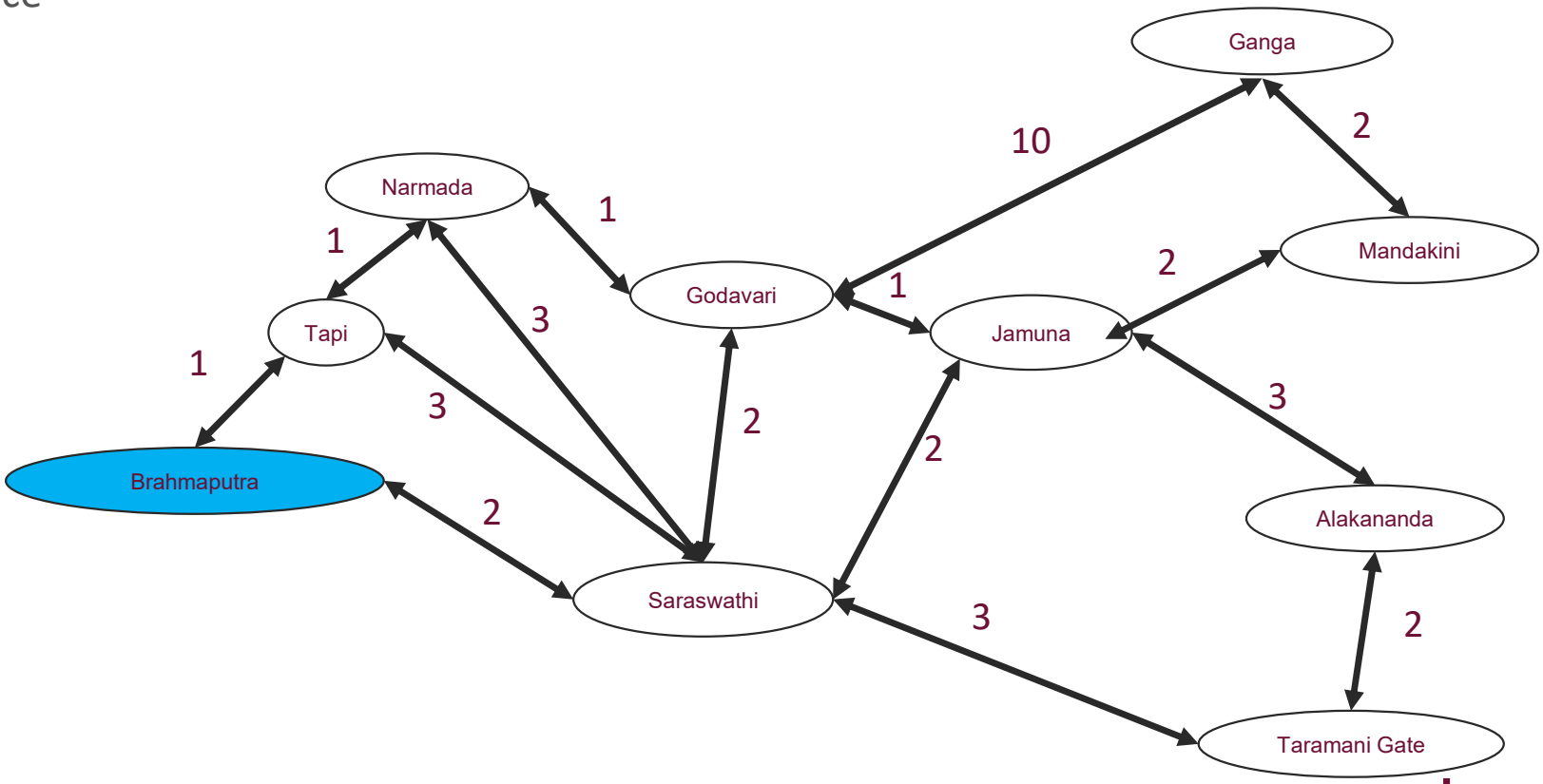
- Is Uniform Cost Search optimal?

Yes, optimal in cumulative cost

DIJKSTRA

Consider a scenario where you care about the shortest path to all nodes on a graph.

