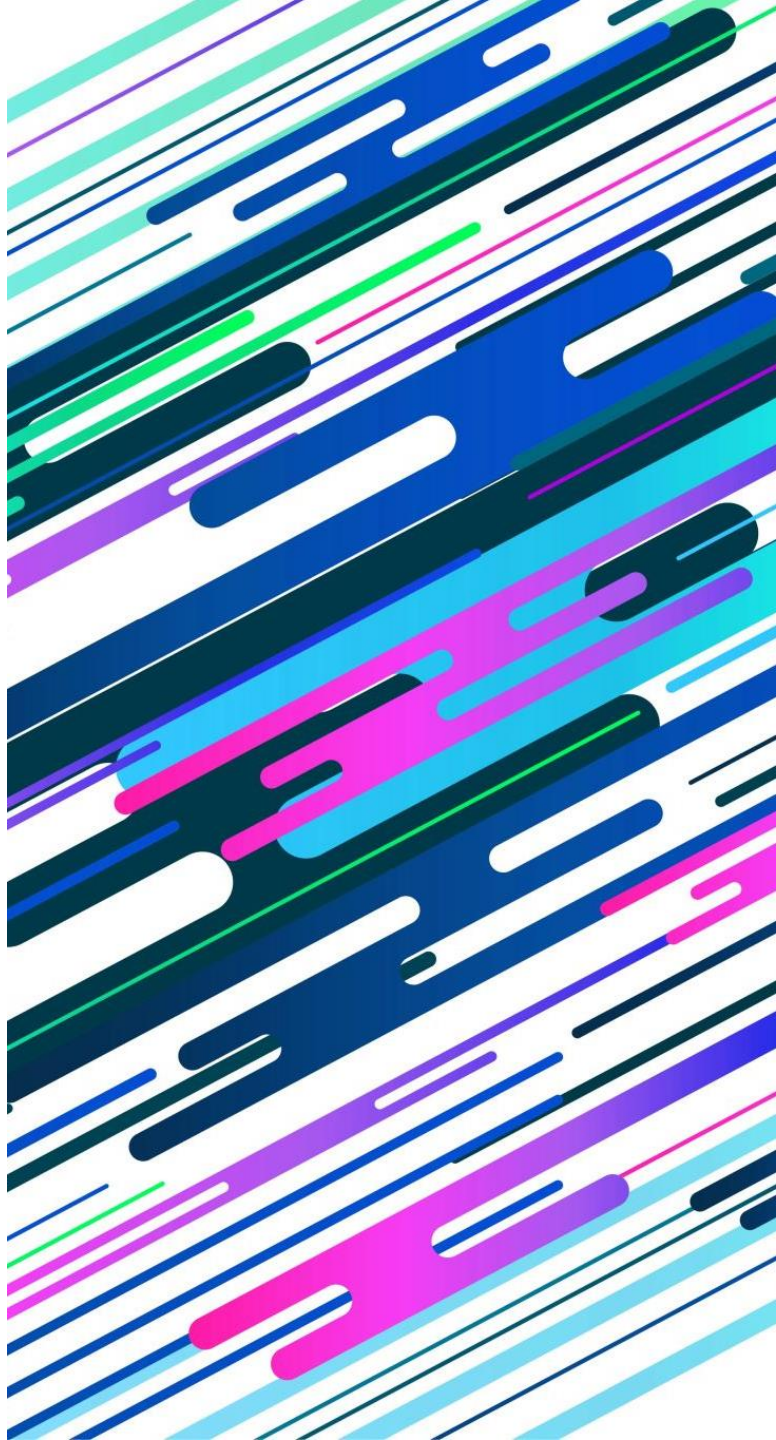
STATE-BASED MODELS

Thomas Tiam-Lee, PhD



Water Pouring Puzzle

GLASS A (8 ml) **GLASS B (5 ml)** **GLASS C (3 ml)**

- You have **3 glasses** A, B, C of different sizes.
- Glass A is filled with water.
- Your goal is to have **two glasses with exactly 4 ml water each**.
- You don't have any other materials aside from the glasses nor any additional water source.
- **What are the sequence of steps you need to do to accomplish this?**

Breaking Down the Problem

- **Goal:**

Breaking Down the Problem

- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.

Breaking Down the Problem

- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.
- **Starting Point:**

Breaking Down the Problem

- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.
- **Starting Point:** Glass A contains 8 ml of water. Glass B and Glass C are empty.

Breaking Down the Problem

- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.
- **Starting Point:** Glass A contains 8 ml of water. Glass B and Glass C are empty.
- **Possible Actions:**

Breaking Down the Problem

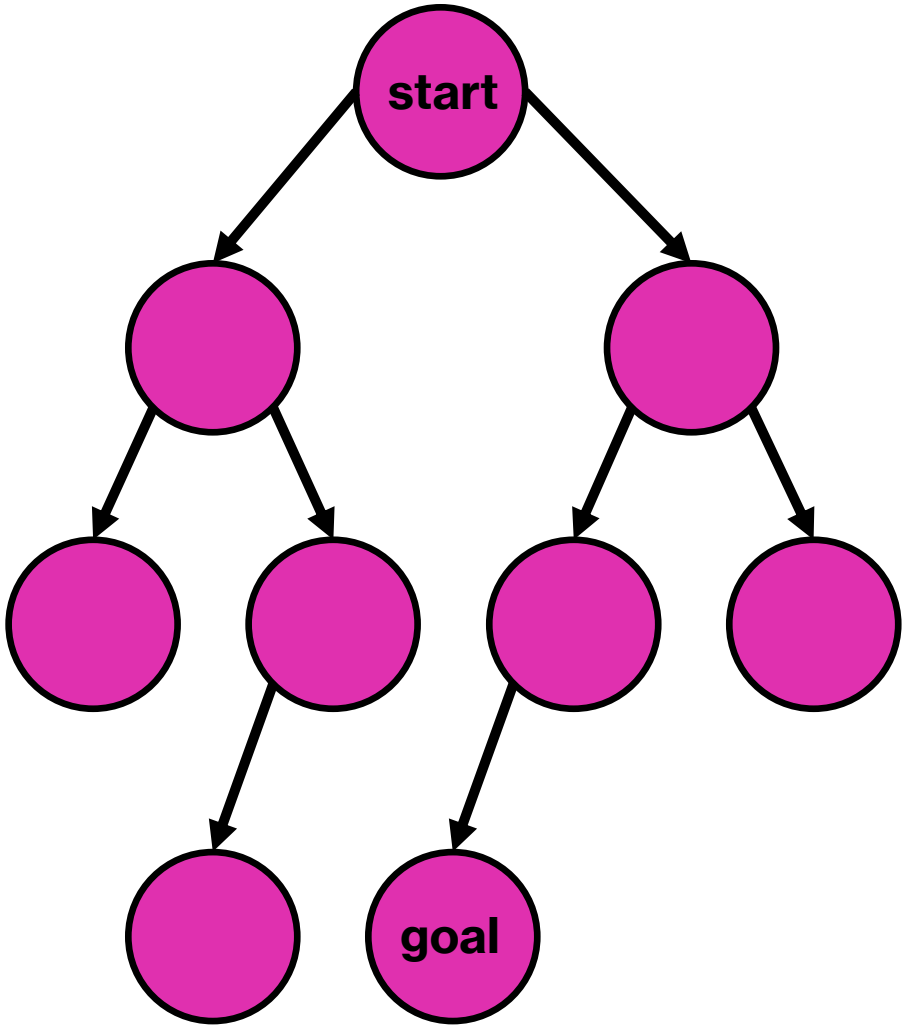
- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.
- **Starting Point:** Glass A contains 8 ml of water. Glass B and Glass C are empty.
- **Possible Actions:** Choose one glass and pour **all** of its contents to another glass.

Breaking Down the Problem

- **Goal:** Glass A and B both contain 4 ml of water. Consequently, Glass C should be empty.
- **Starting Point:** Glass A contains 8 ml of water. Glass B and Glass C are empty.
- **Possible Actions:** Choose one glass and pour **all** of its contents to another glass.
- **What We Want:** A sequence of actions that will move us from the starting point to the goal!

State-Based Models

- **Modeling:** represent a problem as a set of **states**, connected by **actions**. Doing an action causes you to move from one state to another.
- **Inference:** **search for a sequence of actions** starting from the **initial state to a goal state**.
  - In some cases, the sequence of actions must optimize some objective (e.g., safest, fastest).



Search Problem

- Main idea: systematically explore the state-space until a path from the start state to the goal state is found!

- **State space:** all the possible configurations a particular problem can be in

Search Problem Formulation

- **State:** A **single configuration** of the problem. The state should contain **only the relevant information** about the problem.
- **Action:** A valid move from one state that will result into transitioning to another state.
- **Cost:** (optional) the **penalty** associated with a particular action.

Search Problem Formulation

- **State:** A **single configuration** of the problem. The state should contain **only the relevant information** about the problem.
- **Action:** A valid move from one state that will result into transitioning to another state.
- **Cost:** the **penalty** associated with a particular action (can be set to be always 0 if no penalty).

Search Problem Formulation

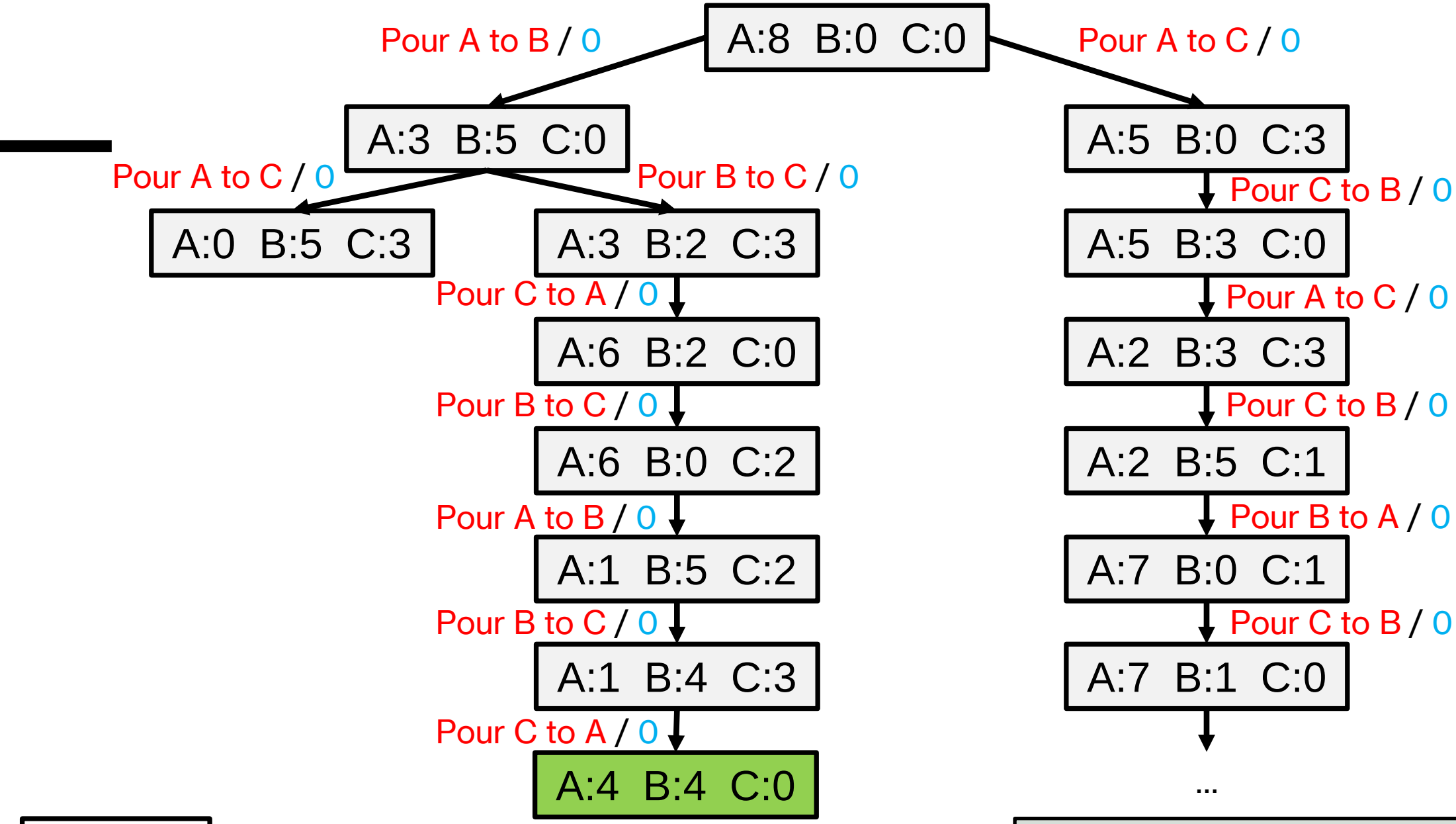
- s_{start}: the initial / start state
- $Actions(s)$: the set of possible actions from state s
- $Cost(s, a)$: the cost of performing action a from state s
- $Succ(s, a)$: the next state after performing action a from state s
- $IsEnd(s)$: true if s is a goal state or false otherwise

Modeling the Water Pouring Puzzle As a Search Problem

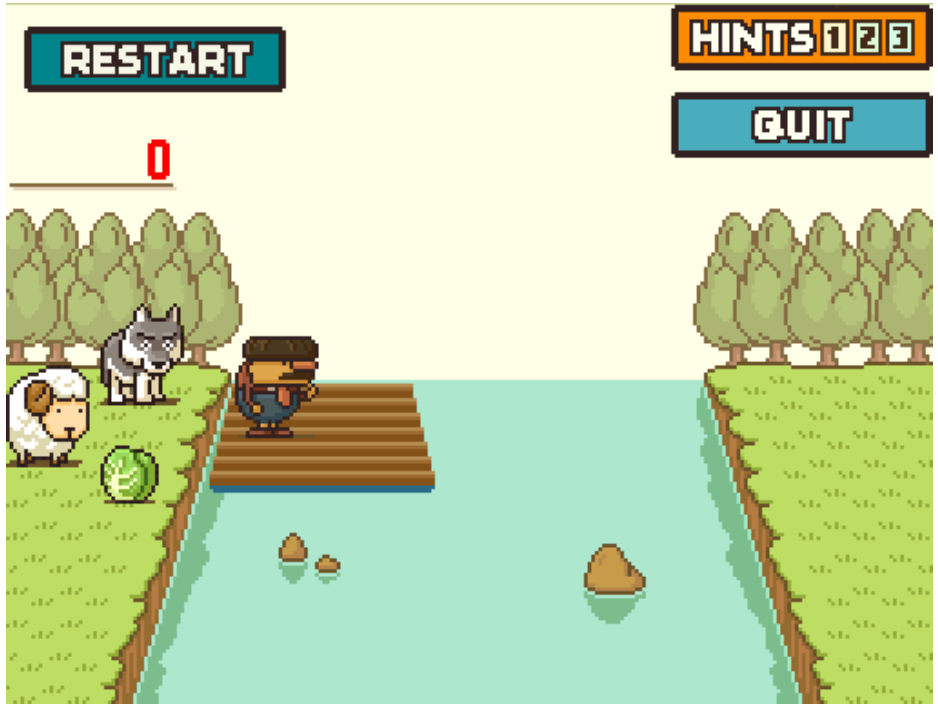
- For simplicity, we will represent a state as follows:

A:8 B:0 C:0

- For now, all costs will be set to 0 since we don't want to optimize anything about the solution (we just want to find any solution).

