PROBLEM SOLVING

2/7/2018

Module-2

Introduction

- It is not that I am so smart, its just that I stay with problems longer –Albert Einstein
- In solving problems, we sometimes have to search through many possible ways of doing something.
- We may know all the possible actions our robot can do, but we have to consider various sequences to find a sequence of actions to achieve a goal.
- We may know all the possible moves in a chess game, but we must consider many possibilities to find a good move.
- Many problems can be formalized in a general way as search problems.

Looking for Parking

- Going home; need to find street parking
- **Formulate Goal:**

  Car is parked
- **Formulate Problem:**

  States: street with parking and car at that street

  Actions: drive between street segments
- **Find solution:**

  Sequence of street segments, ending with a street with parking

Problem Spaces and Search

- Building a system to solve a problem requires the following steps

1. -Define the problem precisely including detailed specifications and what constitutes an acceptable solution;
2. - Analyses the problem thoroughly for some features may have a dominant affect on the chosen method of solution;
3. - Isolate and represent the background knowledge needed in the solution of the problem;
4. -Choose the best problem solving techniques in the solution

Defining the Problem as state Search

- The problem solving procedure applies an operator to a state to get the next state.
- Then it applies another operator to the resulting state to derive a new state.
- The process of applying an operator to a state and its subsequent transition to the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as state space approach.

Defining the Problem as state Search

- Problems dealt with in artificial intelligence generally use a common term called 'state'.
- A state represents a status of the solution at a given step of the problem solving procedure.
- The solution of a problem, thus, is a collection of the problem states.

Defining the Problem as state Search

- The state space representation forms the basis of most of the AI methods.
- Its structure corresponds to the structure of problem solving in two important ways:
  - It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
  - It permits us to define the process of solving a particular problem as a combination of known techniques and search, the general technique of exploring the space to try to find some path from current state to a goal.

Example: Playing Chess

- To build a program that could “play chess”, we could first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side or the other.

- In addition, we must make explicit the previously implicit goal of not only playing the legal game of chess but also winning the game, if possible.

Playing Chess

- The starting position can be described as an 8 by 8 array where each position contains a symbol for appropriate piece.
- We can define as our goal the check mate position.
- The legal moves provide the way of getting from initial state to a goal state.
- They can be described easily as a set of rules consisting of two parts:
  - A left side that serves as a pattern to be matched against the current board position.
  - And a right side that describes the change to be made to reflect the move

Playing chess

- However, this approach leads to large number of rules board positions !!
- Using so many rules poses problems such as:
  - No person could ever supply a complete set of such rules.
  - No program could easily handle all those rules. Just storing so many rules poses serious difficulties.

Production Systems

Systems that generate (produce) rules (states) to reach a solution.

1. A *set of rules of the form* $C_i \quad A_i$ *where* C_i *refers to starting state and* A_i *represents consequent state. Also* C_i *the condition part and* A_i *is the action* part.

2. One or more *knowledge databases that contain whatever information is relevant* for the given problem.

Production Systems

1. A *control strategy that ascertains the order in which the rules must be applied to the available d*atabase

2. A *rule applier which is the computational system that implements the control* strategy and applies the rules to reach to goal (if it is possible).

State Space Search

- The water jug problem: You are given two jugs, a 4-litre one and a 3-litre one. Neither has any measuring markers on it.

- There is a pump that can be used to fill the jugs with water. How can you get exactly 2 litres of water into 4-litre jug.

State Space Search

- Let x and y be the amounts of water in 4-Lt and 3-Lt Jugs respectively
- Then (x,y) refers to water available at any time in 4-Lt and 3-Lt jugs.
- Also (x,y) (x-d,y+dd) means drop some unknown amount d of water from 4-Lt jug and add dd onto 3-Lt jug.

All Possible Production Rules

1. (x, y) → (4, y) if x<4, fill it to 4; y remains unchanged
   if $x < 4$
2. (x, y) → (x, 3) if y<3, fill it to 3; x remains unchanged
   if $y < 3$
3. (x, y) → (x – d, y) if there is some water in 4 Lt jug, drop some more water from it
   if $x > 0$
4. (x, y) → (x, y – d) if there is some water in 3 Lt jug, drop some more water from it
   if $y > 0$

All Possible Production Rules

5. (x, y) → (0, y) if there is some water in 4-Lt, empty it, y remains unchanged
   if x > 0
6. (x, y)→ (x, 0) if there is some water in 3-Lt, empty it, x remains unchanged
   if y > 0
7. (x, y) → (4, y – (4 – x)) if there is some water in 3-Lt, the sum of
   if x + y ≥ 4, y > 0 water of 4-Lt and 3-Lt jug is >=4, then fill
   water in 4-Lt jug to its capacity from 3-Lt jug
8. (x, y) → (x – (3 – y), 3) same as 7 with suitable change in x,y
   if x + y ≥ 3, x > 0
9. (x, y) → (x + y, 0) if sum of water in both jugs <=4, then drop
   if x + y ≤ 4, y > 0 whole water from 3-Lt into 4-Lt
10. (x, y) → (0, x + y) if sum of water in both jugs <=3, then drop
    if x + y ≤ 3, x > 0 whole water from 4-Lt into 3-Lt
11. (0, 2) → (2, 0) Transfer 2-Lt from 3-Lt jug into empty 4-Lt jug
12. (2, y) → (0, y) Empty 2 Lt water onto ground from 4-Lt jug
    without disturbing 3 Lt jug

State Space Search

Solution of Water Jug Problem

Obviously to solve water jug problem, we can perform following sequence of actions,

(0,0) (0,3) (3,0) (3,3) (4,2) (0,2) (2,0)

By applying rules 2,9,2,7,5 and 9 with initial empty jugs

Remember: There is NO hard and fast rules to follow this sequence. In any state space search problem, there can be numerous ways to solve, your approach can be different to solve a problem and sequence of actions too.

Other problems

- Only one disk must be moved at a time.
- Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
- No disk may be placed on top of a smaller disk.

- 3. Monkey Banana Problem: A monkey is in a room. A bunch of bananas is hanging from the ceiling and is beyond the monkey's reach. However, in the room there are also a chair and a stick. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air. What is the best sequence of actions for the monkey?

Other problems

- 1. Cannibals and missionaries problems: In the missionaries (humans)and cannibals (human eaters) problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people. At any time number of cannibals on either side should not be greater than number of missionaries otherwise former will eat latter. Also The boat cannot cross the river by itself with no people on board.
- 2. Tower of Hanoi Problem: It consists of three pegs, and a number of disks (usually 60) of different sizes which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

7- PROBLEM CHARACTERISTICS

1. Is the problem decomposable into set of sub problems?
2. Can the solution step be ignored or undone?
3. Is the problem universally predictable?
4. Is a good solution to the problem obvious without comparison to all the possible solutions?
5. Is the desire solution a state of world or a path to a state?
6. Is a large amount of knowledge absolutely required to solve the problem?
7. Will the solution of the problem required interaction between the computer and the person?

Search Techniques

- **Un-informed (Blind)** Search Techniques do not take into account the location of the goal. Intuitively, these algorithms ignore where they are going until they find a goal and report success.

- Uninformed search methods use only information available in the problem definition and past explorations, e.g. cost of the path generated so far. Examples are
  - – **Breadth-first search (BFS)**
  - – **Depth-first search (DFS)**
  - – **Iterative deepening (IDA)**
  - – **Bi-directional search**

Un-informed (Blind) Search Technique

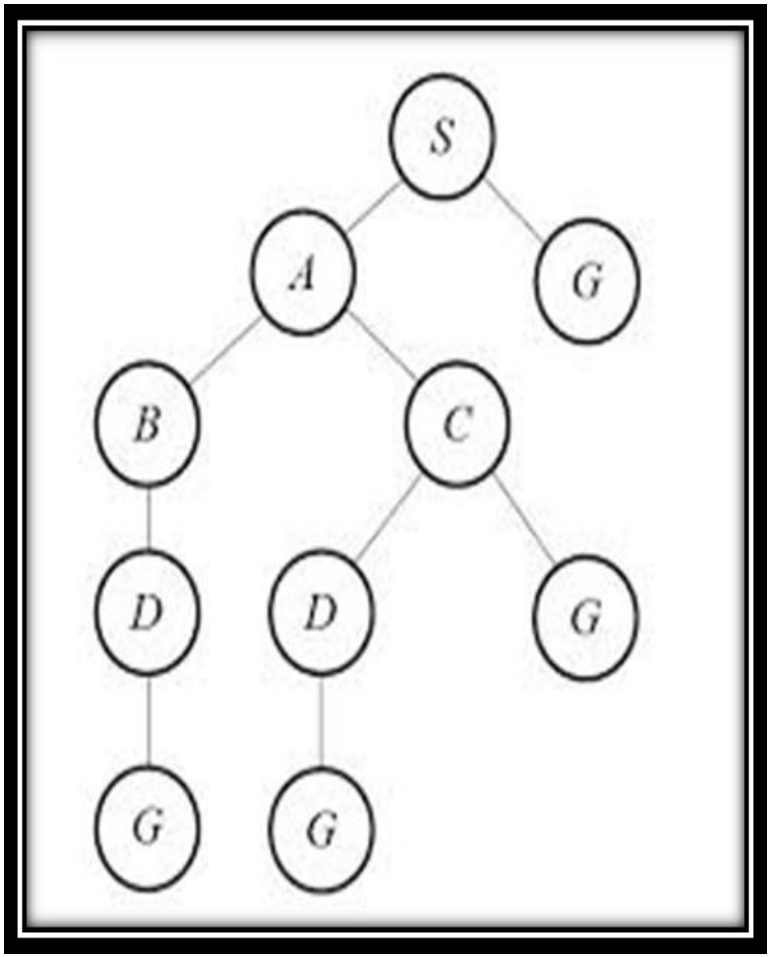
Search Techniques

Issues In Design Of Search Program

- The problem-solving power of search techniques is limited in part because of their generality
- It is generally understood (in symbolic AI) that solving complex problems depends on knowledge and mechanisms to manipulate it
- The challenge is known as the (knowledge) representation problem
- The discussion on knowledge level and symbol level, and "representation mappings", is all about the relation between symbols (syntax) and meaning.

Search Techniques: DFS

- **Depth-first search (DFS):** We can start with a node and explore with all possible solutions available with this node.
- **It is not complete, non-optimal, may stuck in infinite loop .**
- **DFS** starts at the root node and explores as far as possible along each branch before backtracking.



Search Techniques: DFS

1 . Create a stack with the root node and add its direct children

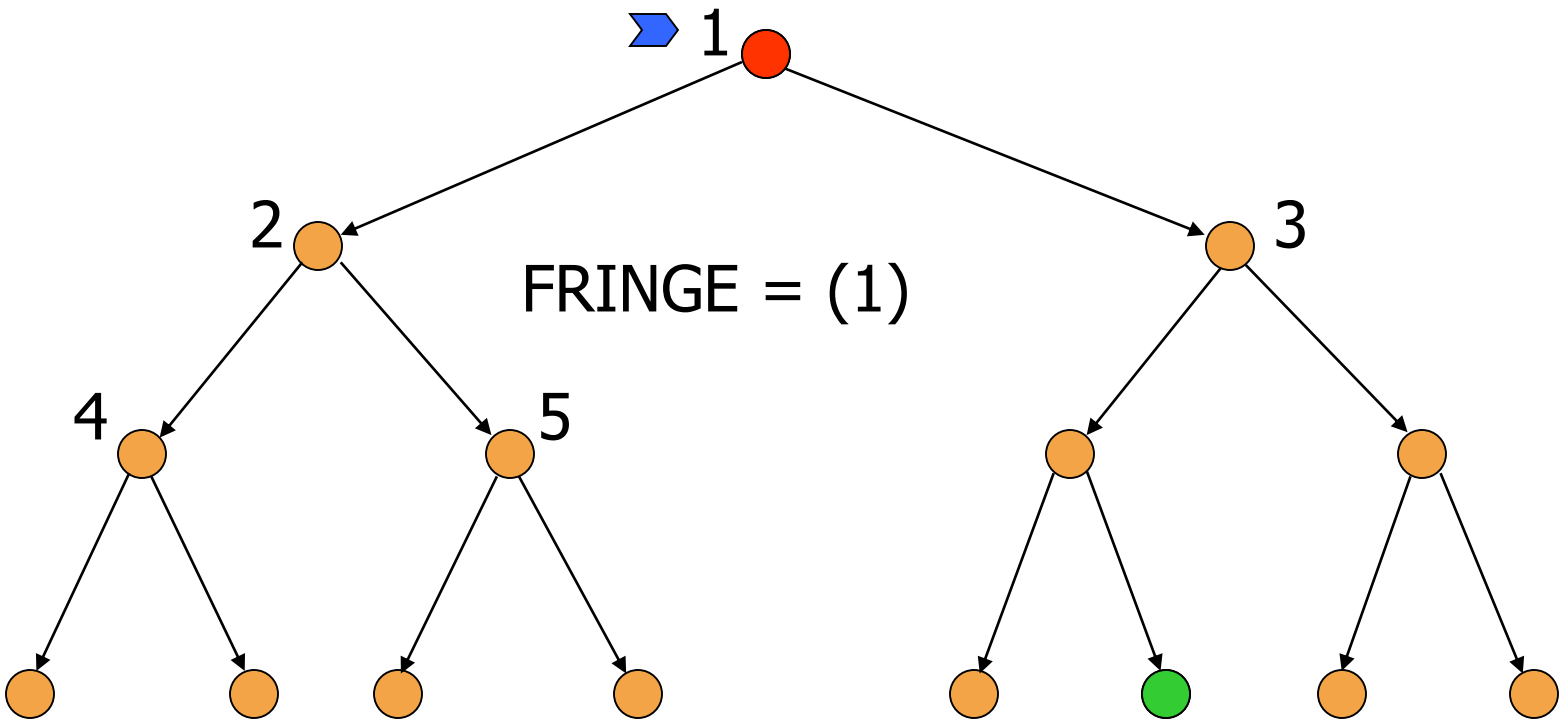
2. Remove a node in order from stack and examine it
   - ✓ If the element sought is found in this node, quit the search and return a result.
   - ✓ Otherwise insert any successors (the direct child nodes) that have not yet been discovered before existing nodes.

3. If the stack is empty, every node on the graph has been examined – quit the search and return "not found".

4. If the stack is not empty, repeat from Step 2.

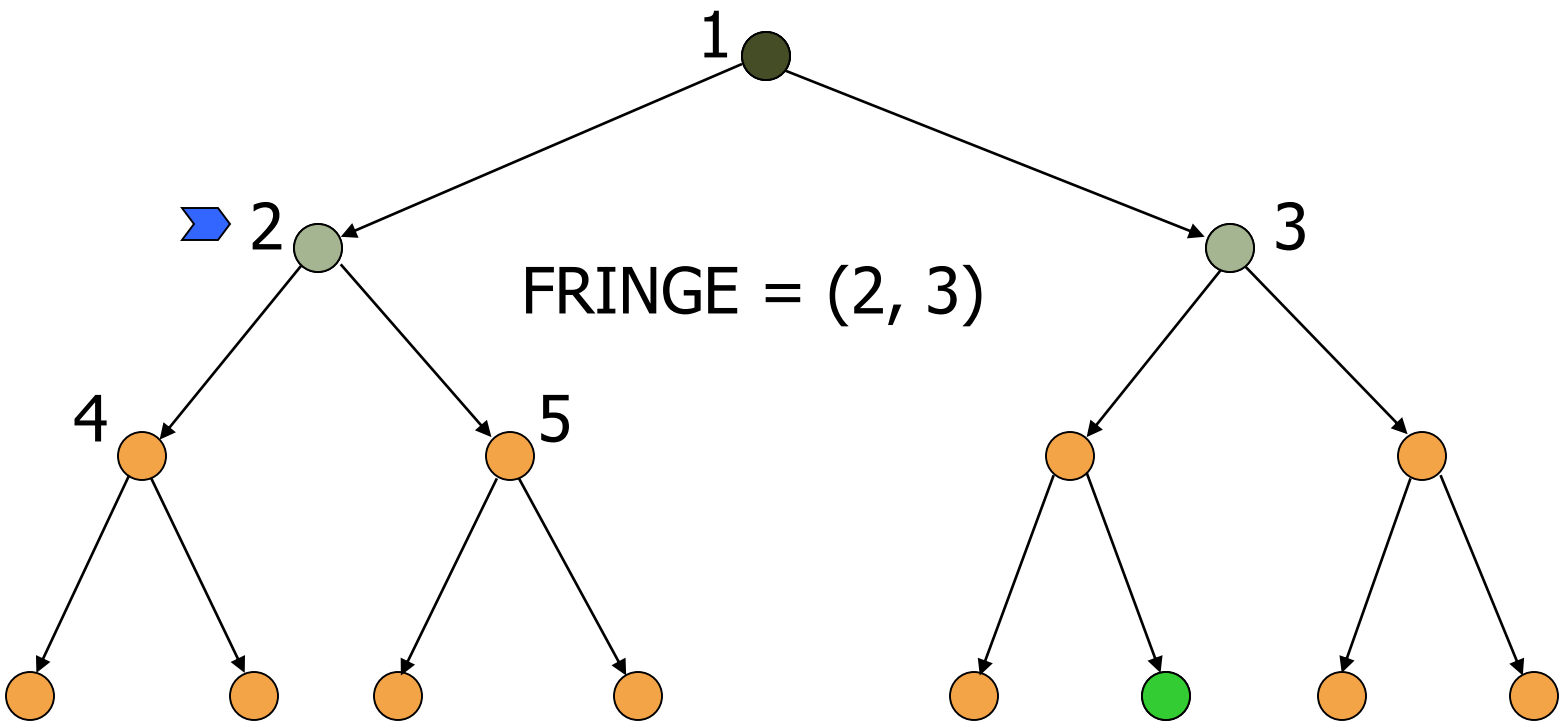
Depth-First Strategy

New nodes are inserted at the front of FRINGE



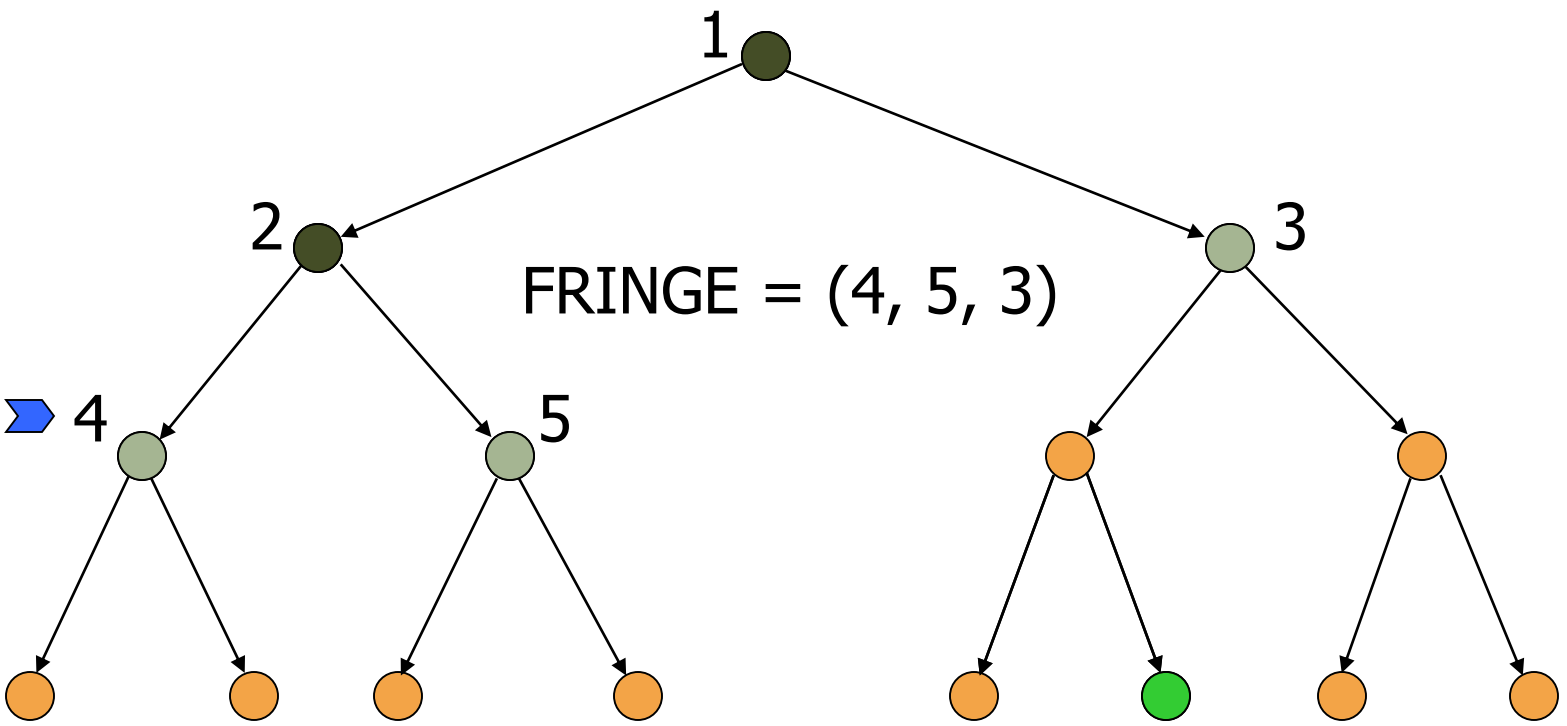
Depth-First Strategy

New nodes are inserted at the front of FRINGE



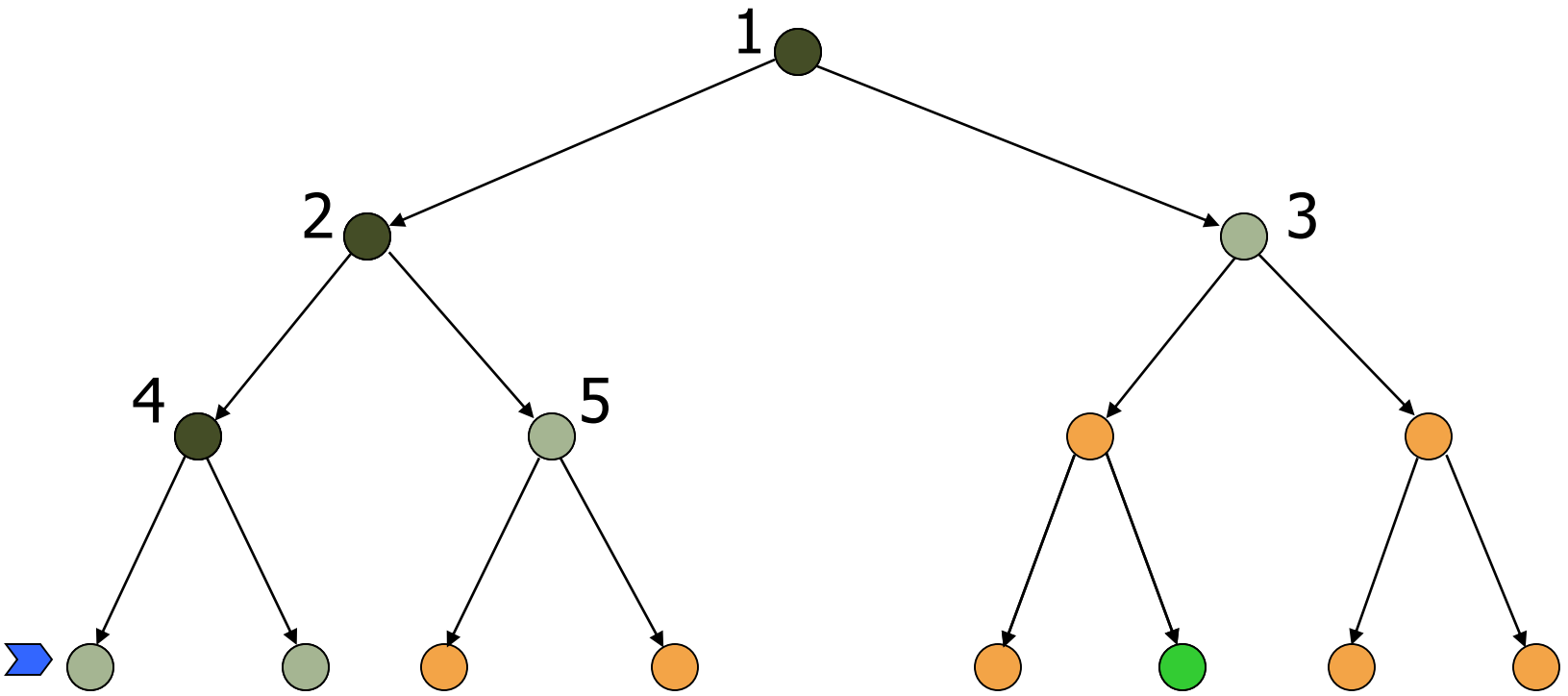
Depth-First Strategy

New nodes are inserted at the front of FRINGE



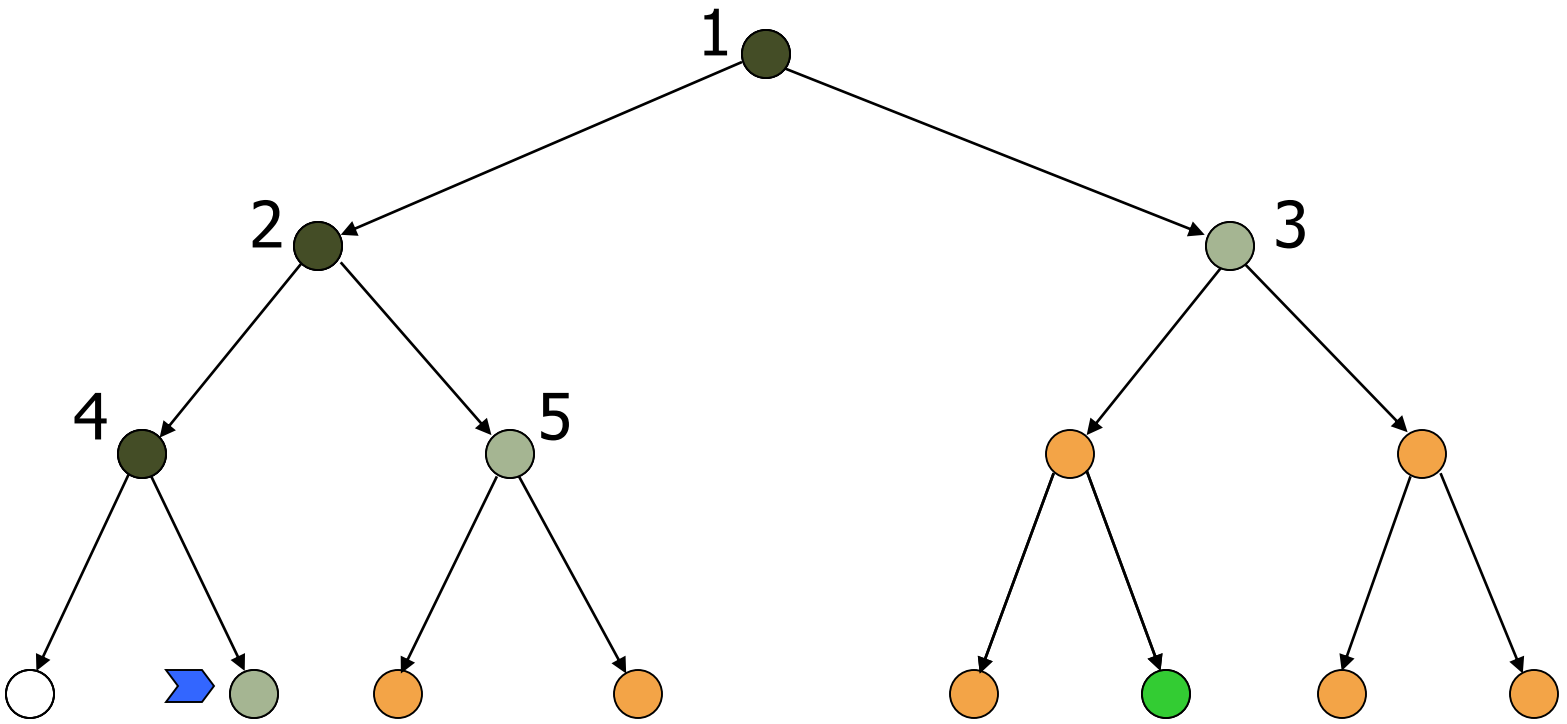
Depth-First Strategy

New nodes are inserted at the front of FRINGE



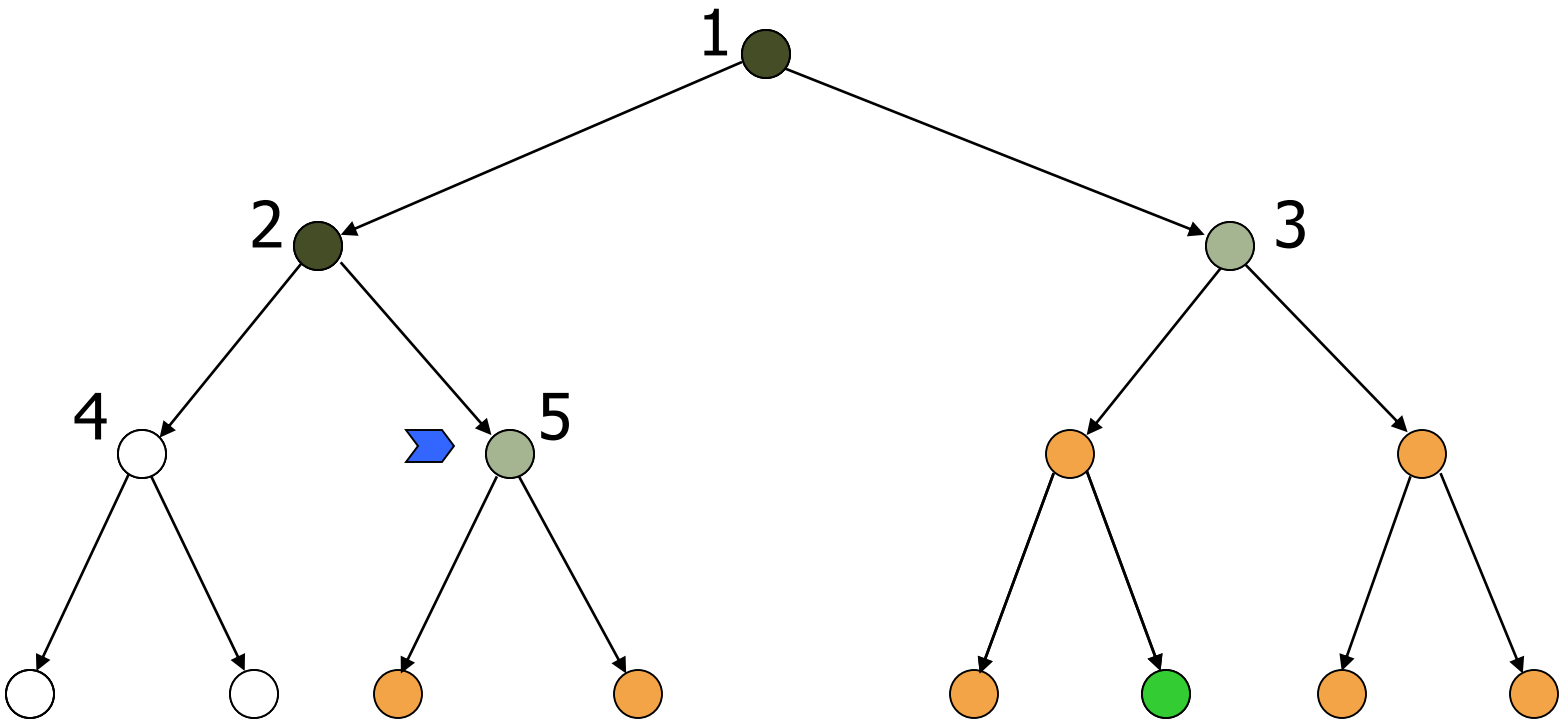
Depth-First Strategy

New nodes are inserted at the front of FRINGE



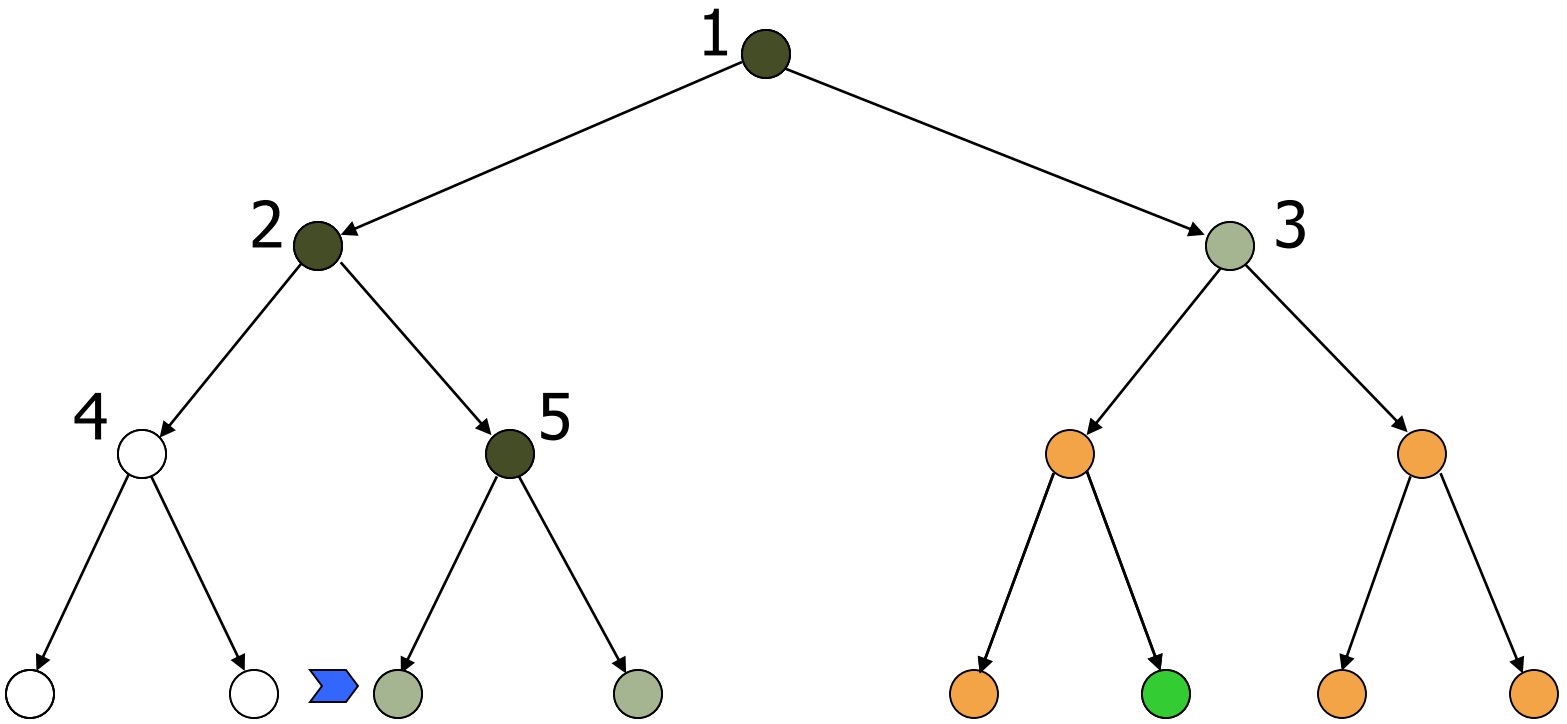
Depth-First Strategy

New nodes are inserted at the front of FRINGE



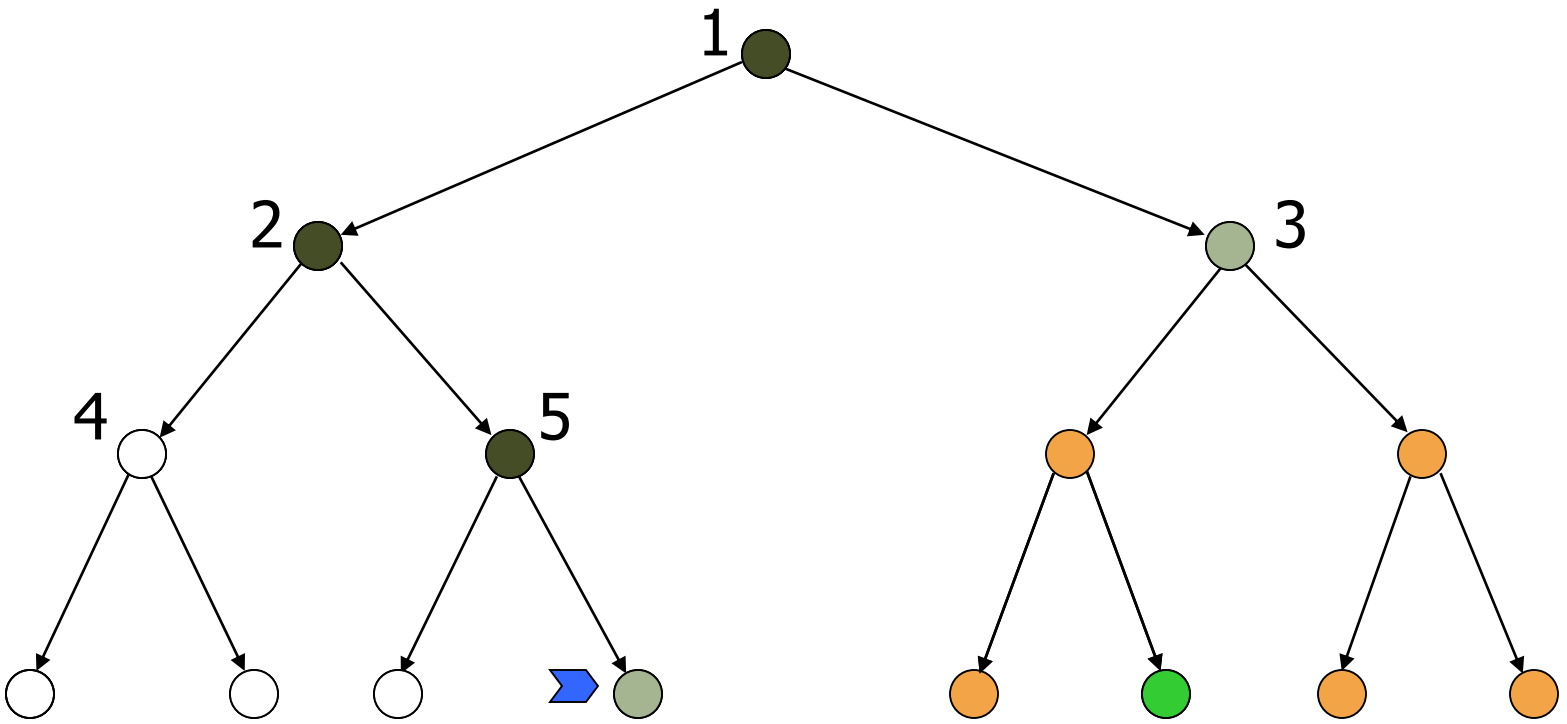
Depth-First Strategy

New nodes are inserted at the front of FRINGE



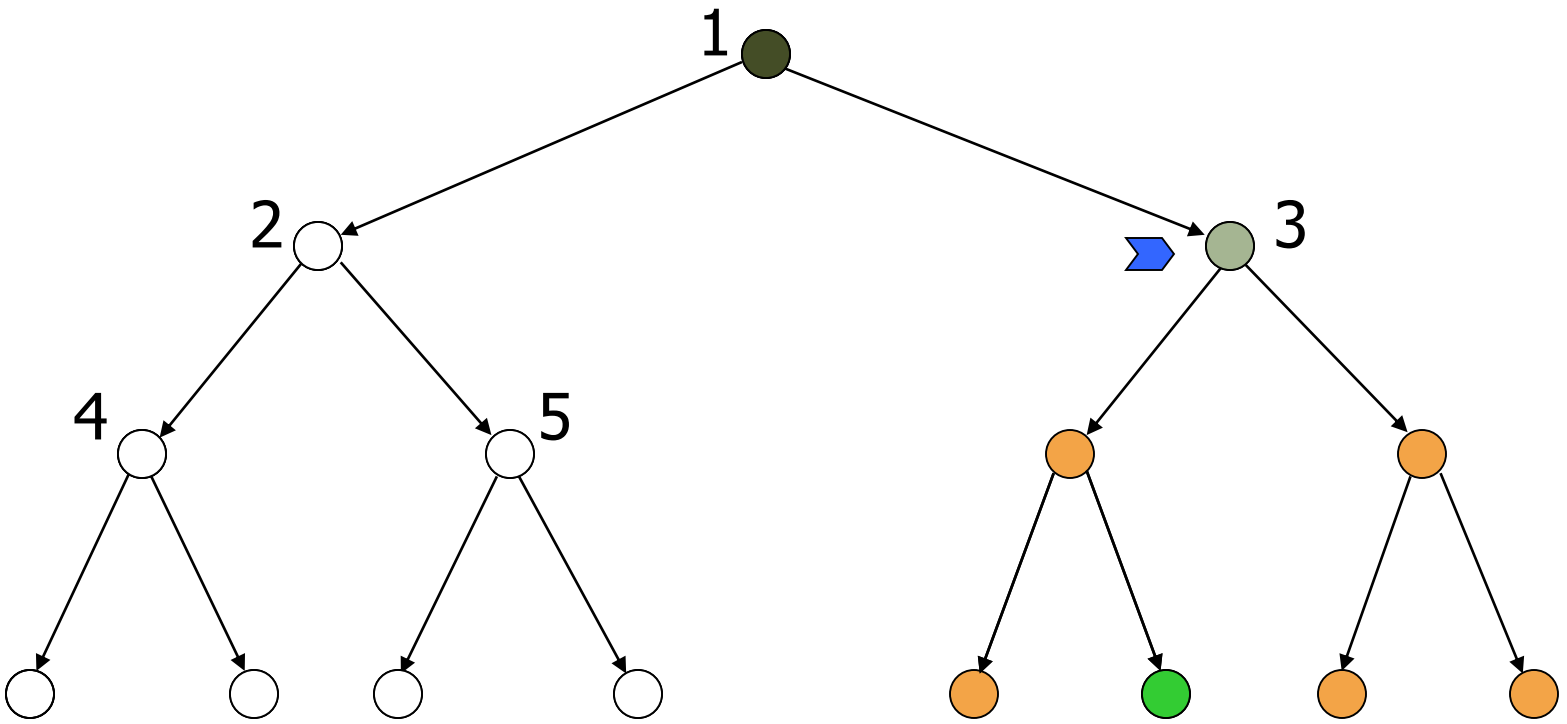
Depth-First Strategy

New nodes are inserted at the front of FRINGE



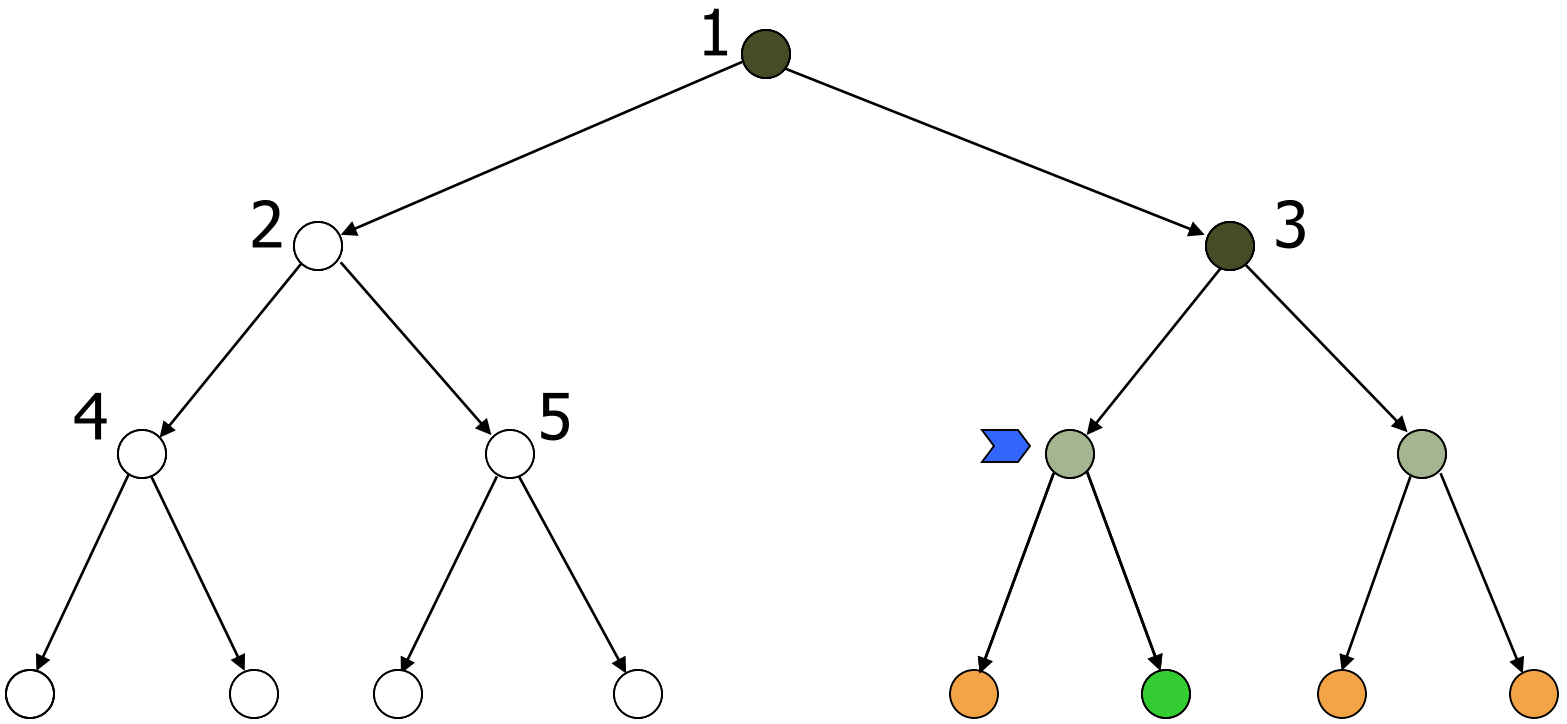
Depth-First Strategy

New nodes are inserted at the front of FRINGE



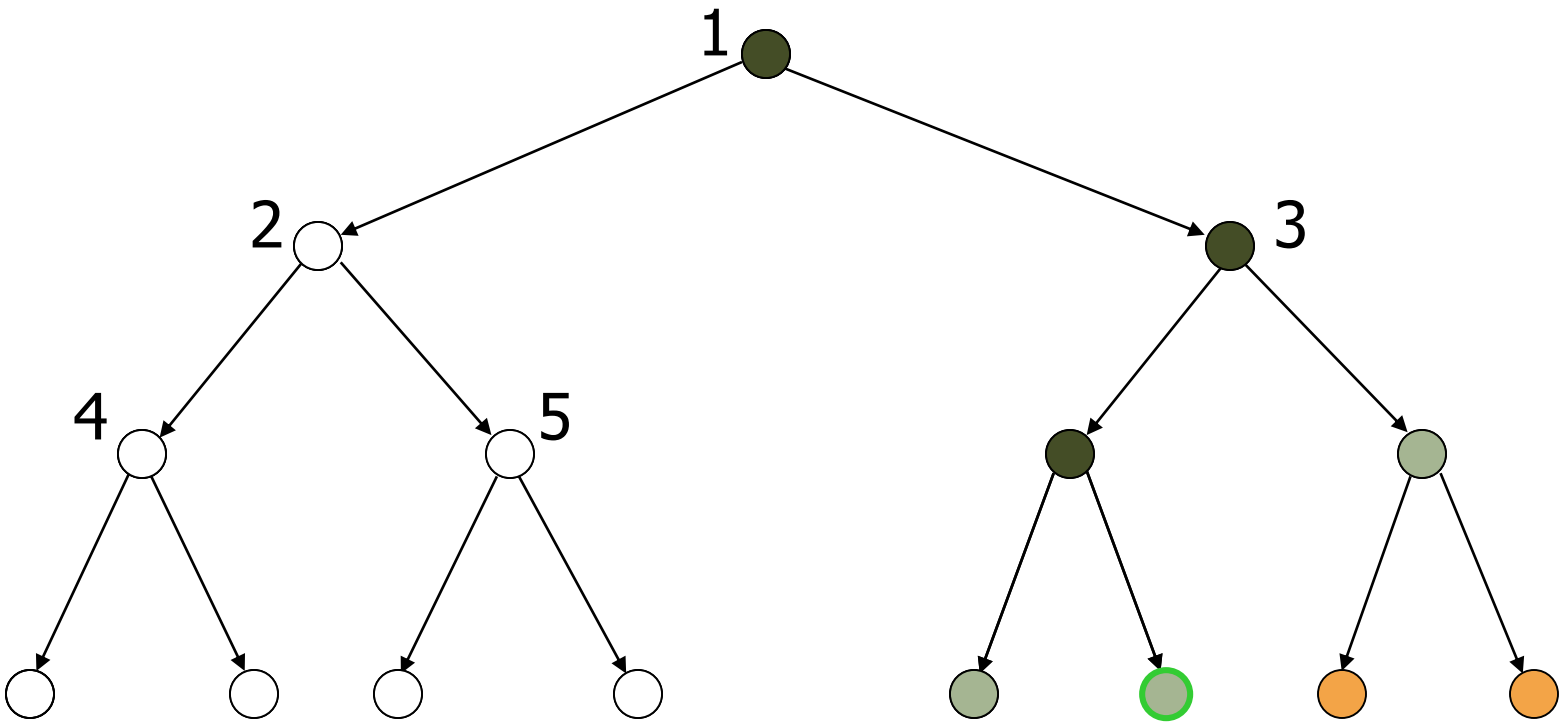
Depth-First Strategy

New nodes are inserted at the front of FRINGE



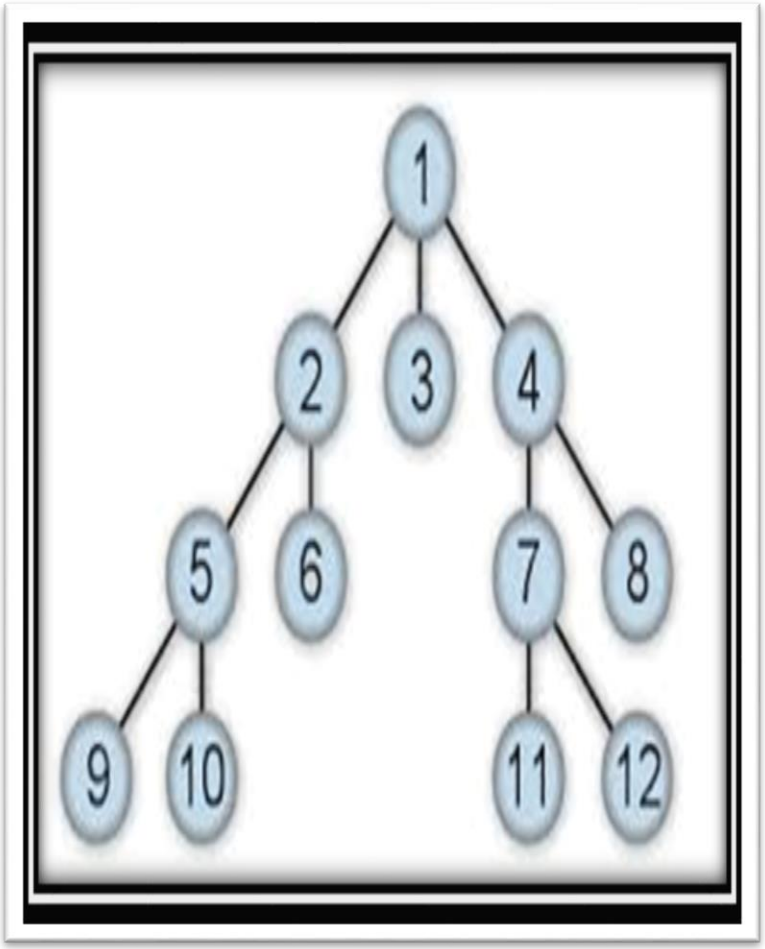
Depth-First Strategy

New nodes are inserted at the front of FRINGE



Search Techniques: BFS

- Breadth-first search (BFS): At each level, we expand all nodes (possible solutions), if there exists a solution then it will be found.
- It is complete, optimal, best when space is no problem as it takes much space



Search Techniques: BFS

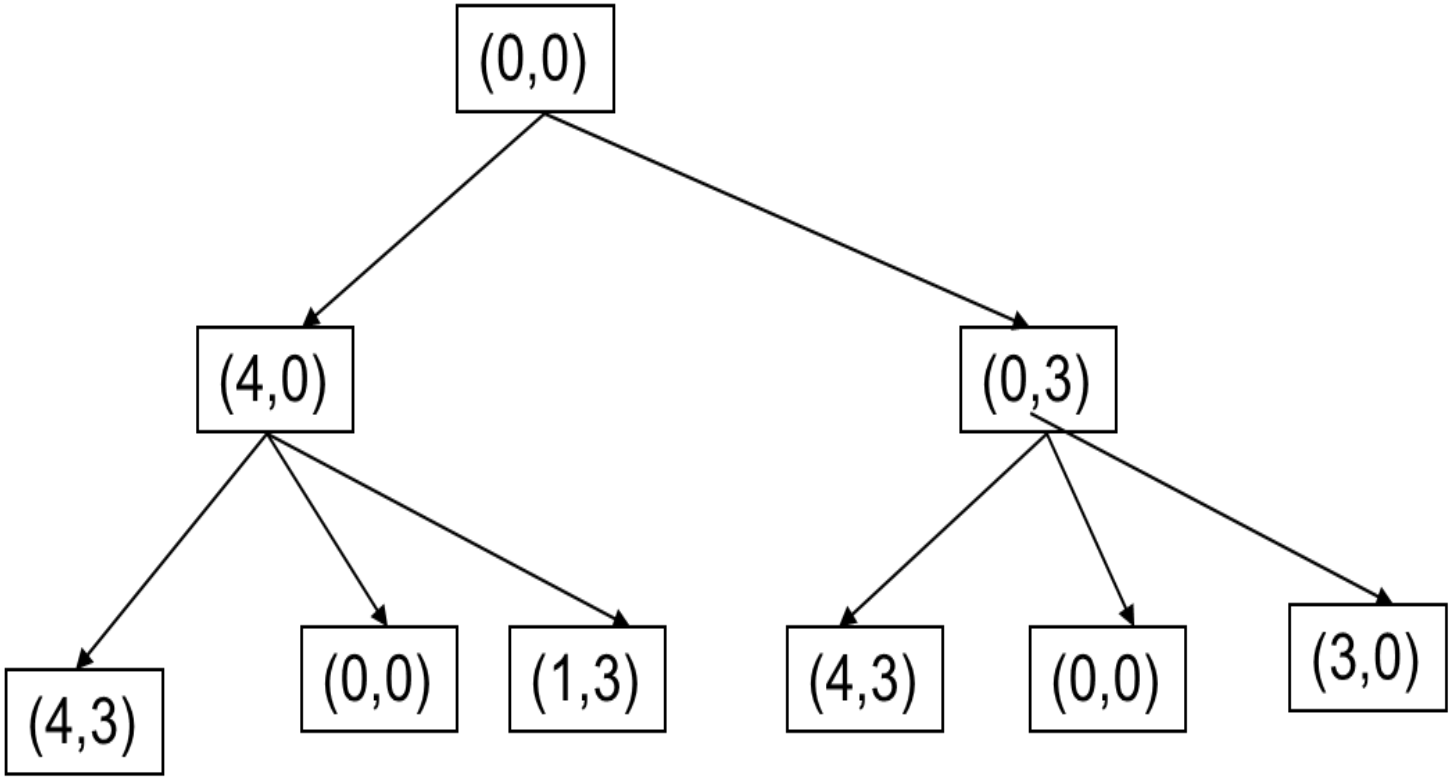
- Algorithm BFS
- The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

  1. Create a queue with the root node and add its direct children
  2. Remove a node in order from queue and examine it

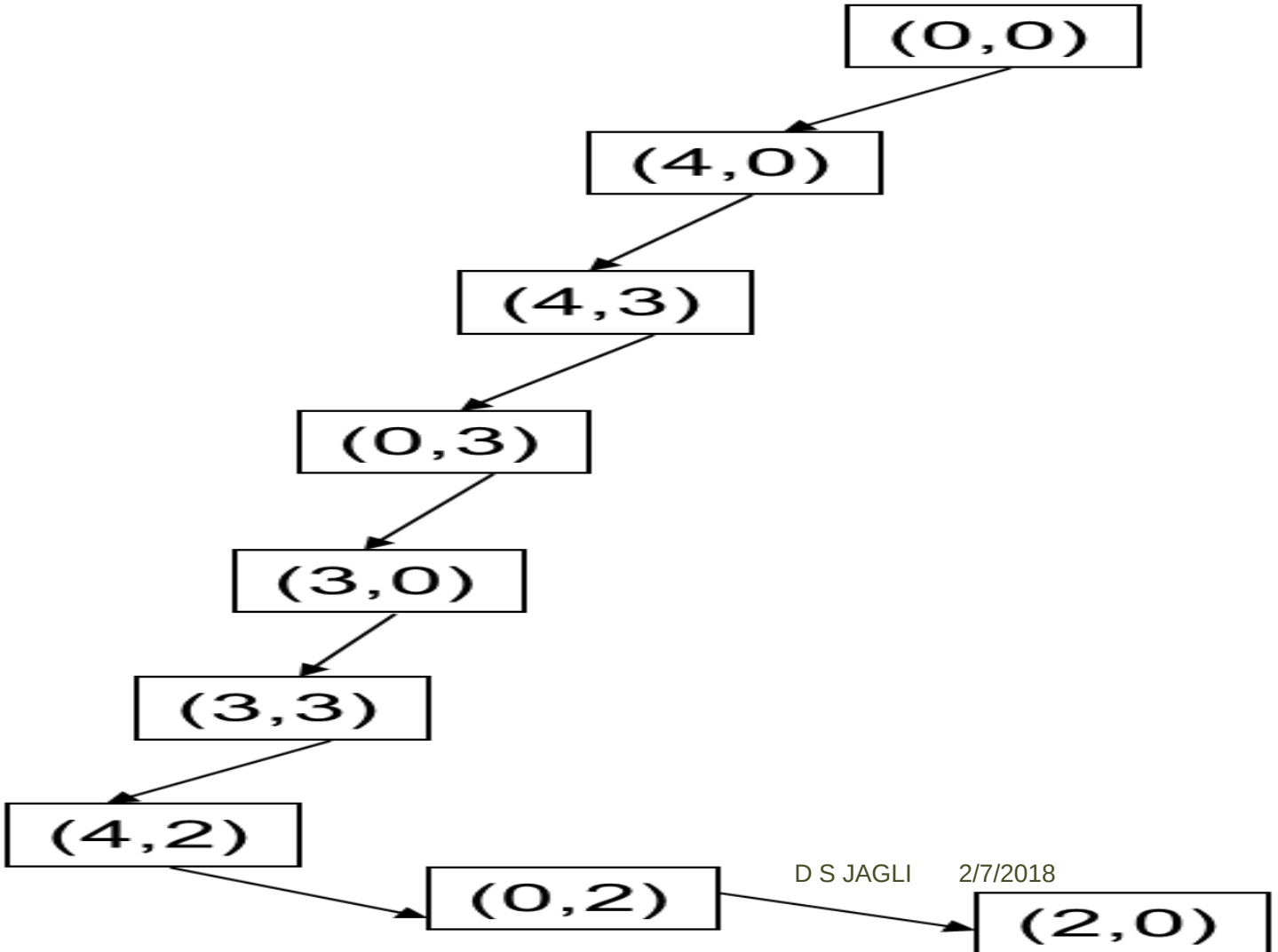
     If the element sought is found in this node, quit the search and return a result.

     Otherwise append any successors (the direct child nodes) that have not yet been discovered.
  3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
  4. If the queue is not empty, repeat from Step 2.

BFS For A Water Jug Problem



DFS For A Water Jug Problem



Search Techniques

- Informed Search Techniques A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more informed.
- It incorporates additional measure of a potential of a specific state to reach the goal. The potential of a state (node) to reach a goal is measured through a heuristic function. These are also called intelligent search

  - Best first search
  - Greedy Search
  - A* search

Local Search: Hill Climbing

- Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution.

- If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found.

- In **simple hill climbing**, the first closer node is chosen, whereas in **steepest ascent hill climbing** all successors are compared and the closest to the solution is chosen.

Local Search: Hill Climbing

- Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions.

- Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.
- **Stochastic hill climbing** does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

Local Search: Hill Climbing

Pseudo Algorithm

1. Pick initial state *s*
2. Pick *t in neighbors(s) with the largest f(t)*
3. IF *f(t) <= f(s) THEN stop, return s*
4. *s = t. GOTO 2.*

Features:

Not the most sophisticated algorithm in the world.

- Very greedy.
- Easily stuck.

Local Search: Hill Climbing

Simple Hill Climbing

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:

– Select and apply a new operator

– Evaluate the new state:

if goal then quit

otherwise better than current state <- - new current state

Drawbacks: it not try all possible new states!

???

THANK YOU