Example Parcel delivery service

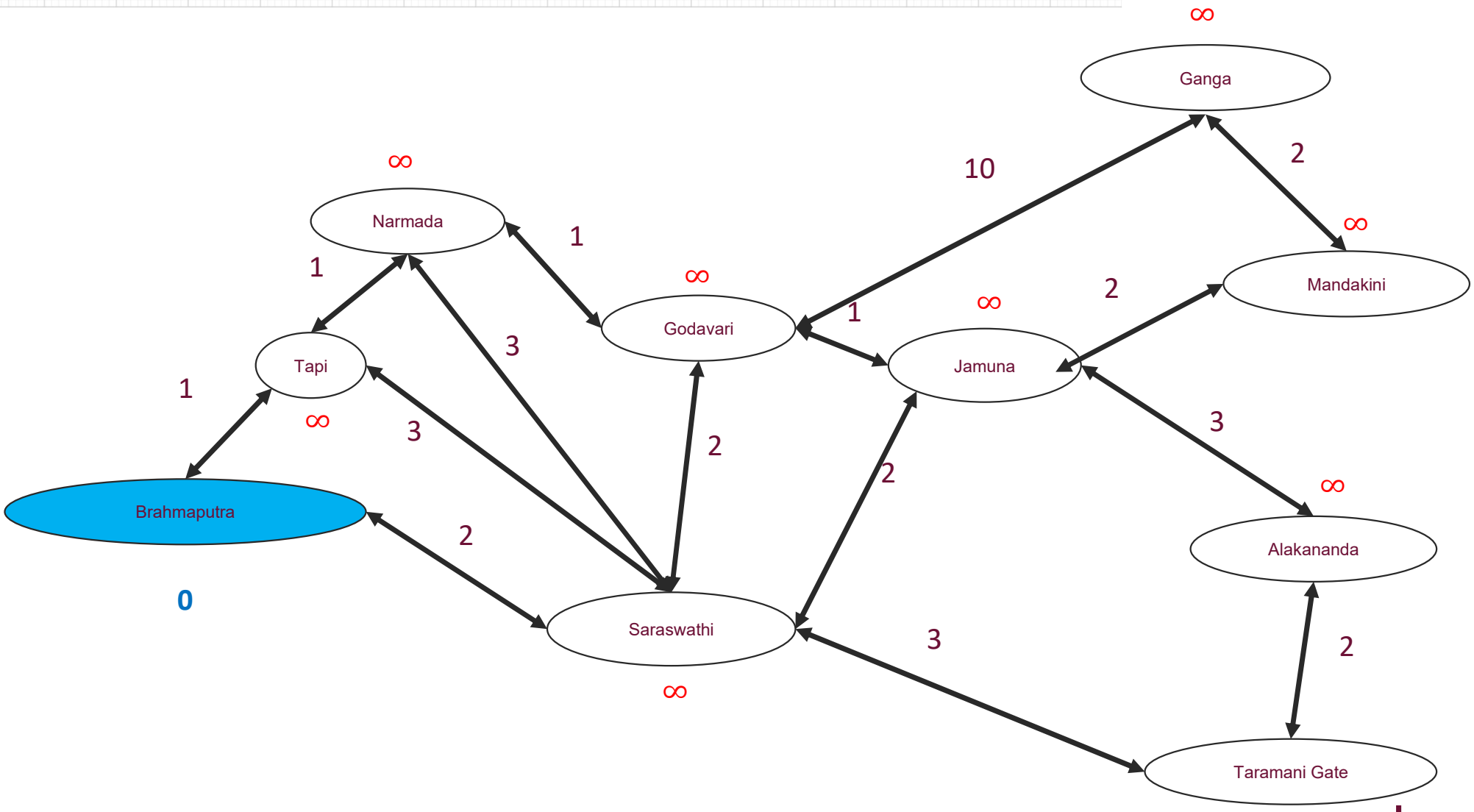


DIJKSTRA

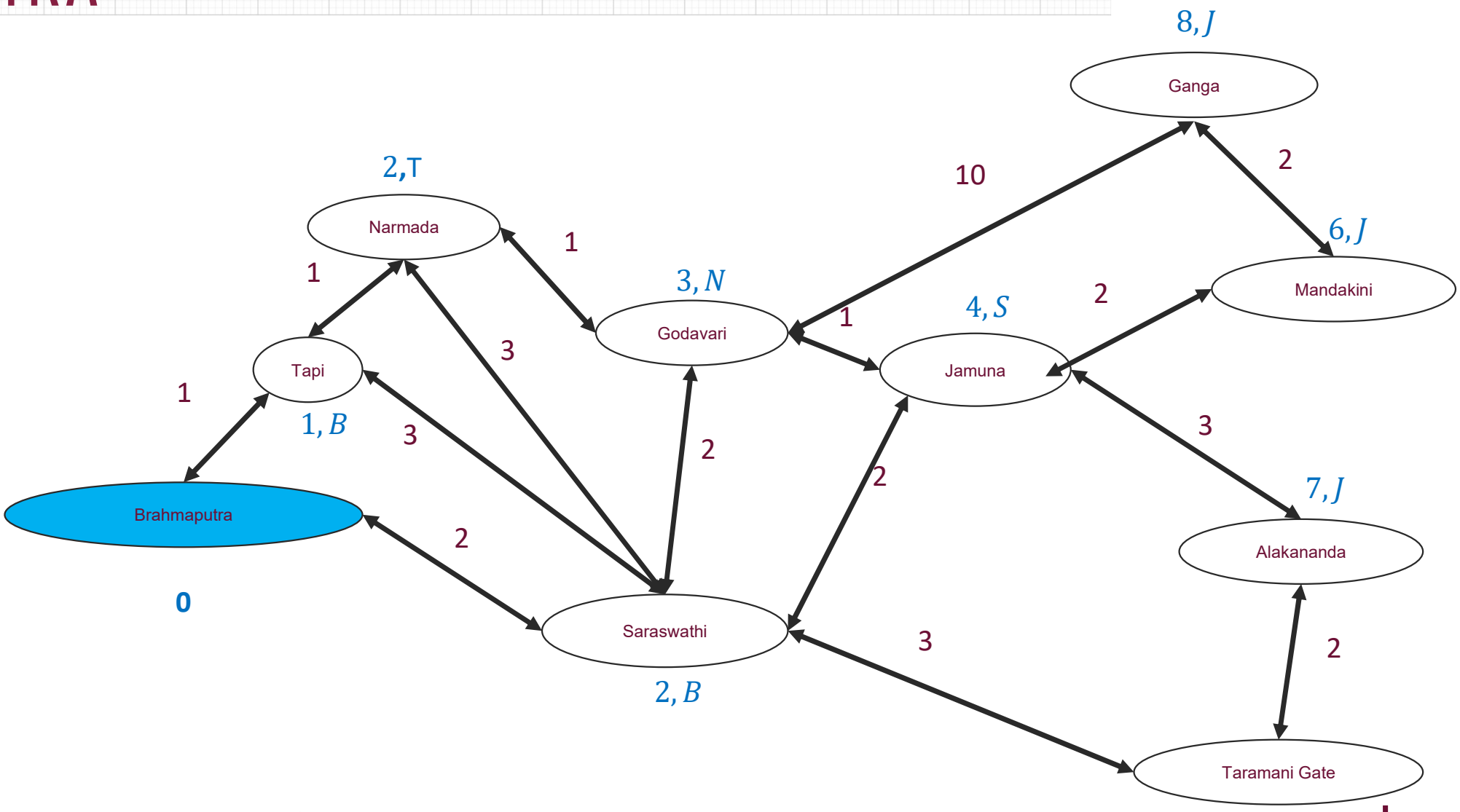
Pseudo code:

```
function Dijkstra( problem) returns a solved tree
        create open list (fringe) and add all nodes of the graph to it
        set the cost of every node in open list to be ∞, except start node. set start node to 0
        while open list is not empty do:
                sort open list and pop the node with least cost
                expand the popped node by performing all possible actions
                for each child node:
                        compute new cost = popped node cost + cost of action to get to child
                add the child node to the fringe
                        except:
                                if child node exists in closed set
                                if duplicate child node exists in fringe:
                                        store the one with lower cost
```

DIJKSTRA


∞
Ganga
10
2
∞
Narmada
1
∞
Mandakini
1
∞
Godavari
1
∞
2
Jamuna
Tapi
3
1
∞
3
2
3
∞
Brahmaputra
2
Alakananda
0
Saraswathi
2
3
∞
2
Taramani Gate
∞

DIJKSTRA



Question: What are the limitations of Dijkstra?

DIJKSTRA

Question: What are the limitations of Dijkstra?

Dijkstra's algorithm is only applicable for explicit graphs where we know all vertices and edges

It needs to store all the nodes on the graph → more memory requirements
Uniform-cost search algorithm on the other hand traverses only the necessary graph parts

ADDITIONAL READING

Question: Is this the best we can do?

What could be the disadvantage of UCS or Dijkstra's algorithm, what can we do improve?

Additional reading:

- Chapter 2 of "Planning Algorithms" by S. M. LaValle
- Chapter 3 of "Principles of Robot Motion, Theory, Algorithms, and Implementation" by Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun