Elective Course
Course Code: CS4103
Autumn 2025-26

Lecture #05

# Artificial Intelligence for Data Science

**Week-2: PROBLEM SOLVING BY SEARCH**

Introduction to Uninformed Search Techniques

Course Instructor:

**Dr. Monidipa Das**

**Assistant Professor**

**Department of Computational and Data Sciences**

**Indian Institute of Science Education and Research Kolkata, India 741246**



# Home Assignments

## Problem-1: Missionaries and Cannibals Problem

- "**Missionaries and Cannibals**" is a problem in which 3 missionaries and 3 cannibals want to cross from the left bank of a river to the right bank of the river. There is a boat on the left bank, but it only carries at most two people at a time (and can never cross with zero people). If cannibals ever outnumber missionaries on either bank, the cannibals will eat the missionaries.
- A state can be represented by a triple, (m c b), where m is the number of missionaries on the left, c is the number of cannibals on the left, and b indicates whether the boat is on the left bank or right bank.

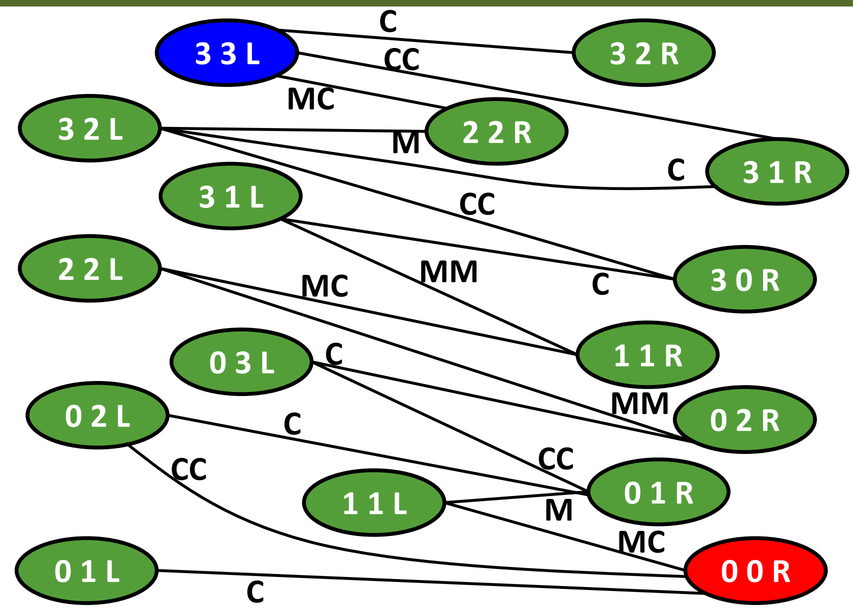



## Problem-1: Missionaries and Cannibals Problem

- For example, the **initial state** is (3 3 L) and the **goal state** is (0 0 R).
- **Operators/Actions are:**
  - MM: 2 missionaries cross the river
  - CC: 2 cannibals cross the river
  - MC: 1 missionary and 1 cannibal cross the river
  - M: 1 missionary crosses the river
  - C: 1 cannibal crosses the river

**Draw a diagram showing all the legal states and transitions from states corresponding to all legal operations.**

## Missionaries and Cannibals Problem: State Space





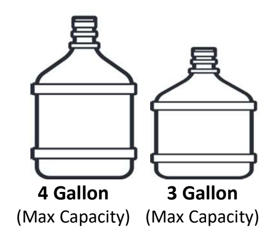

## Problem-2: Water Jug Problem

- You have a 4-gallon and a 3-gallon water jug
- You have a faucet with an unlimited amount of water
- You need to get exactly 2 gallons in 4-gallon jug

- State representation: **(x, y)**
  - x: Contents of 4-gallon jug
  - y: Contents of 3-gallon jug

- **Start state:** (0, 0)
- **Goal state:** (2, n)

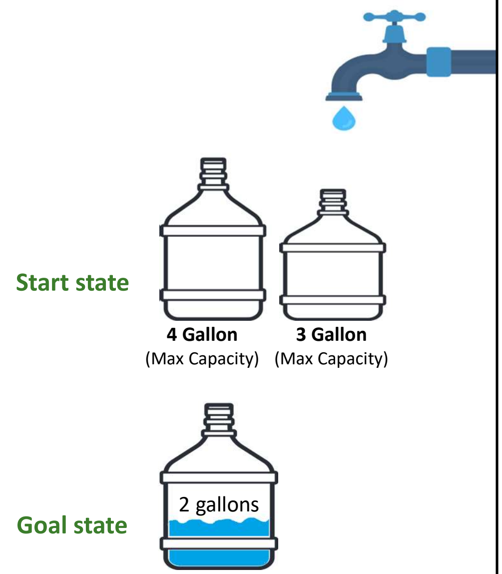
Start state




Goal state

## Problem-2: Water Jug Problem

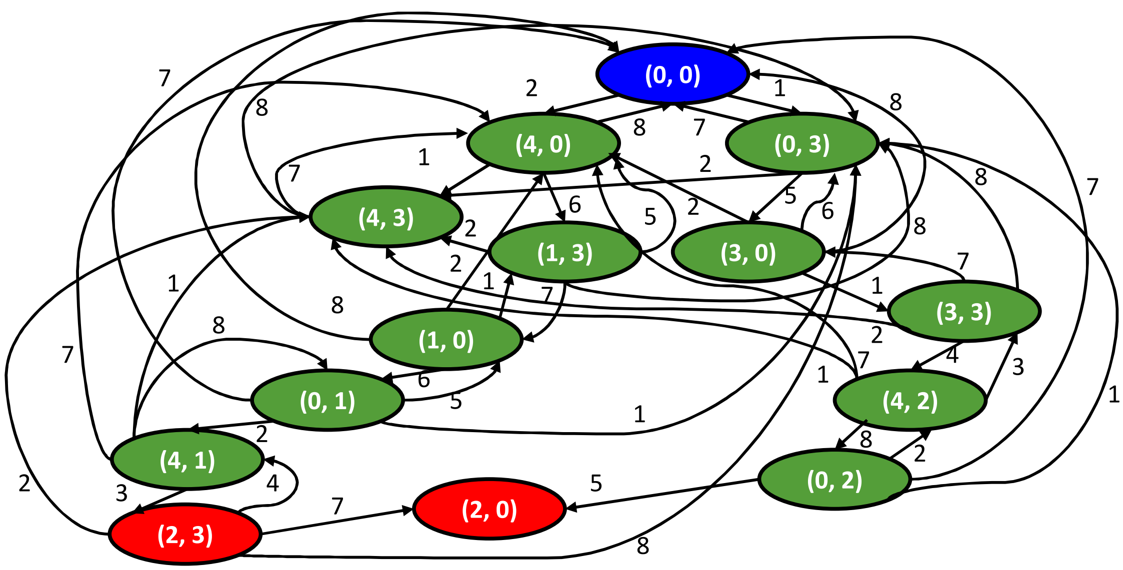
- **Operators/Actions**
  1. Fill 3-gallon from faucet
  2. Fill 4-gallon from faucet
  3. Fill 3-gallon from 4-gallon
  4. Fill 4-gallon from 3-gallon
  5. Empty 3-gallon into 4-gallon
  6. Empty 4-gallon into 3-gallon
  7. Dump 3-gallon down drain
  8. Dump 4-gallon down drain




**Draw a diagram showing all the legal states and transitions from states corresponding to all legal operations.**



## State Space: Water Jug Problem (4 Gallon 3 Gallon)

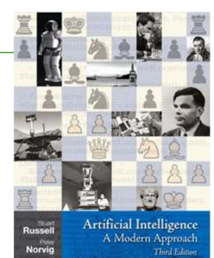


# Basic Search Algorithm

Let L be a list containing the initial state
(L=the **fringe**)
Loop
    if L is empty return *failure*
    Node ← **select**(L)
    if Node is a goal
        then return Node
        (*the path from initial state to Node*)
        else apply all applicable operators to Node
        and **merge** the newly generated states into L



# General Tree Search
## (informal description)




**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

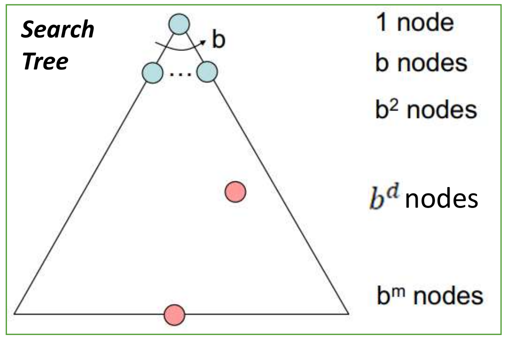
**Important ideas:**
✓Fringe or Frontier
✓Expansion
✓Exploration strategy

**Main question:**
✓which fringe nodes to explore

# Basic Search Algorithm

11

Let L be a list containing the initial state
(L=the **fringe**)
Loop
  if L is empty return *failure*
  Node ← **select**(L)
  if Node is a goal
    then return Node
    (*the path from initial state to Node*)
    else apply all applicable operators to Node
    and **merge** the newly generated states into L




**b** is the ***branching factor***
**m** is the ***maximum depth***
**d** is the ***depth containing another solution***

Number of nodes in entire tree?
$1 + b + b^2 + \cdots + b^m = O(b^m)$



# Search Strategy

12

- A **strategy** is defined by picking the <u>order of node expansion</u>
- **Strategies are evaluated along the following dimensions:**
  - **Completeness** – does it always find a solution if one exists?
  - **Time complexity** – number of nodes generated/expanded
  - **Space complexity** – maximum nodes in memory
  - **Optimality** – does it always find a least-cost solution?

# Search Strategy [contd.]

- **Time and space complexity are measured in terms of:**
  – b – ***maximum branching factor*** of the search tree (may be infinite)
  – d – ***depth of the least-cost solution***
  – m – ***maximum depth of the search tree*** (may be infinite)



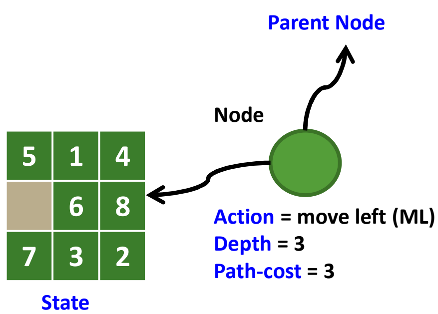
# Search Strategy [contd.]

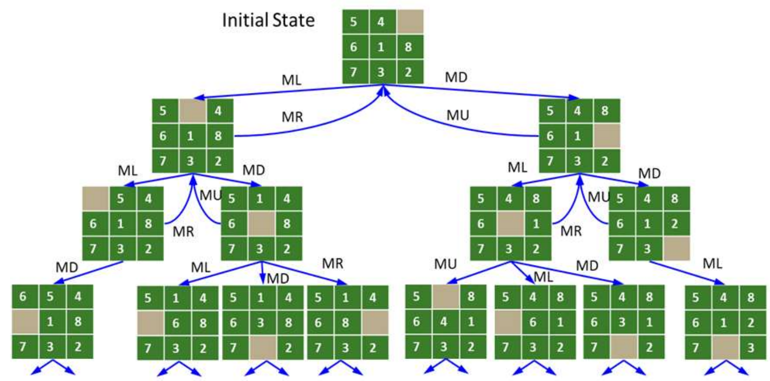
- **Blind/Uninformed Search** (does not use additional information other than what given in problem definition)
  – Depth first search (DFS)
  – Breadth first search (BFS)
  – Iterative deepening search (IDS)
  – Uniform-Cost search (UCS)
- **Informed/Heuristic Search** (use problem specific or domain related information)
  – Best First Search
  – A*,
  – Hill climbing
  – Simulated Annealing

# Implementation: states vs. nodes

15

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost *g(n)*, depth






# General Tree Search
## (formal description)

16

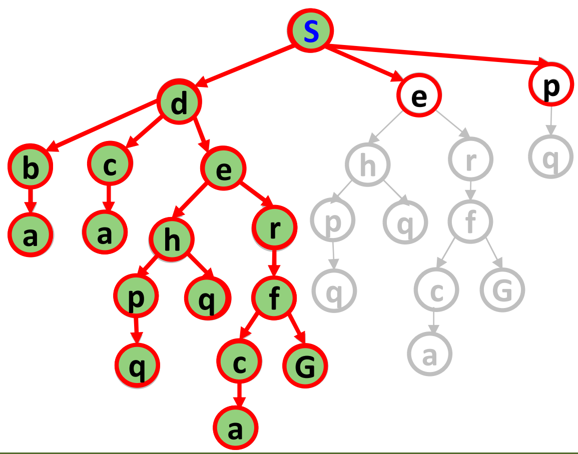
```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
       if fringe is empty then return failure
       node ← REMOVE-FRONT(fringe)
       if GOAL-TEST[problem] applied to STATE(node) succeeds return node
       fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
   successors ← the empty set
   for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
       s ← a new NODE
       PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
       PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
       DEPTH[s] ← DEPTH[node] + 1
       add s to successors
   return successors
```

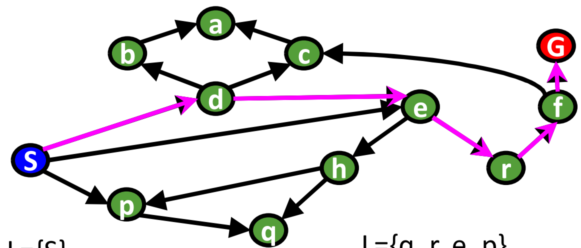
Dr. Monidipa Das, Department of CDS, IISER Kolkata

# Depth-First Search

17

- **Strategy:** expand a **deepest node first**
- **Implementation: Fringe is a LIFO stack**





L={S}
L={d, e, p}
L={b, c, e, e, p}
L={a, c, e, e, p}
L={c, e, e, p}
L={a, e, e, p}
L={e, e, p}
L={h, r, e, p}
L={p, q, r, e, p}
L={q, q, r, e, p}
L={q, r, e, p}
L={r, e, p}
L={f, e, p}
L={c, G, e, p}
L={a, G, e, p}
L={G, e, p}

**G (goal state) found!**

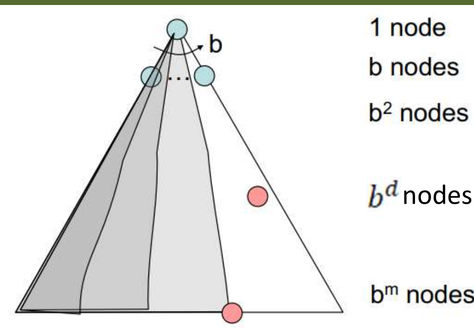
**Path:**
S→ d→ e→ r→ f→G

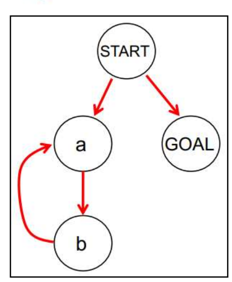
Dr. Monidipa Das, Department of CDS, IISER Kolkata

# Depth-First Search (DFS) Properties

18

- **How many nodes does DFS expand?**
  - DFS expand some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- **How much space does the fringe take?**
  - Only has siblings on path to root, so $O(bm)$
- **Is DFS complete?**
  - only if we prevent cycles
  - **m** could be infinite. So, No.
  - Infinite paths and infinite search space makes DFS incomplete
- **Is it optimal?**
  - No, it finds the "leftmost" solution, regardless of depth or cost

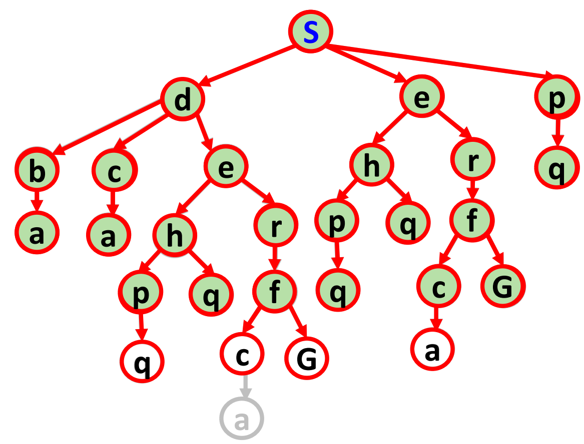


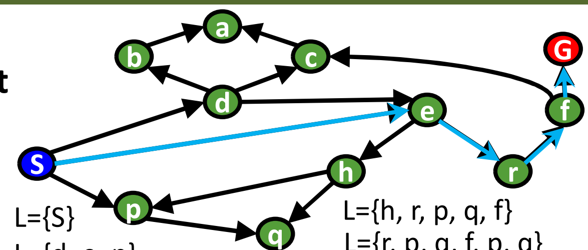




Dr. Monidipa Das, Department of CDS, IISER Kolkata

# Breadth-First Search

- **Strategy:** expand a **shallowest node first**
- **Implementation: Fringe is a FIFO queue**




L={S}
L={d, e, p}
L={e, p, b, c, e}
L={p, b, c, e, h, r}
L={b, c, e, h, r, q}
L={c, e, h, r, q, a}
L={e, h, r, q, a, a}
L={h, r, q, a, a, h, r}
L={r, q, a, a, h, r, p, q}
L={q, a, a, h, r, p, q, f}
L={a, a, h, r, p, q, f}
L={a, h, r, p, q, f}
L={h, r, p, q, f}
L={r, p, q, f, p, q}
L={p, q, f, p, q, f}
L={q, f, p, q, f, q}
L={f, p, q, f, q}
L={p, q, f, q, c, G}
L={q, f, q, c, G, q}
L={f, q, c, G, q}
L={q, c, G, q, c, G}
L={c, G, q, c, G}
L={G, q, c, G, a}

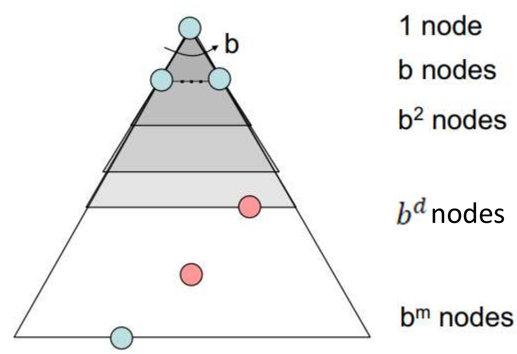
**G (goal state) found!**

**Path:** **S→ e→ r→ f→G**



# Breadth-First Search (BFS) Properties

- **How many nodes does BFS expand?**
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be d
  - Search takes time $O(b^d)$
- **How much space does the fringe take?**
  $O(b^d)$
- **Is it complete?**
  - d must be finite if a solution exists, so yes!
- **Is it optimal?**
  - Only if costs are all 1

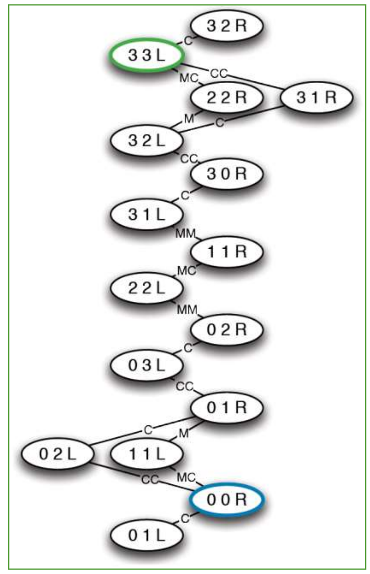


# DFS vs. BFS

21

- **When will BFS outperform DFS?**
  - **Branching factor is less**
  - **Goal state is closer to the initial state**
  - **State-space (though finite) contains loop**

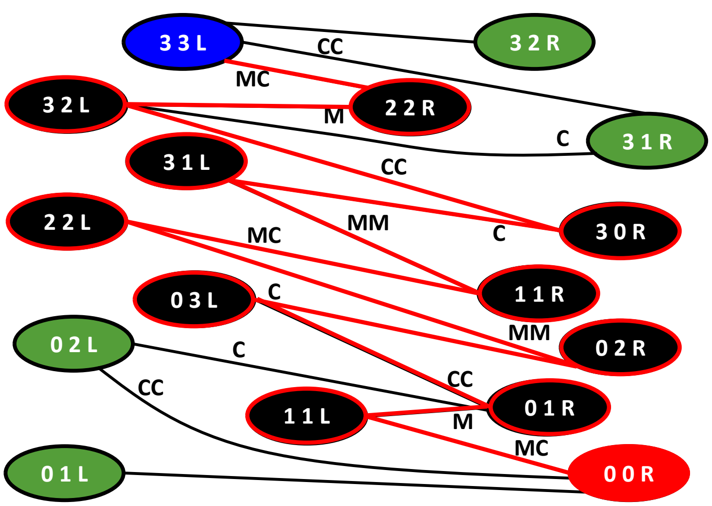
- **When will DFS outperform BFS?**
  - **Branching factor is very large**
  - **Goal state is far from the initial state**
  - **State-space is finite and contains no loop**






# Missionaries and Cannibals Problem Solution: DFS [scenario-1]

22



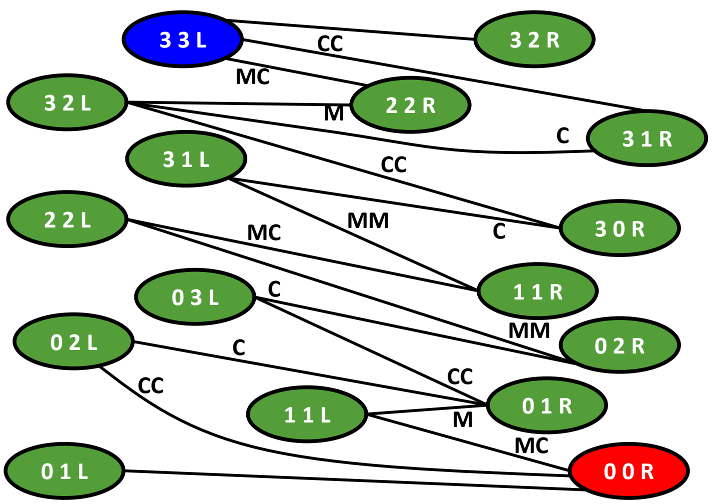

**Soln: MC M CC C MM MC MM C CC M MC**

L={(~~33L~~)}
L={(~~22R~~),(31R),(32R)}
L={(~~32L~~), (33L), (31R),(32R)}
L={(~~30R~~),(31R),(22R),(33L),(31R),(32R)}
L={(~~31L~~),(32L),(31R),(22R),(33L),(31R),(32R)}
L={(~~11R~~),(30R),(32L),(31R),(22R),(33L),(31R),(32R)}
L={(~~22L~~),(31L),(30R),(32L),(31R),(22R),(33L),(31R),(32R)}
L={(~~02R~~),(11R),(31L),(30R),(32L),(31R),(22R),(33L),(31R),(32R)}
L={(~~03L~~),(22L),(11R),(31L),(30R),(32L),(31R),(22R),(33L), (31R),(32R)}
L={(~~01R~~),(02R),(22L),(11R),(31L),(30R),(32L),(31R),(22R), (33L), (31R),(32R)}
L={(~~11L~~)(02L),(03L),(02R),(22L),(11R),(31L),(30R),(32L),(31R),(22R), (33L), (31R),(32R)}
L={(~~00R~~),(01R),(02L),(03L),(02R),(22L),(11R),(31L),(30R),(32L),(31R),(22R), (33L), (31R),(32R)}

**Goal State!**

# Missionaries and Cannibals Problem Solution: DFS [scenario-2]

23



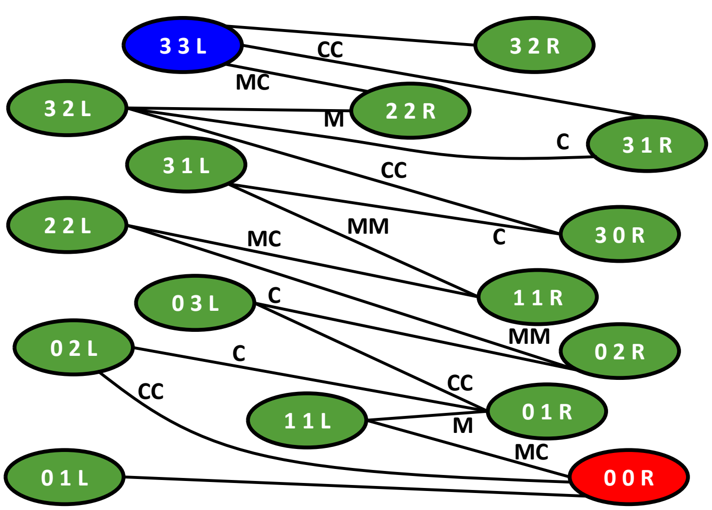

L={(~~33L~~)}
L={(~~32R~~),(31R),(22R)}
L={(~~33L~~),(31R),(22R)}
L={(~~32R~~),(31R),(22R),(31R),(22R)}
L={(~~33L~~),(31R),(22R),(31R),(22R)}
L={(~~32R~~),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~33L~~),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~32R~~),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~33L~~),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~32R~~),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~33L~~),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R)}
L={(~~32R~~),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R),(31R),(22R), (31R),(22R)}
. . . . . . .
. . . . . . .

**Infinite times…….. NO solution found**



# Missionaries and Cannibals Problem Solution: BFS

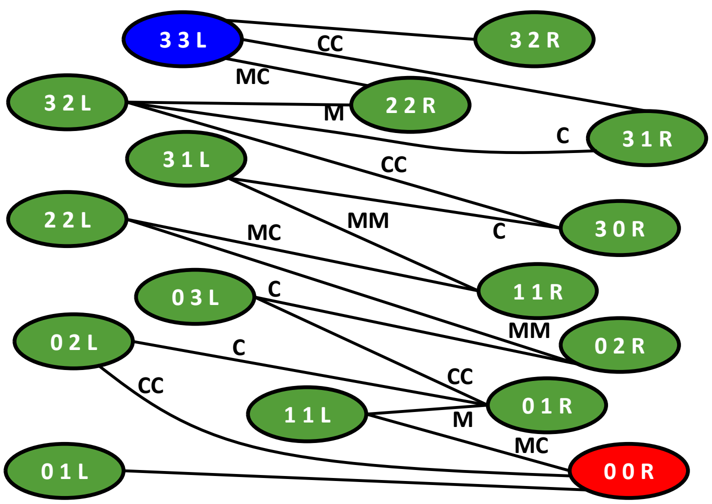
24




L={(~~33L~~)}
L={(~~22R~~),(31R),(32R)}
L={(~~31R~~),(32R), (32L), (33L)}
L={(~~32R~~), (32L), (33L),(32L),(33L)}
L={(~~32L~~), (33L),(32L),(33L),(33L)}
L={(~~33L~~),(32L),(33L),(33L),(30R),(22R),(31R)}
L={(~~32L~~),(33L),(33L),(30R),(22R),(31R),(22R), (31R),(32R)}
L={(~~33L~~),(33L),(30R),(22R),(31R),(22R),(31R),(32R),(30R),(22R),(31R)}
L={(~~33L~~),(30R),(22R),(31R),(22R),(31R),(32R),(30R),(22R),(31R),(22R),(31R),(32R)}
L={(~~30R~~),(22R),(31R),(22R),(31R),(32R),(30R),(22R),(31R),(22R),(31R),(32R), (22R),(31R),(32R)}
L={(~~22R~~),(31R),(22R),(31R),(32R),(30R),(22R), (31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L)}
L={(~~31R~~),(22R),(31R),(32R),(30R),(22R),(31R),(22R),(31R),(32R), (22R),(31R),(32R),(31L),(32L),(32L),(33L)}

# Missionaries and Cannibals Problem Solution: BFS

25




L={~~(22R)~~,(31R),(32R),(30R),(22R),(31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L),(32L),(33L),(32L),(33L)}

L={~~(31R)~~,(32R),(30R),(22R),(31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L),(32L),(33L),(32L),(33L),(32L),(33L)}

L={~~(32R)~~,(30R),(22R),(31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L),(32L),(33L),(32L),(33L),(32L),(33L),(32L),(33L)}

L={~~(30R)~~,(22R),(31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L),(32L),(33L),(32L),(33L),(32L),(33L),(32L),(33L),(33L)}

L={~~(22R)~~,(31R),(22R),(31R),(32R),(22R),(31R),(32R),(31L),(32L),(32L),(33L),(32L),(33L),(32L),(33L),(32L),(33L),(33L),(31L),(32L)}

. . . . . . .

. . . . . . .

. . . . . . .

**Huge computational time and space!**



# How to Avoid Exploring Redundant Path?

26

- The way to avoid exploring redundant paths is to **remember where one has been.**
- To do this, TREE-SEARCH algorithm is augmented with a data structure called the ***explored set*** (also known as the ***closed list***), which ***remembers every expanded node.***
- Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier—can be discarded instead of being added to the frontier.
- The new algorithm, called **GRAPH-SEARCH**

# Graph Search

27

**function** GRAPH-SEARCH(*problem*) **returns** a solution, or failure
initialize the frontier using the initial state of *problem*
***initialize the explored set to be empty***
**loop do**
**if** the frontier is empty **then return** failure
choose a leaf node and remove it from the frontier
**if** the node contains a goal state **then return** the corresponding solution
***add the node to the explored set***
expand the chosen node, adding the resulting nodes to the frontier
***only if not in the frontier or explored set***

✓GRAPH-SEARCH algorithm contains at most one copy of each state

✓the frontier separates the state-space graph into the explored region and the unexplored region, so that every path from the initial state to an unexplored state has to pass through a state in the frontier.



28

# Questions?