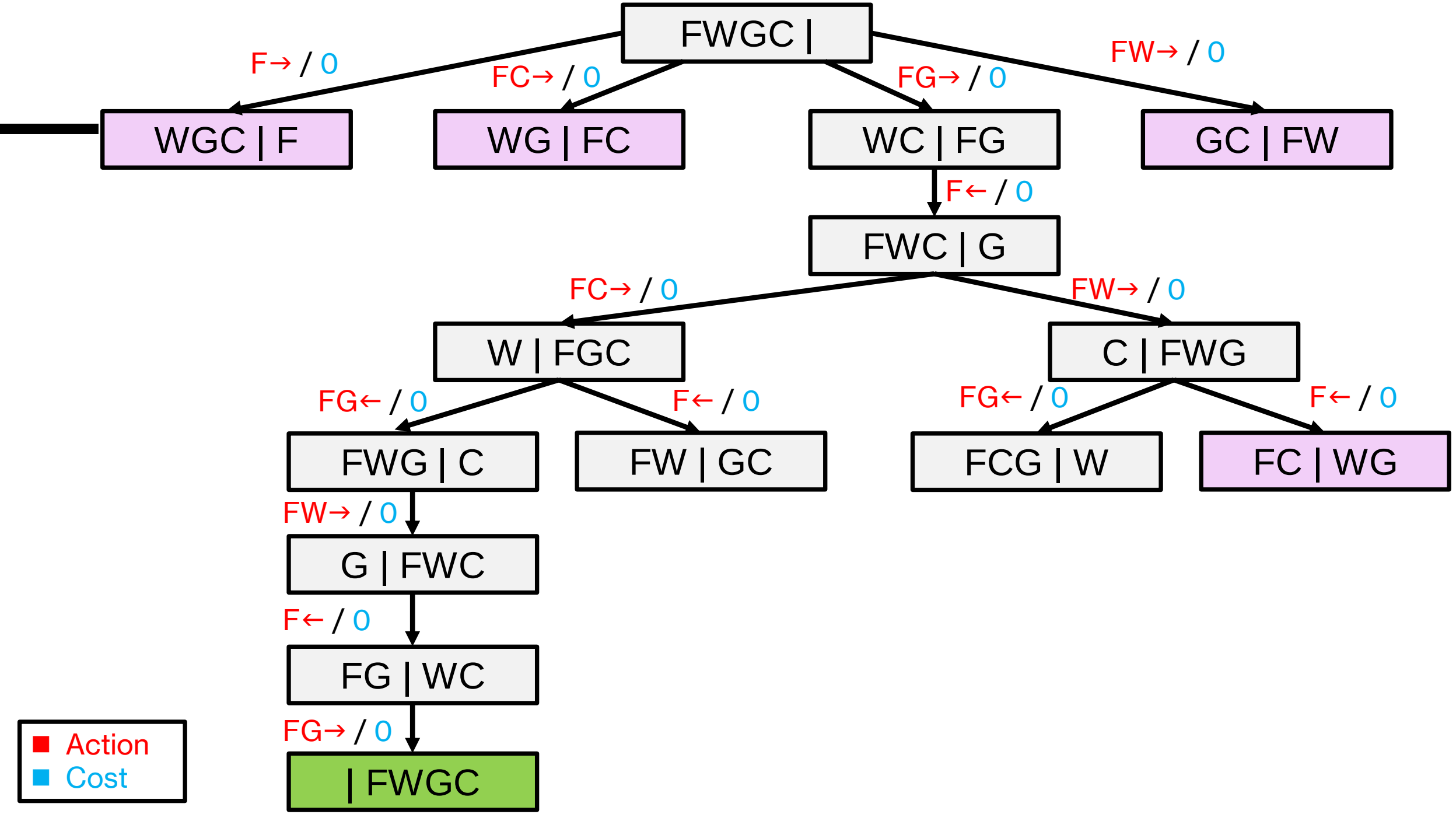

A:8 B:0 C:0
Pour A to B / 0
Pour A to C / 0
A:3 B:5 C:0
A:5 B:0 C:3
Pour A to C / 0
Pour B to C / 0
Pour C to B / 0
A:0 B:5 C:3
A:3 B:2 C:3
A:5 B:3 C:0
Pour C to A / 0
Pour A to C / 0
A:6 B:2 C:0
A:2 B:3 C:3
Pour B to C / 0
Pour C to B / 0
A:6 B:0 C:2
A:2 B:5 C:1
Pour A to B / 0
Pour B to A / 0
A:1 B:5 C:2
A:7 B:0 C:1
Pour B to C / 0
Pour C to B / 0
A:1 B:4 C:3
A:7 B:1 C:0
Pour C to A / 0
A:4 B:4 C:0
...
Action
Cost
Note: We don't include actions that lead us back to states that we've already seen

Wolf, Goat, and Cabbage Puzzle



The puzzle as featured in Professor Layton and the Curious Village (Level-5, 2008)

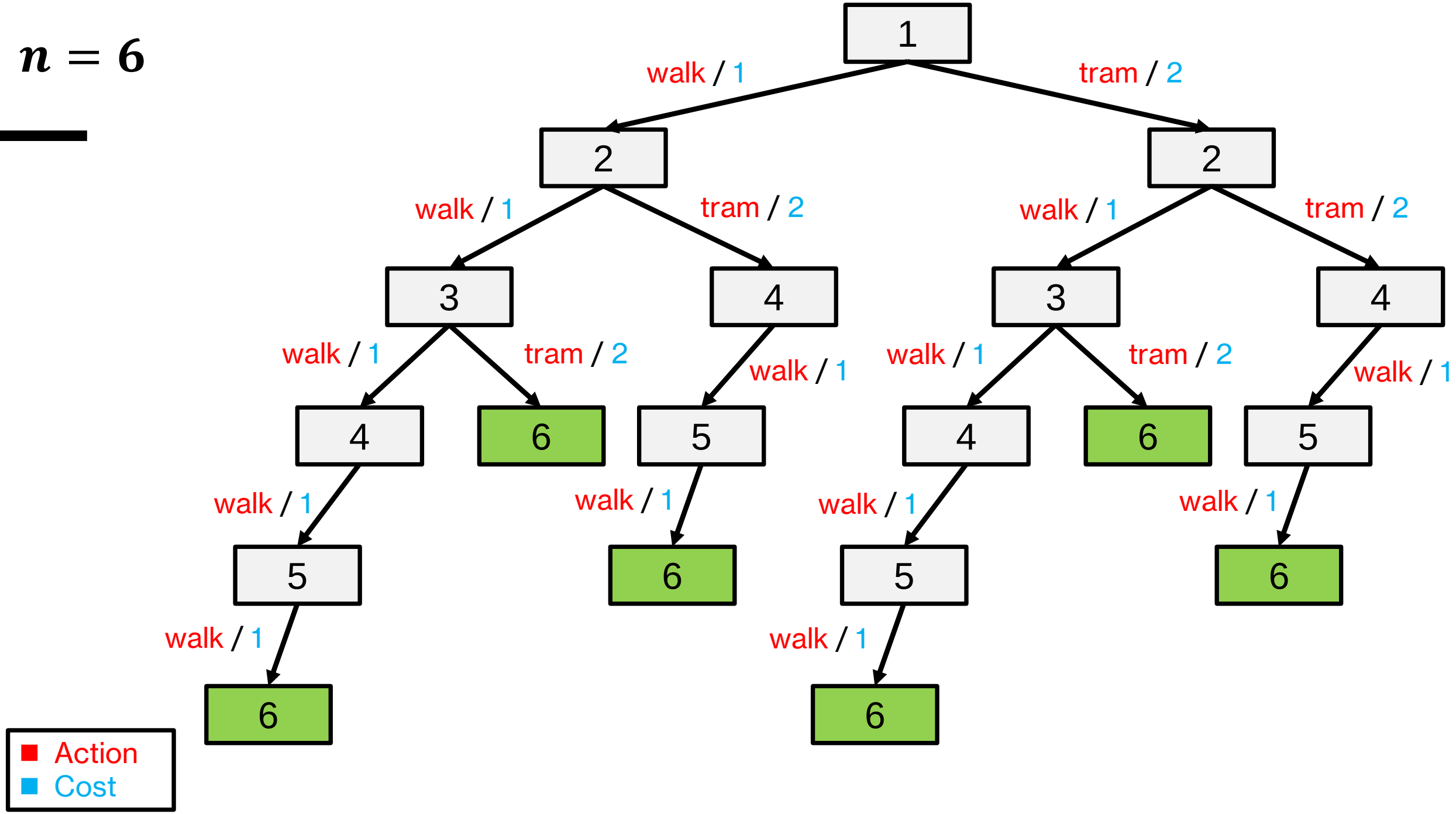
- A **farmer** purchased a **goat**, **wolf**, and **cabbage** and he has to cross the river.
- There is only one raft across, and the farmer can only take one item at a time per crossing.
- If left unattended on either side, the **wolf will eat the goat**, and the **goat will eat the cabbage**.
- Challenge: cross the river with all his possessions intact.


FWGC |
F→ / 0
FC→ / 0
FG→ / 0
FW→ / 0
WGC | F
WG | FC
WC | FG
GC | FW
F← / 0
FWC | G
FC→ / 0
FW→ / 0
W | FGC
C | FWG
FG← / 0
F← / 0
FG← / 0
F← / 0
FWG | C
FW | GC
FCG | W
FC | WG
FW→ / 0
G | FWC
F← / 0
FG | WC
FG→ / 0
| FWGC
Action
Cost

Transport Puzzle

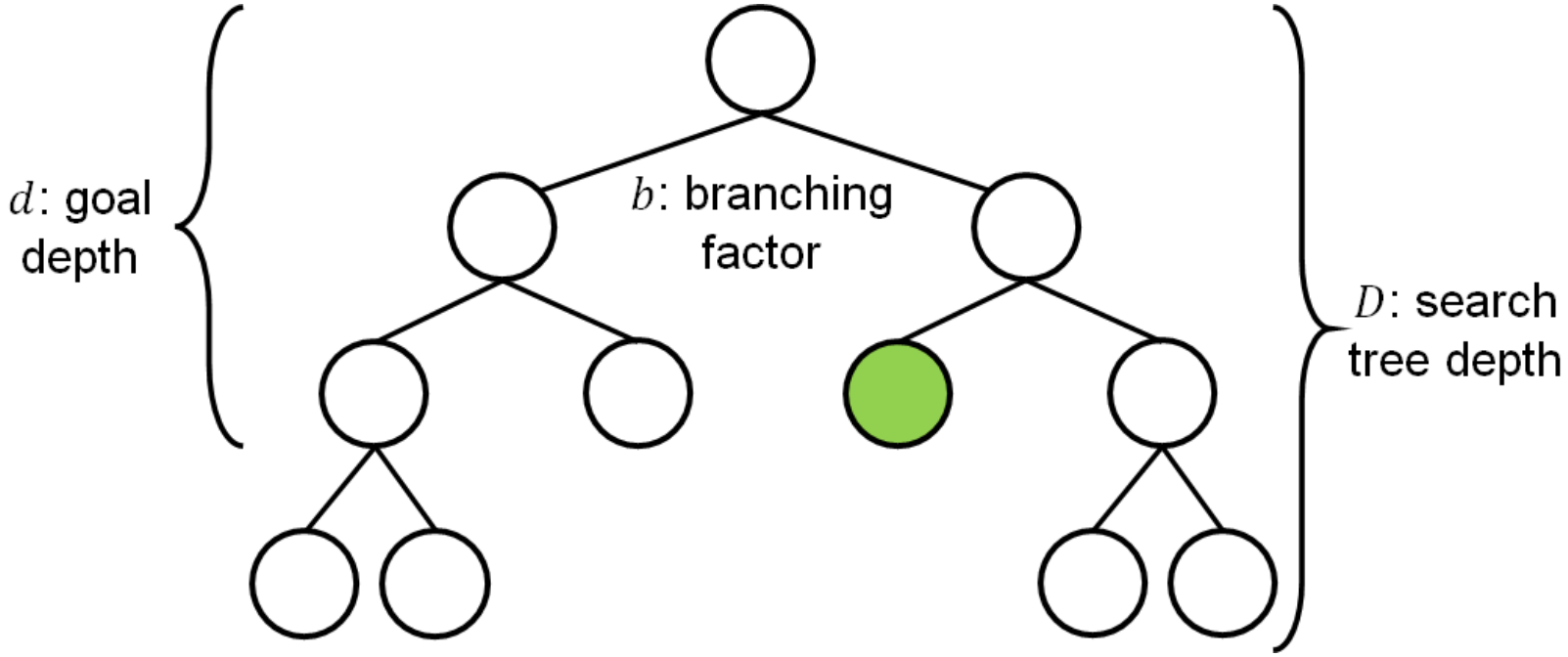
- There are streets with blocks numbered 1 to n.
- Walking from s to $(s + 1)$ takes 1 minute.
- Taking a magic tram from s to $(s \times 2)$ takes 2 minutes.
- How do you travel from 1 to n in the least amount of time?

$n = 6$


1
walk / 1
tram / 2
2
2
walk / 1
tram / 2
walk / 1
tram / 2
3
4
3
4
walk / 1
tram / 2
walk / 1
walk / 1
tram / 2
walk / 1
4
6
5
4
6
5
walk / 1
walk / 1
walk / 1
walk / 1
5
6
5
6
walk / 1
walk / 1
6
6
Action
Cost

Size of a Problem

- Total number of possible states



- 8-puzzle: 181440 states
- Tic-Tac-Toe: 3^9 states
- Rubik's Cube: 10^{19} states
- Chess: 10^{111} to 10^{123} states

Inference: Tree Search Algorithms

- Backtracking Search
- Depth-First Search (DFS)
- Breadth-First Search (BFS)
- DFS with Iterative Deepening (DFS-ID)

Evaluating a Search Algorithm

- **Completeness**: does it always find a solution if it exists?
- **Time Complexity**: how many states does it have to explore before returning a solution? (worst case)
- **Space Complexity**: how many states does it have to store in memory at any given point in time? (worst case)
- **Optimality**: does it always return the solution with least total cost?

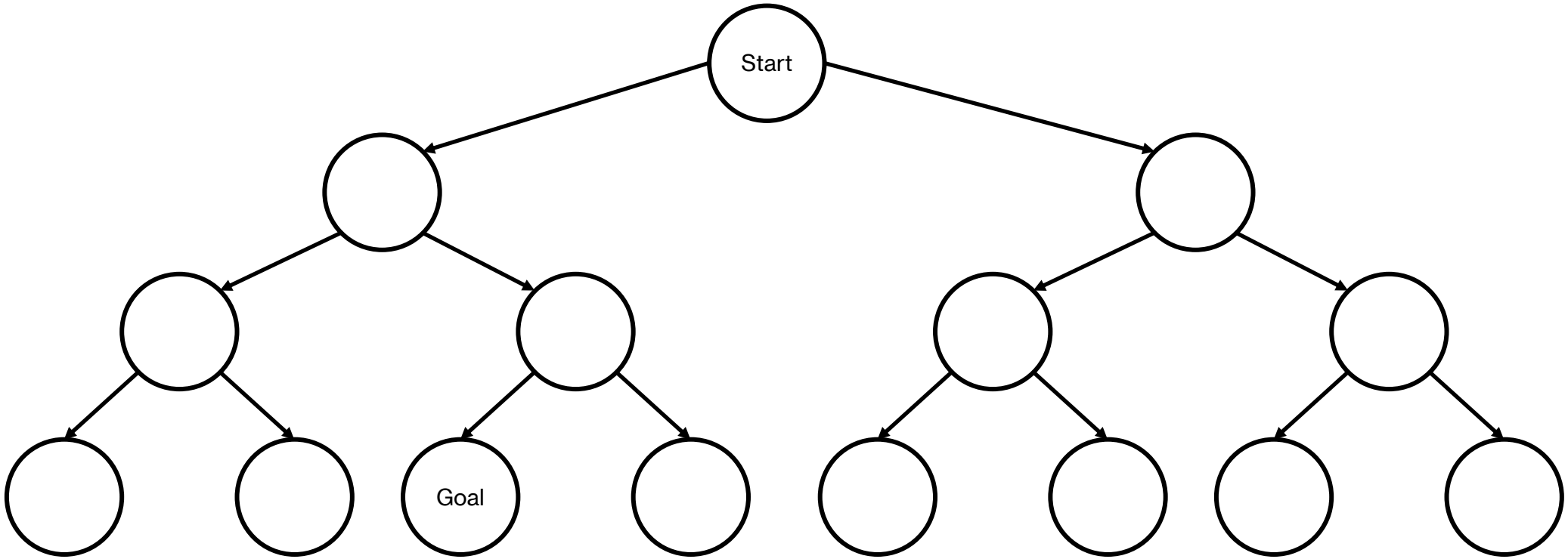
Backtracking Search

- Explore **all** states naively.

ALGORITHM

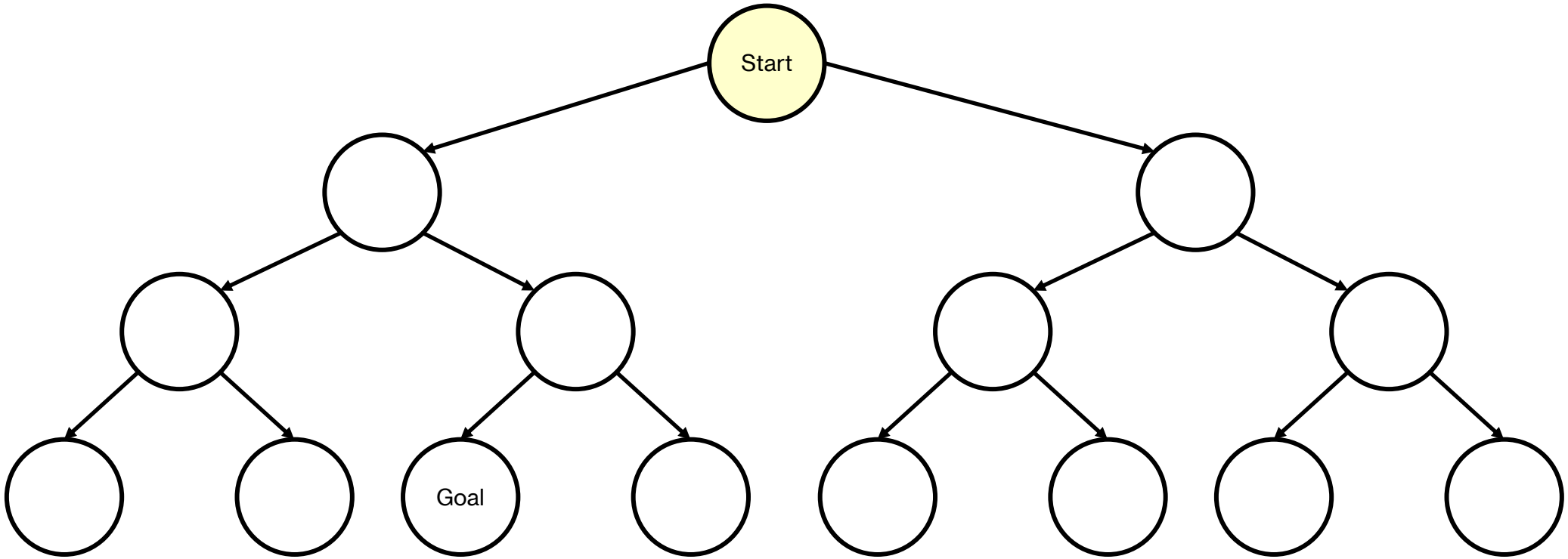
```
def backtrackingSearch(s, path)
    if IsEnd(s) then
        Update the minimum cost path
    for each a ∈ Actions(s) do
        Extend path with Succ(s, a) and Cost(s, a)
        Call backtrackingSearch(Succ(s, a), path)
    Return the minimum cost path
```

Backtracking Search



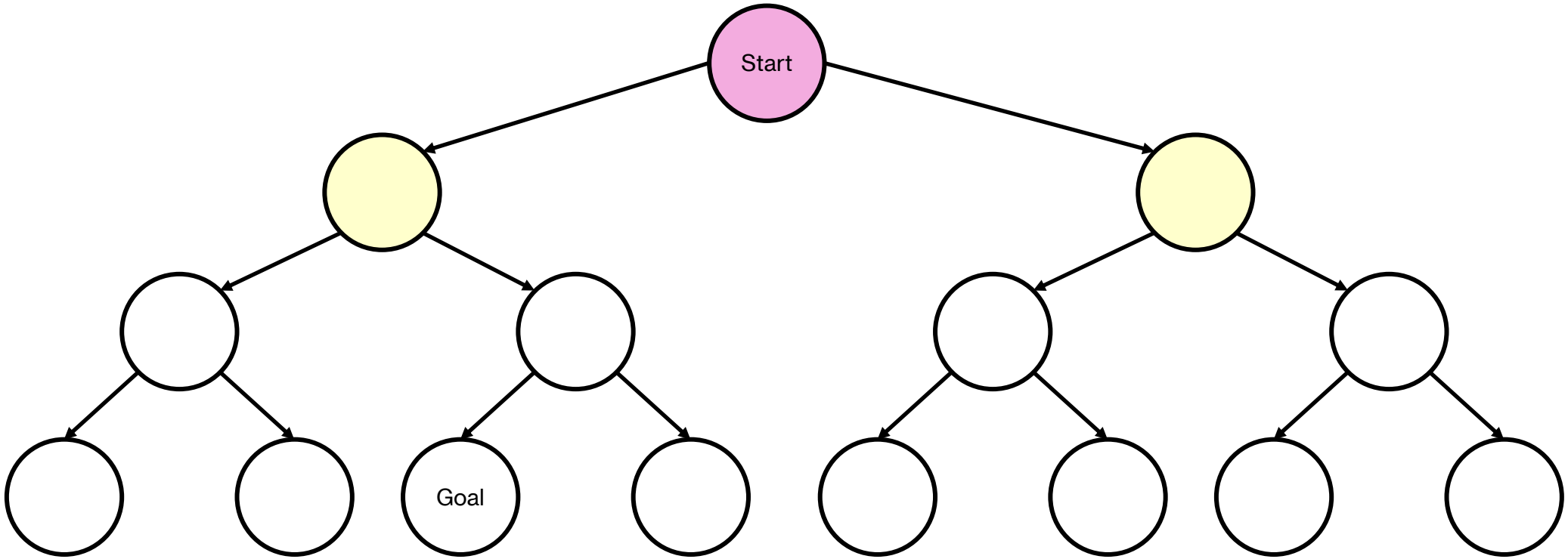
- Frontier
- Explored

Backtracking Search



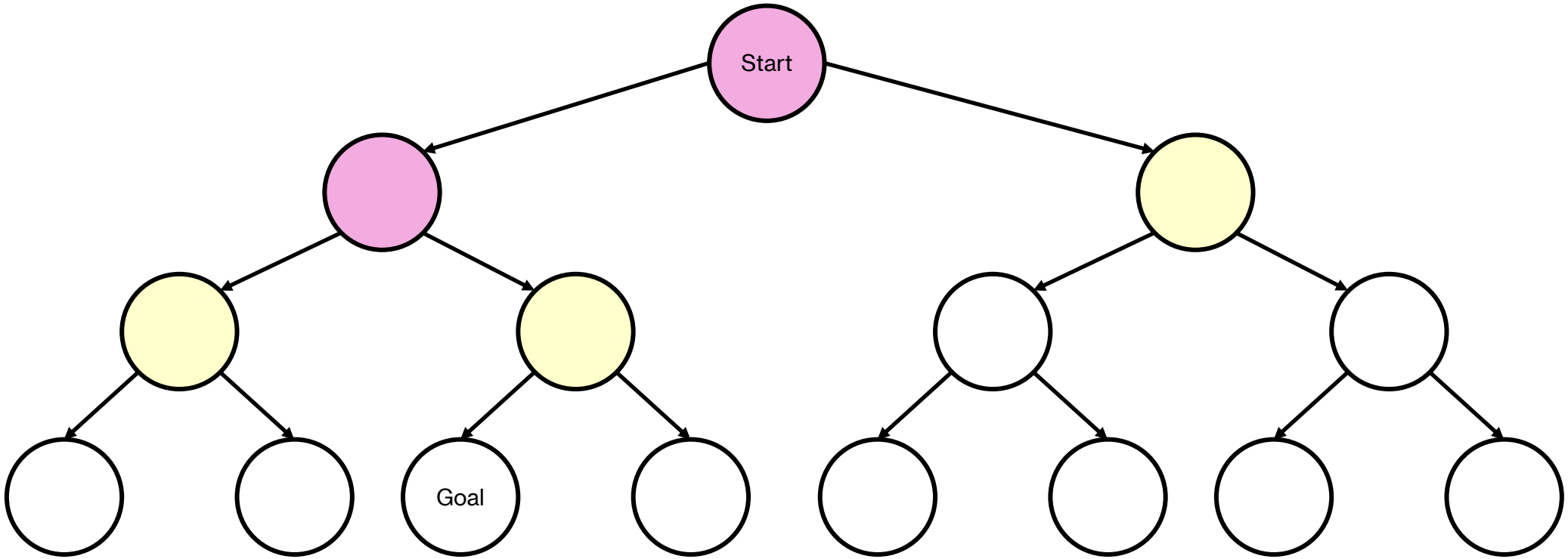
- Explored
- Frontier

Backtracking Search



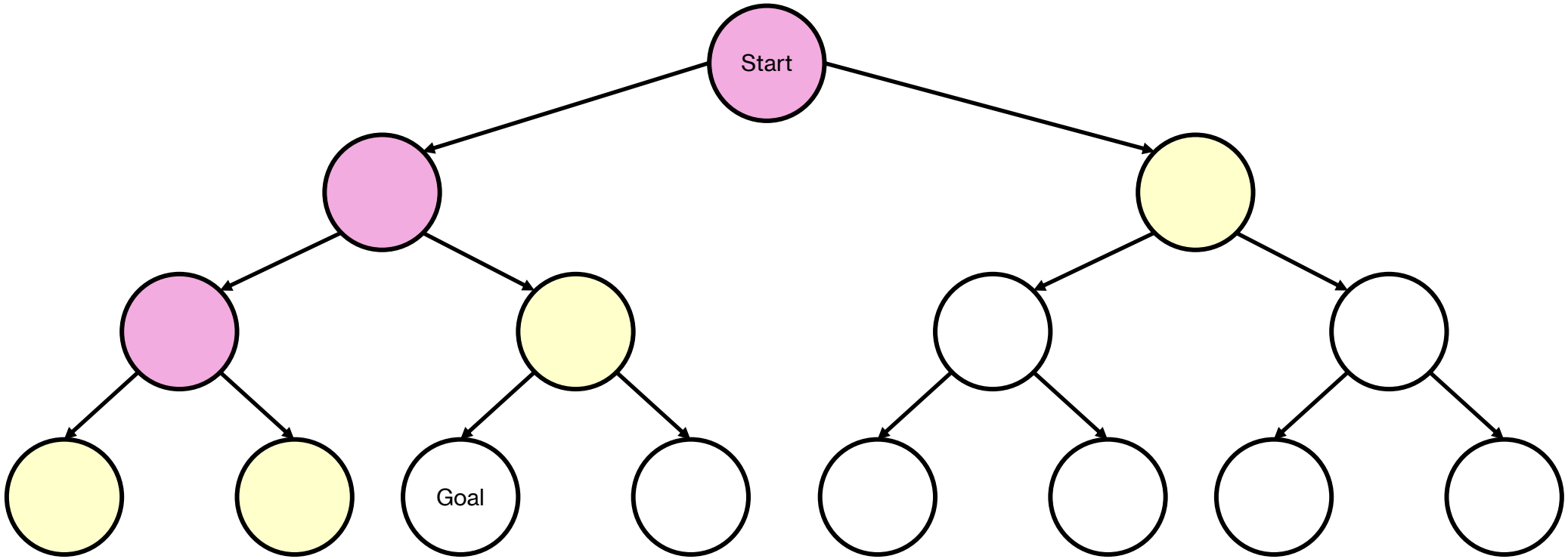
- Explored
- Frontier

Backtracking Search



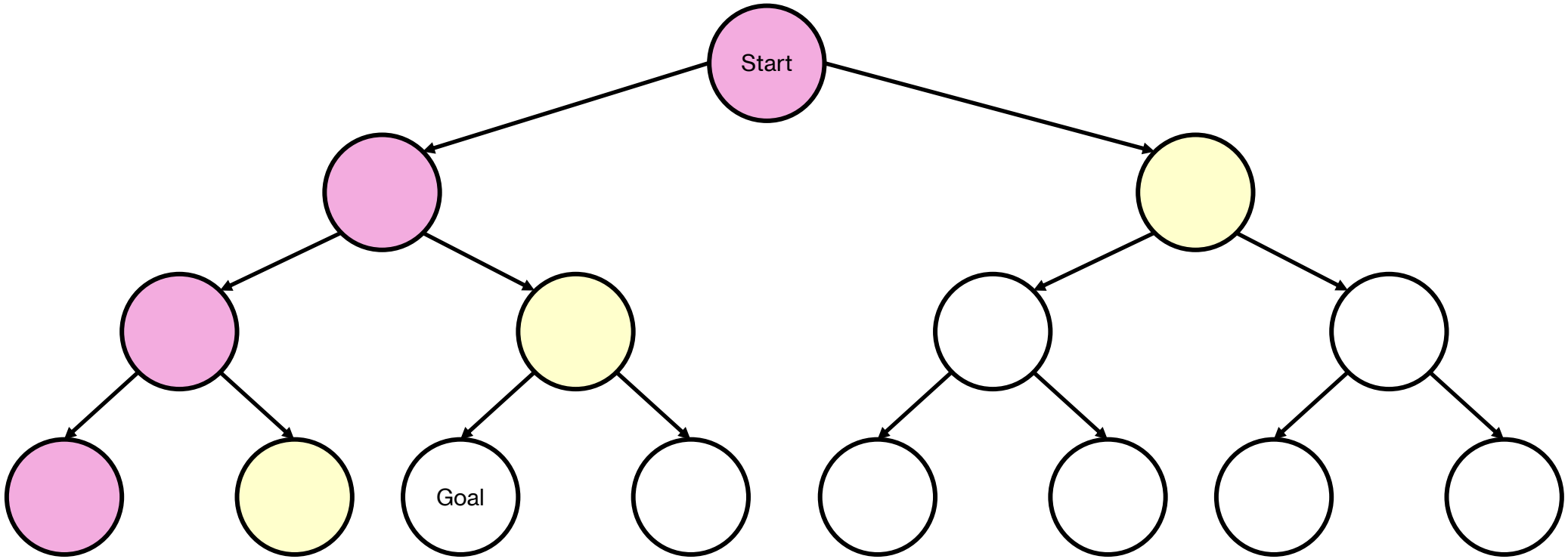
- Explored
- Frontier

Backtracking Search



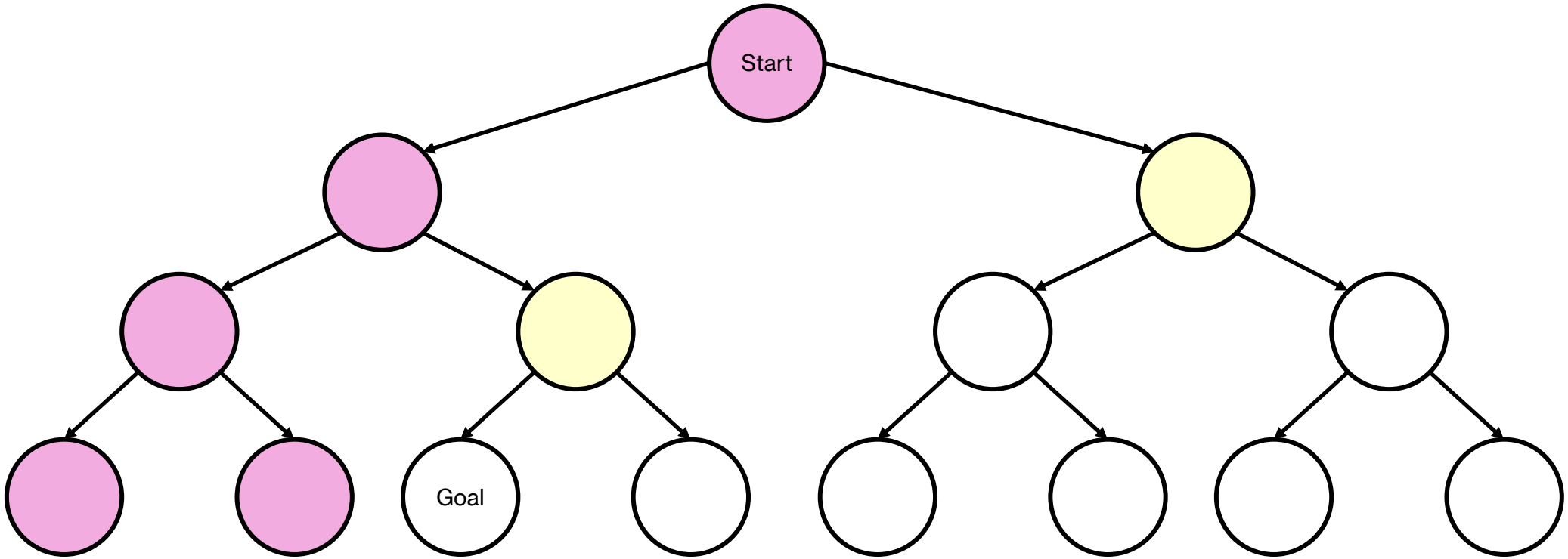
- Explored
- Frontier

Backtracking Search



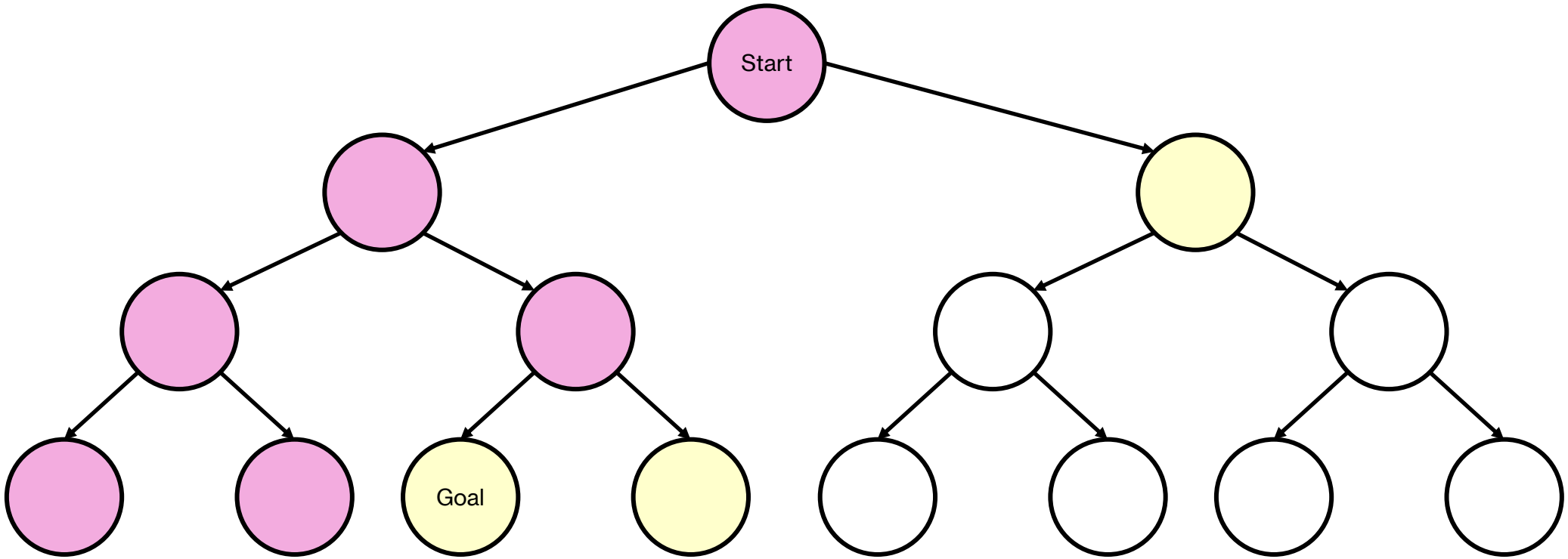
Explored
Frontier

Backtracking Search



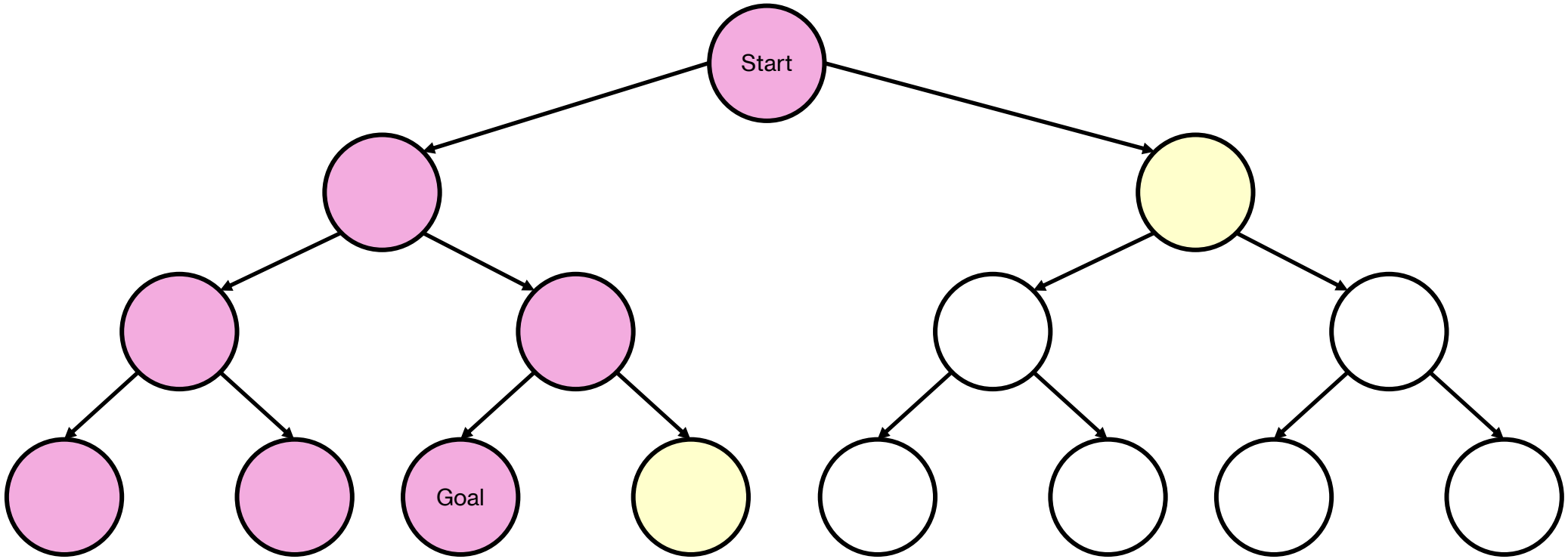
- Explored
- Frontier

Backtracking Search



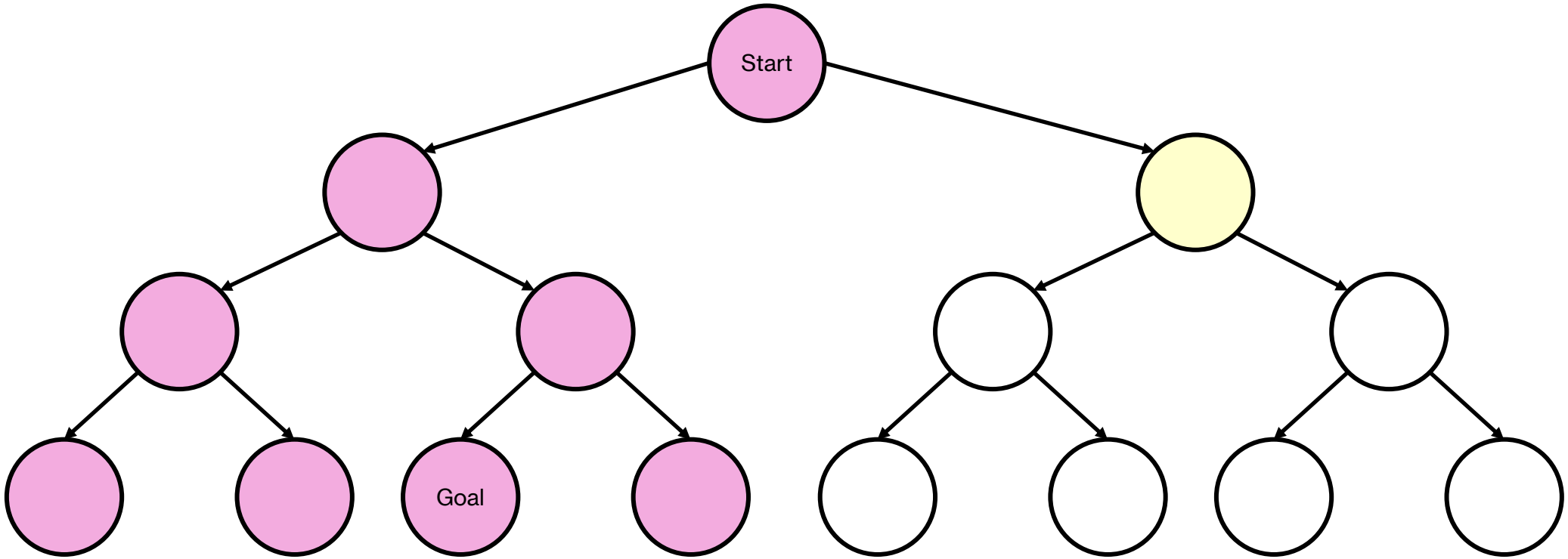
- Explored
- Frontier

Backtracking Search



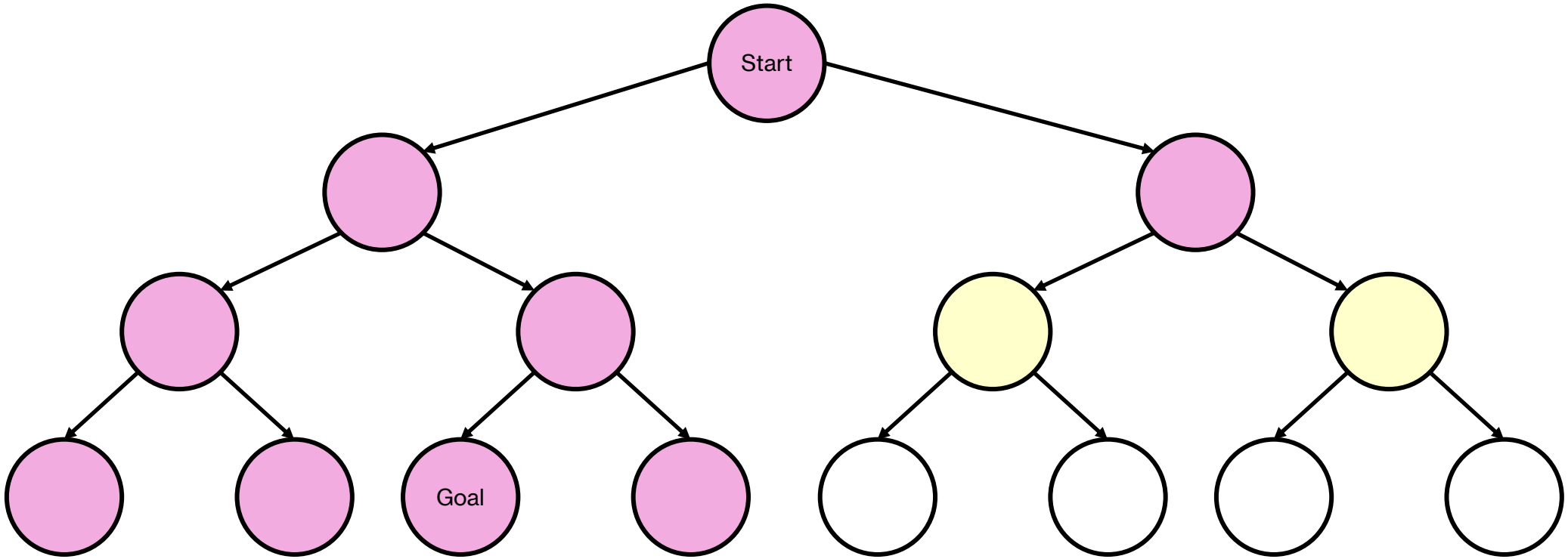
Explored
Frontier

Backtracking Search



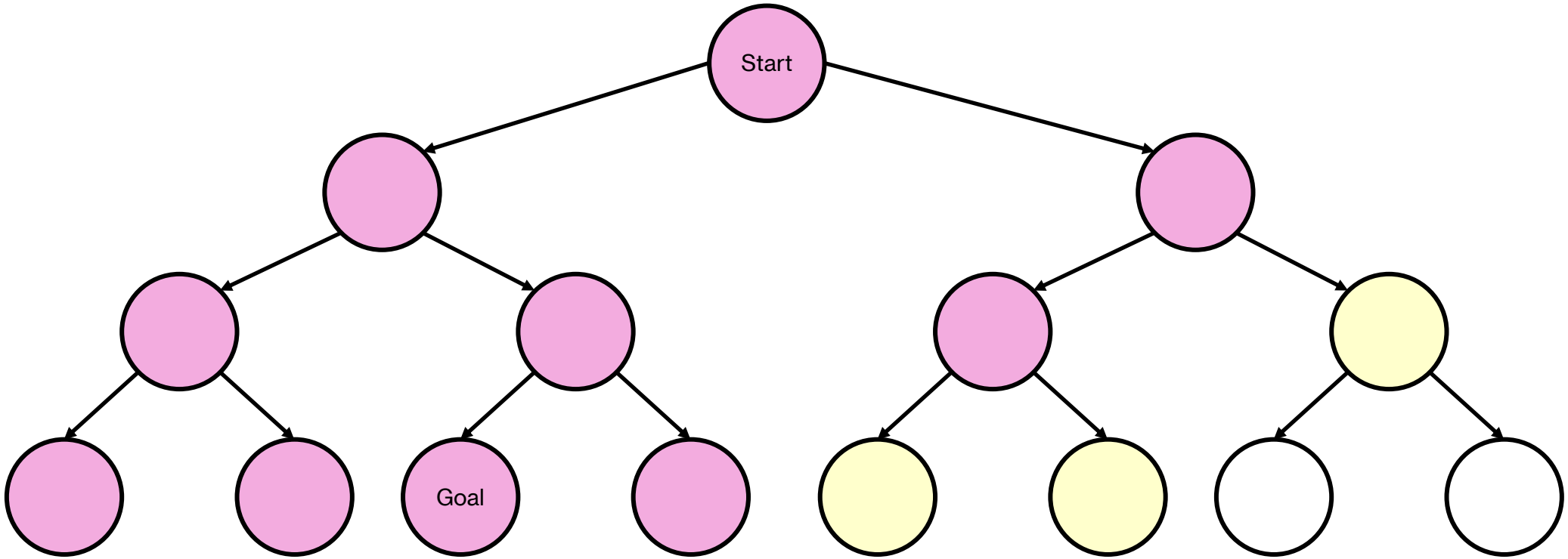
- Explored
- Frontier

Backtracking Search



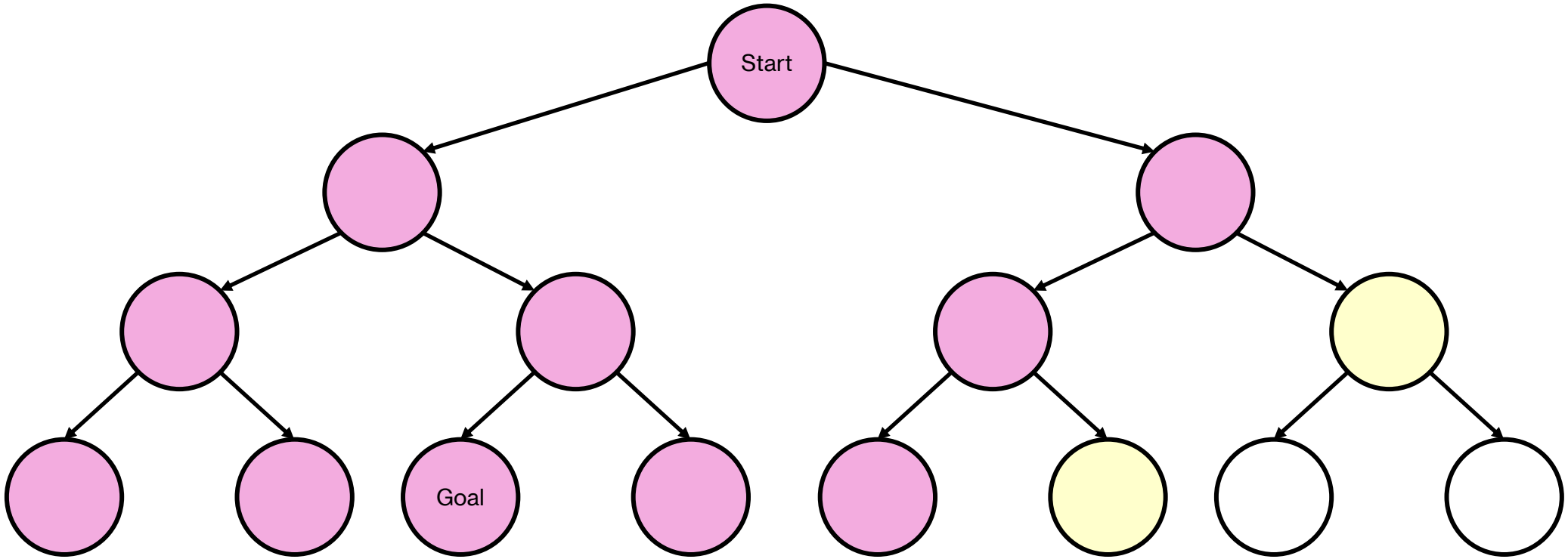
Explored
Frontier

Backtracking Search



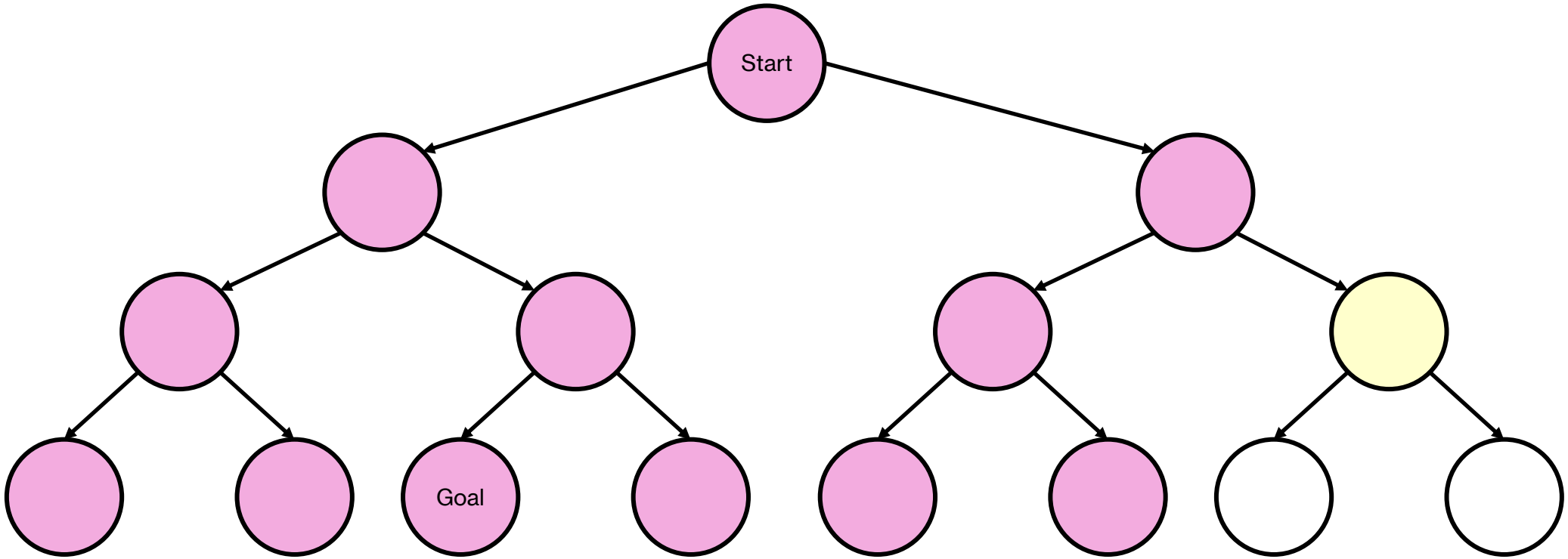
Explored
Frontier

Backtracking Search



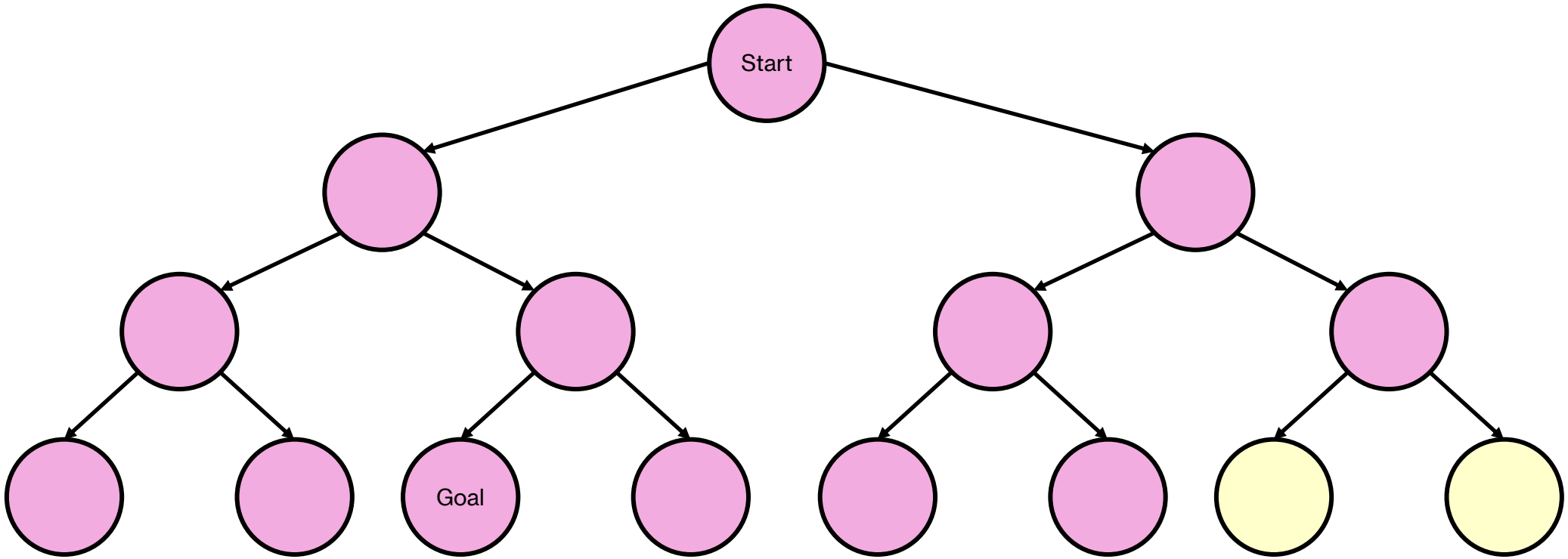
Explored
Frontier

Backtracking Search



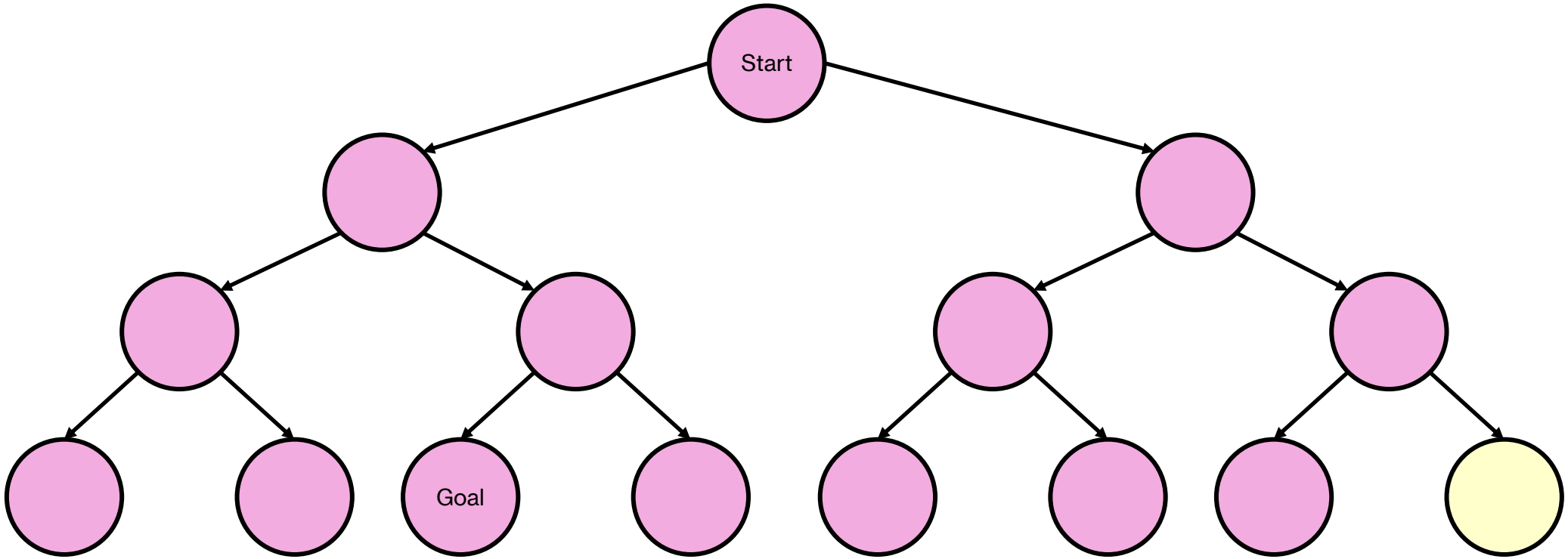
Explored
Frontier

Backtracking Search



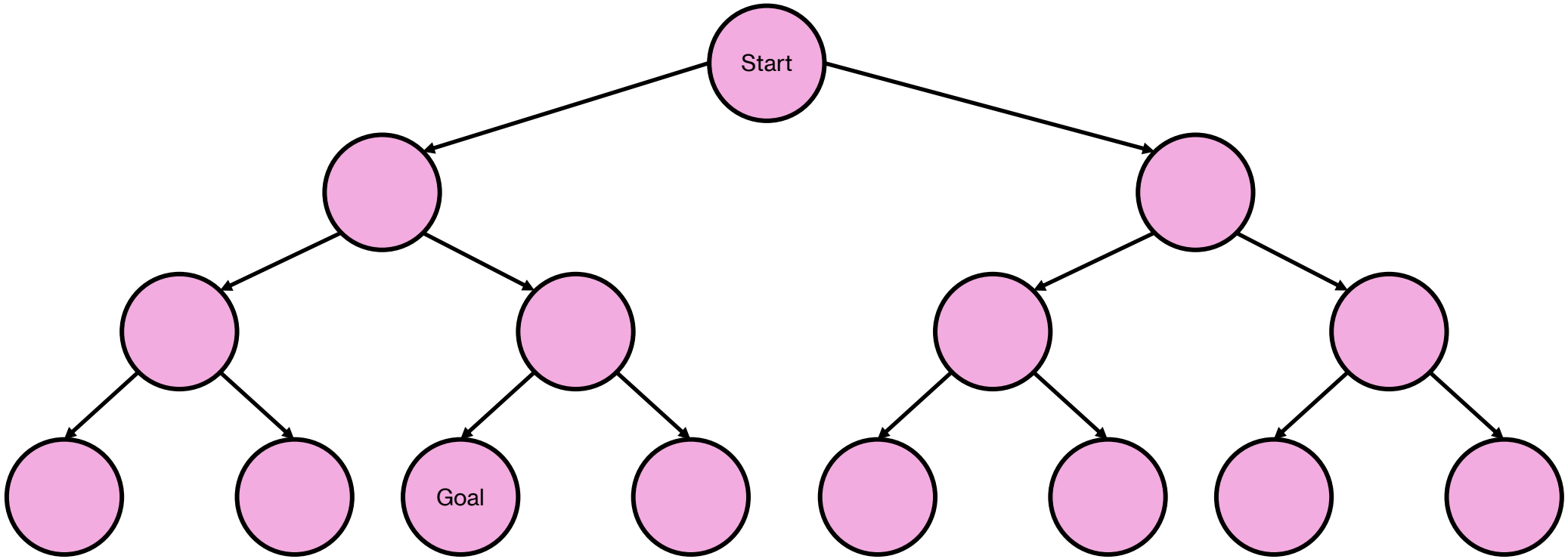
- Explored
- Frontier

Backtracking Search



- Explored
- Frontier

Backtracking Search



- Explored
- Frontier

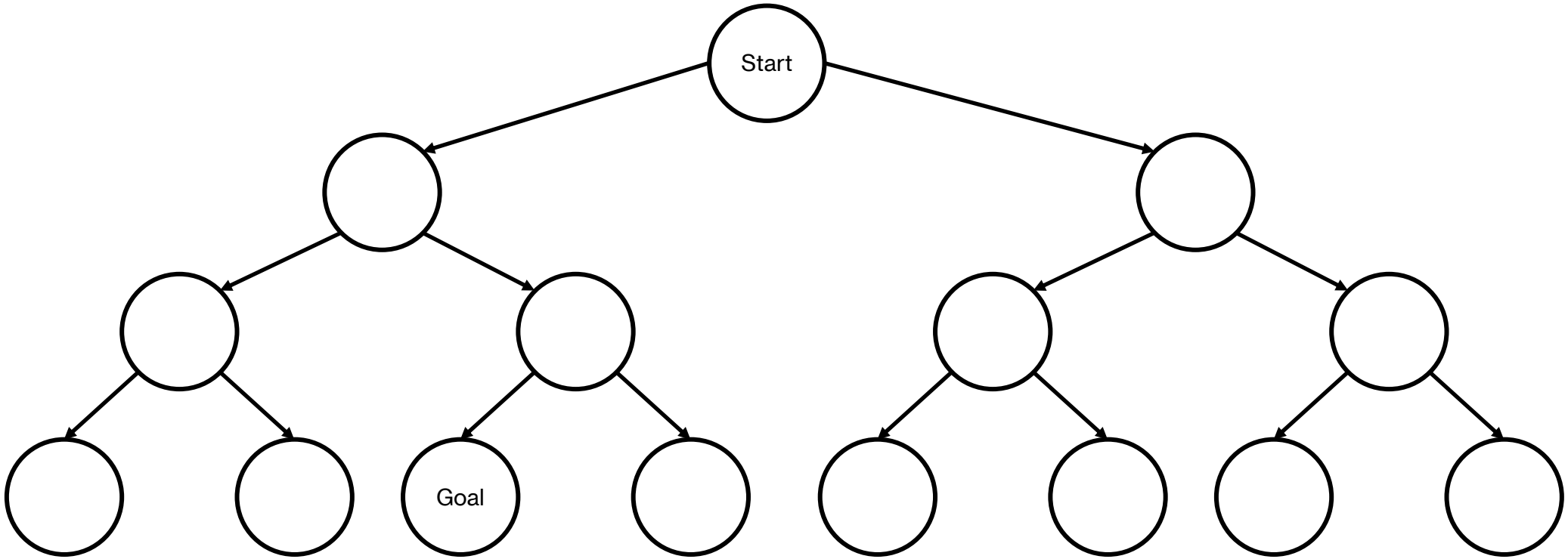
Backtracking Search Characteristics

- If state space is finite, it is **complete** and **optimal**.
- Time complexity: $O(D)$
- Space complexity: $O(b^D)$

Depth-First Search

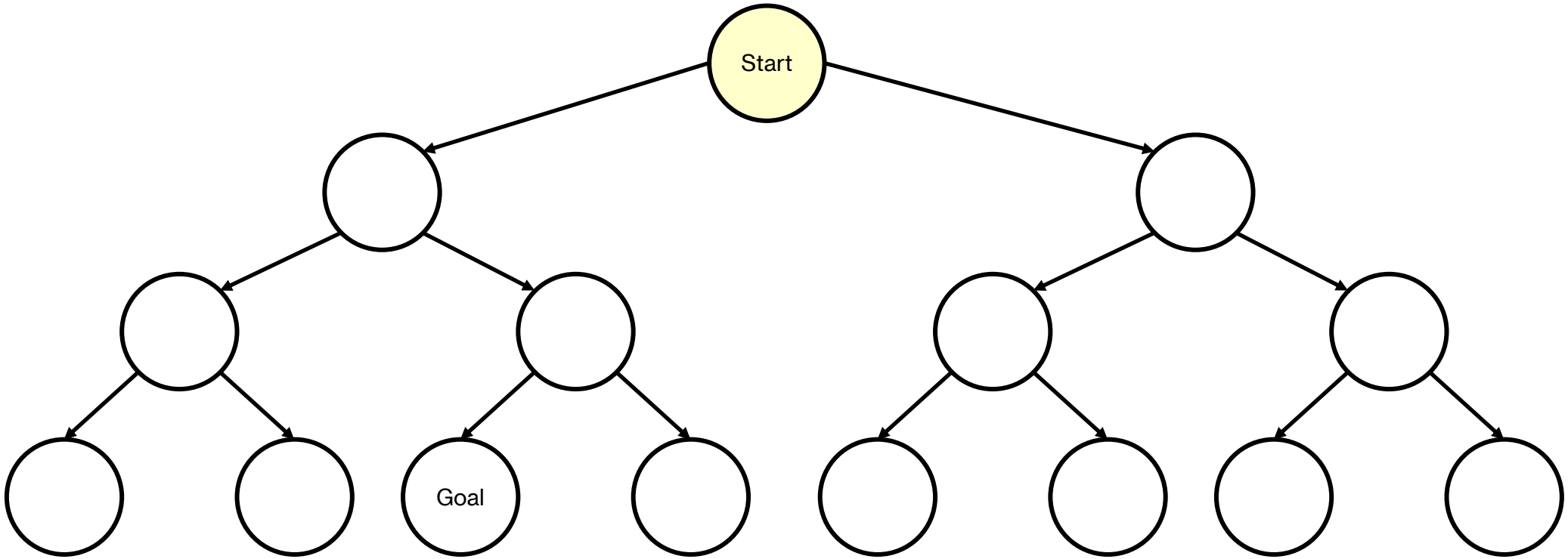
- Prioritize exploring deeper states
- Stop once a solution is found

Depth-First Search



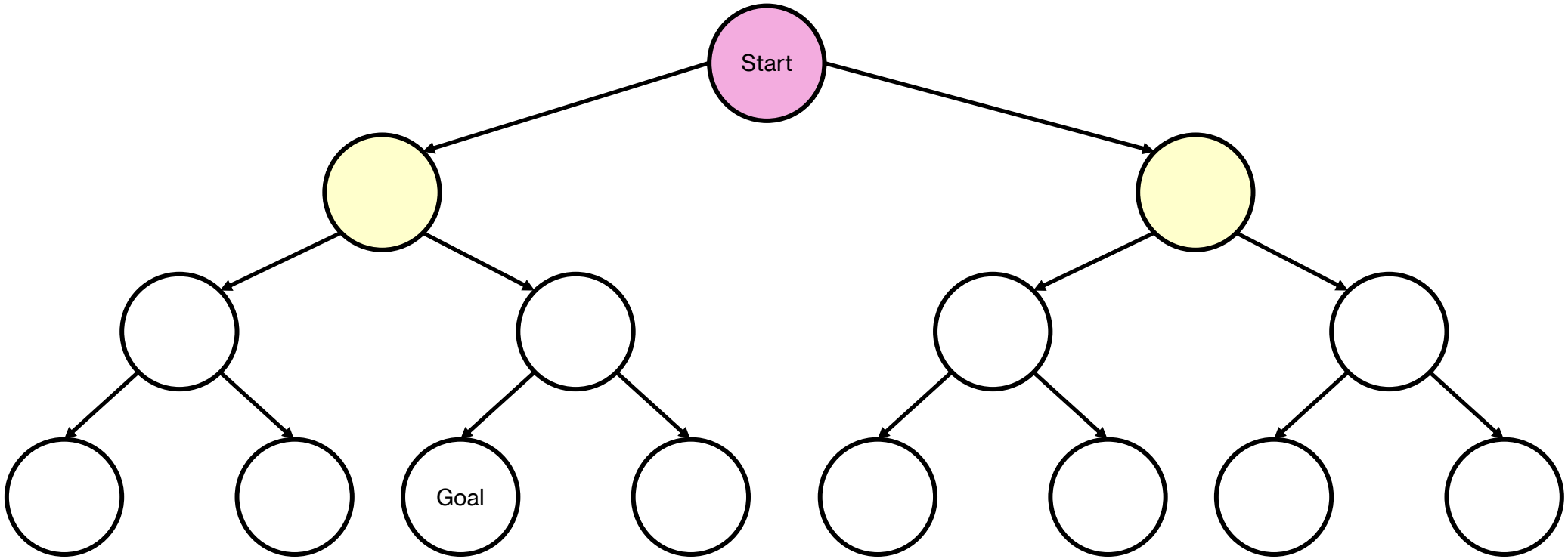
- Explored
- Frontier

Depth-First Search



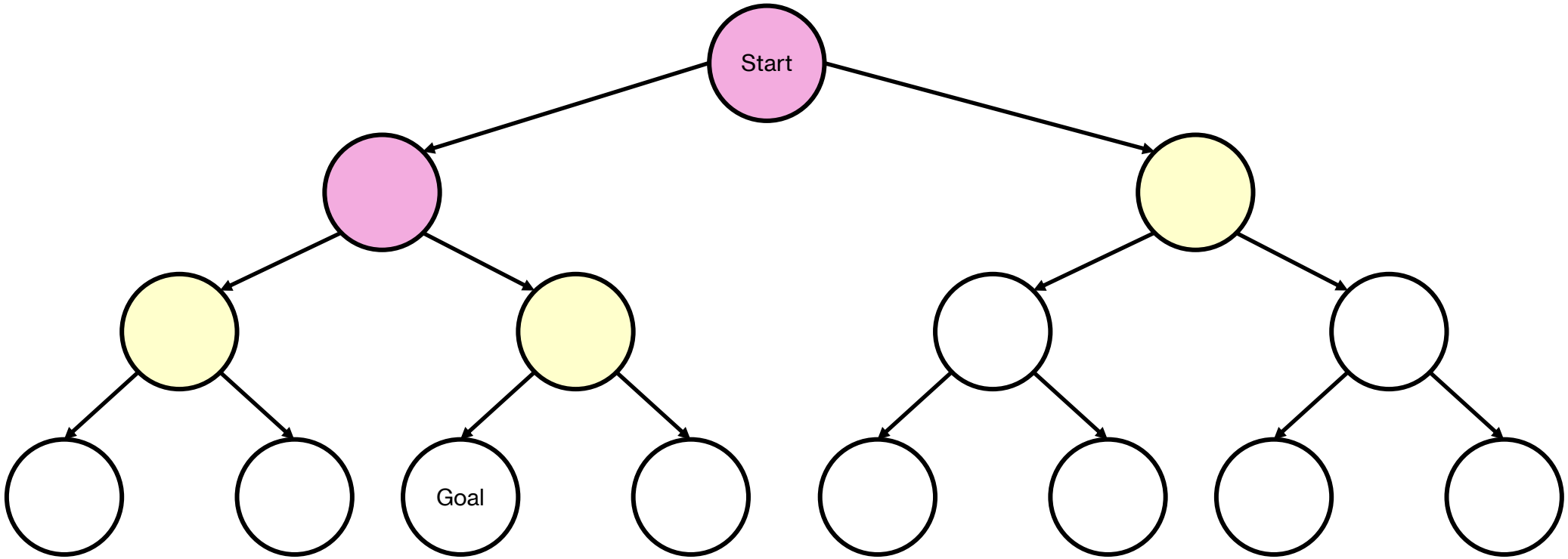
- Explored
- Frontier

Depth-First Search



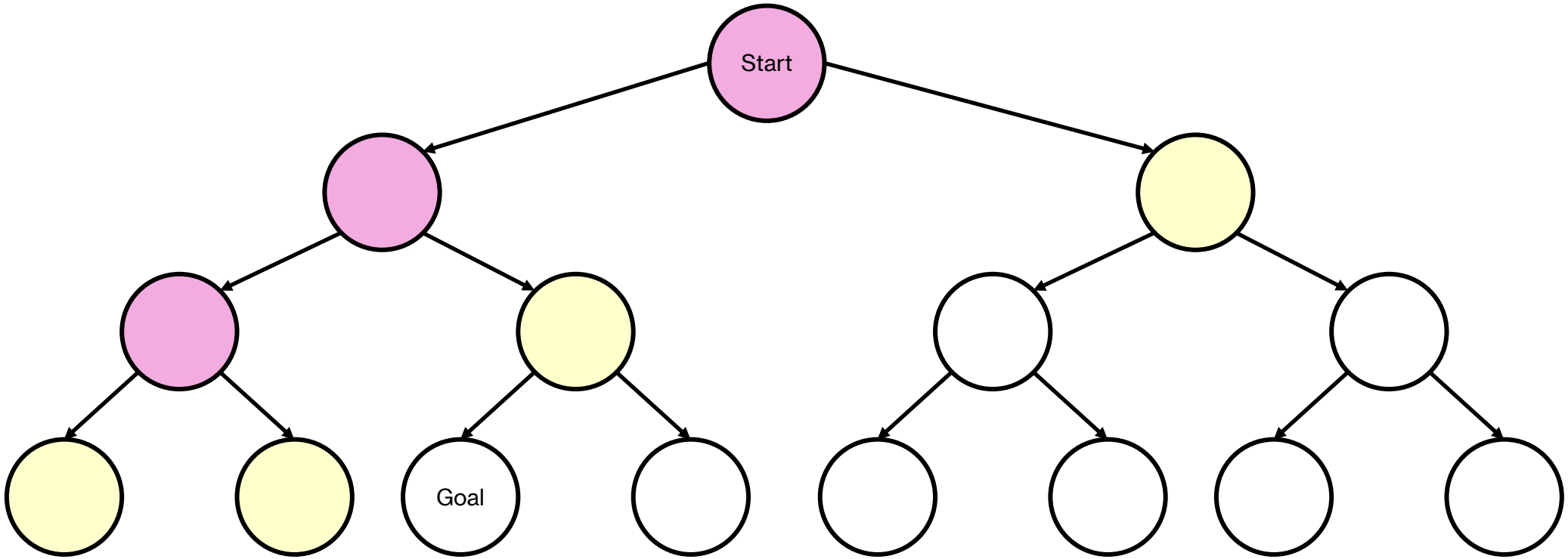
- Explored
- Frontier

Depth-First Search



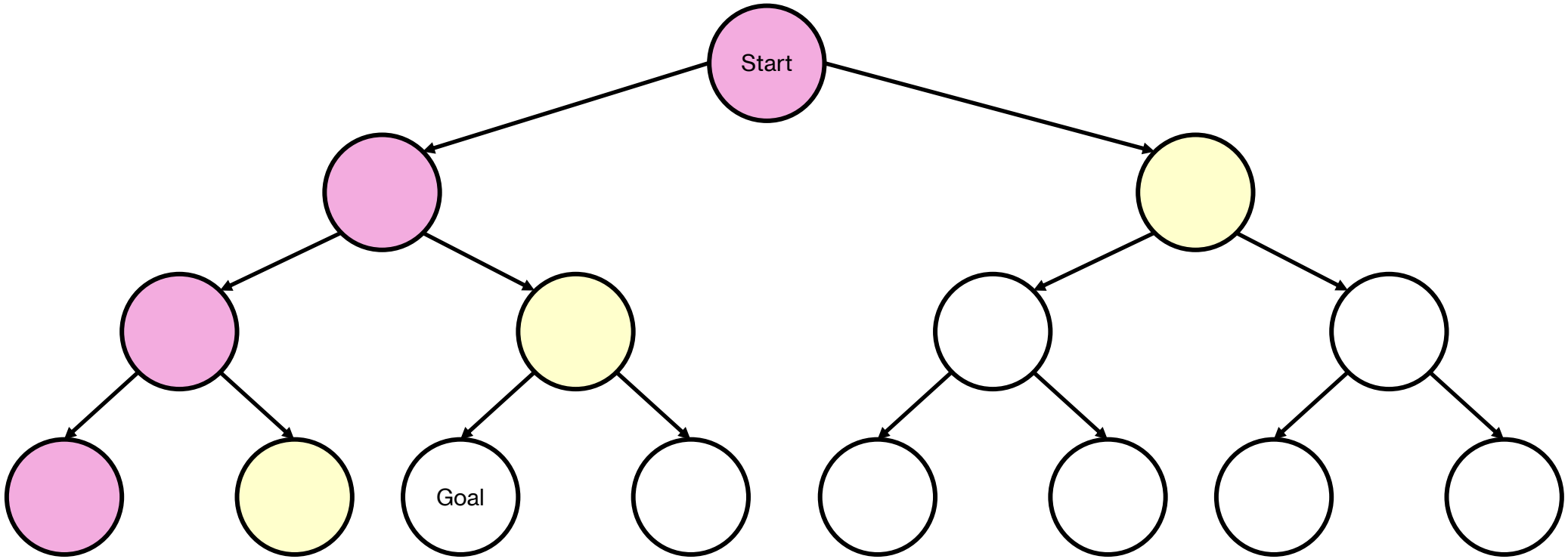
- Explored
- Frontier

Depth-First Search



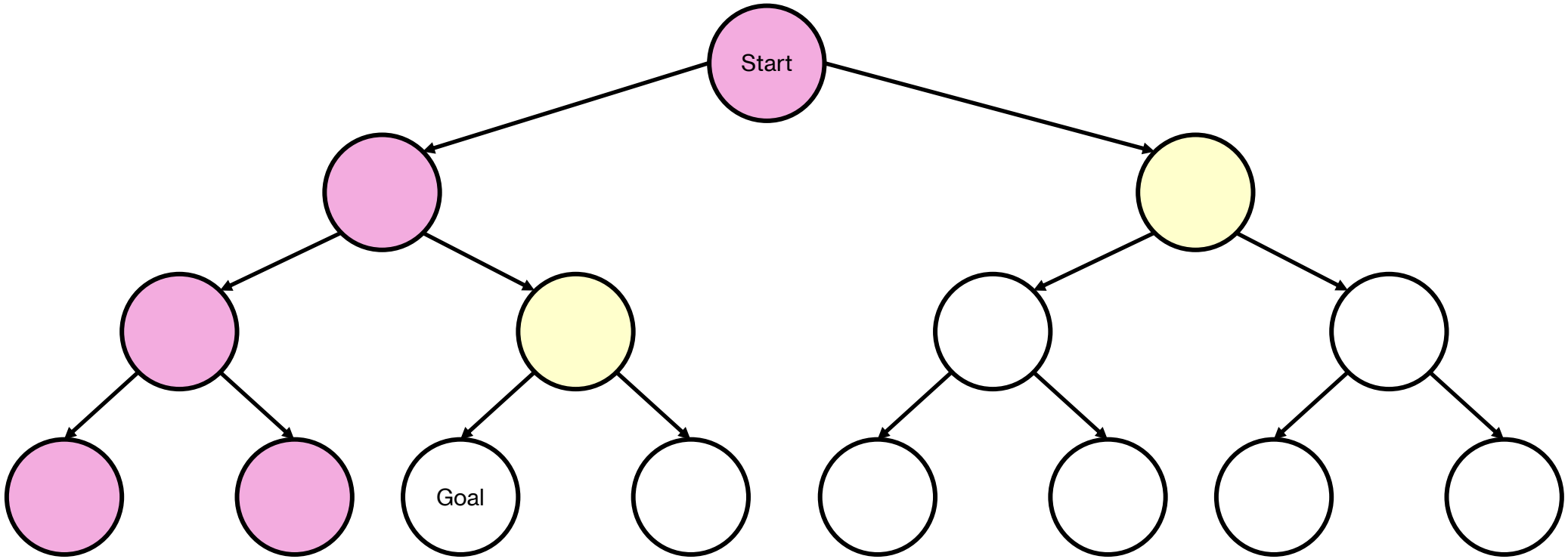
- Explored
- Frontier

Depth-First Search



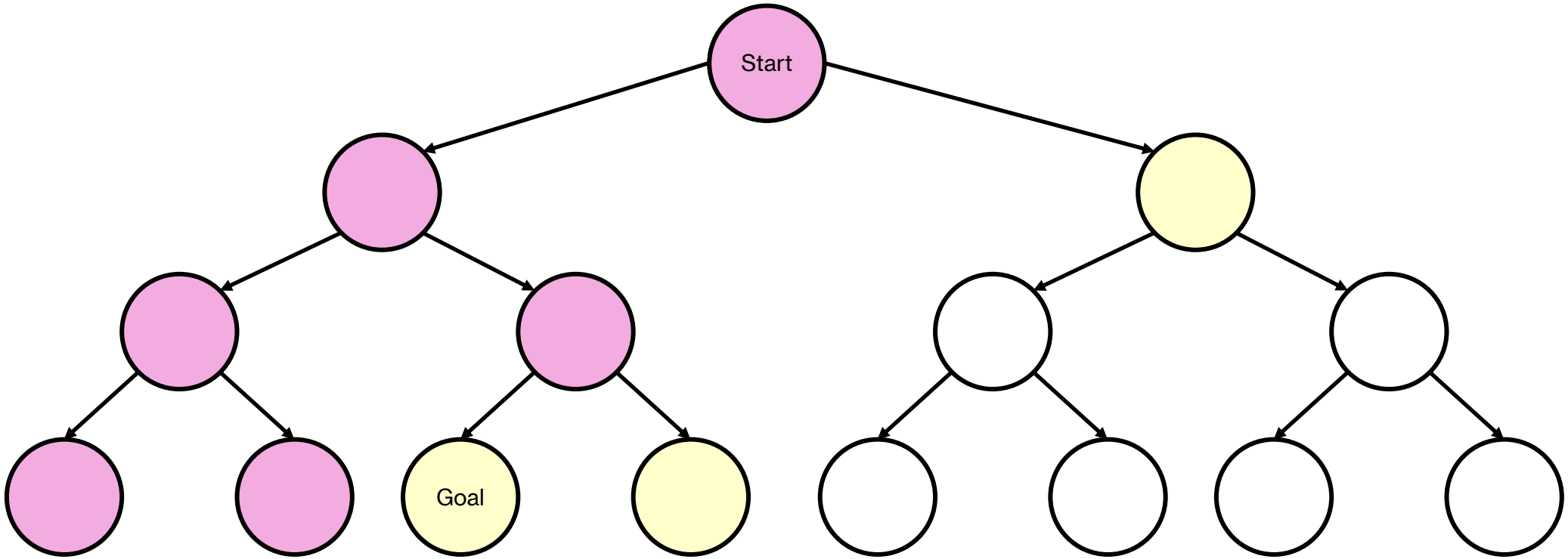
- Explored
- Frontier

Depth-First Search



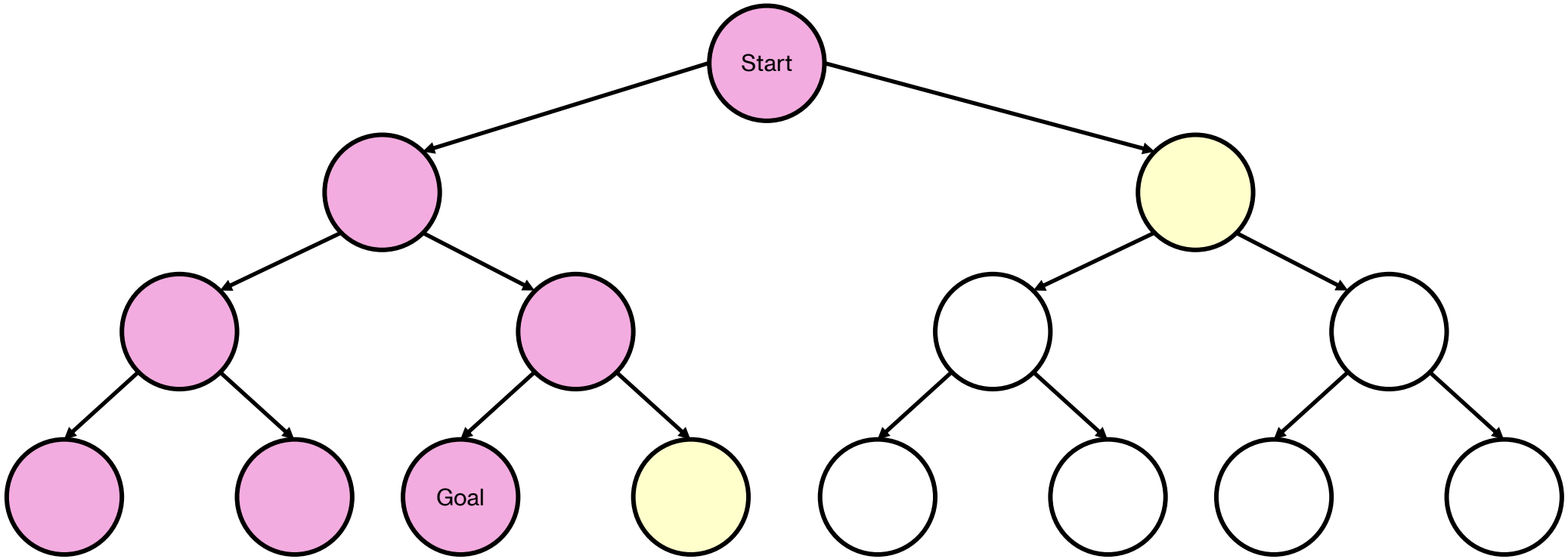
- Explored
- Frontier

Depth-First Search



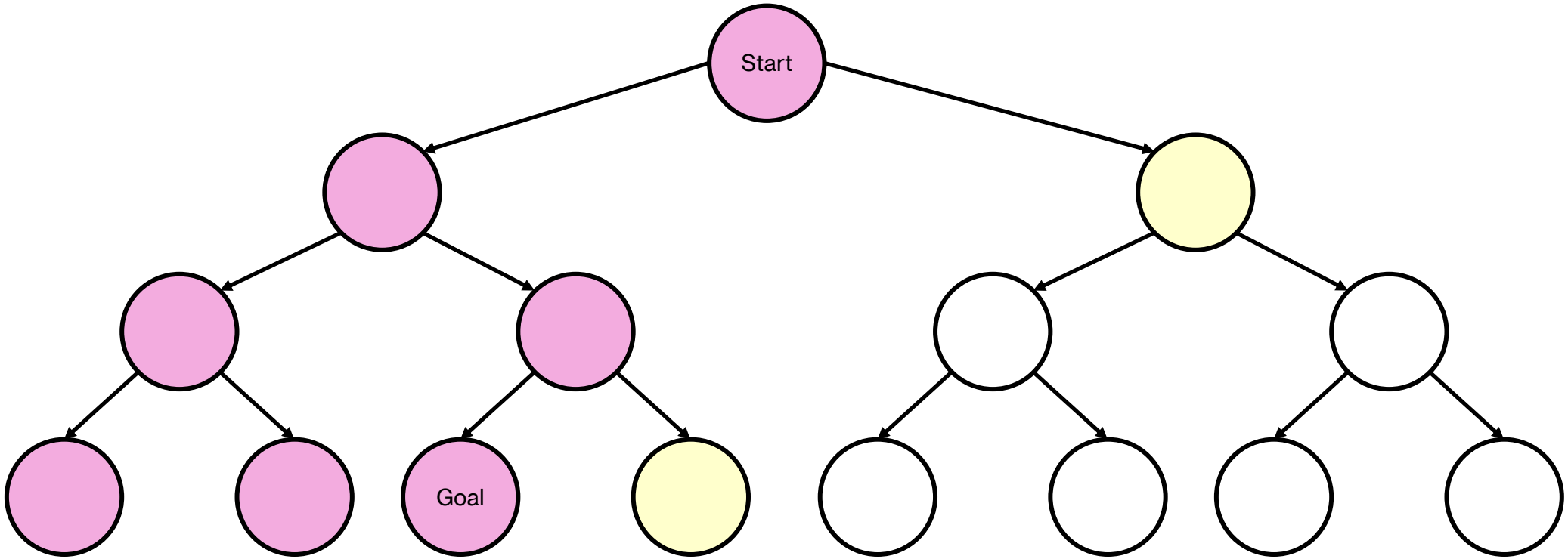
- Explored
- Frontier

Depth-First Search



- Explored
- Frontier

Depth-First Search



Return Solution!

- Explored
- Frontier

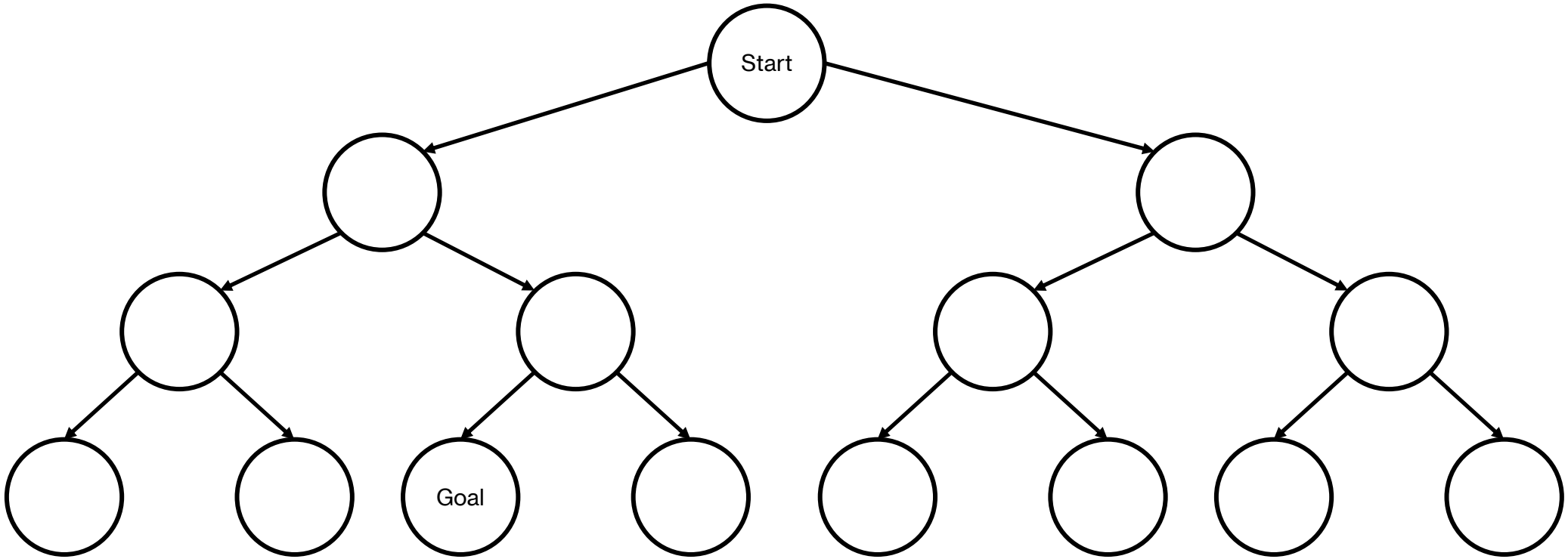
Depth-First Search Characteristics

- If state space is finite, it is **complete** but not **optimal**.
- If state space is infinite, it is **not complete**.
- Time complexity: $O(D)$
- Space complexity: $O(b^D)$

Breadth-First Search

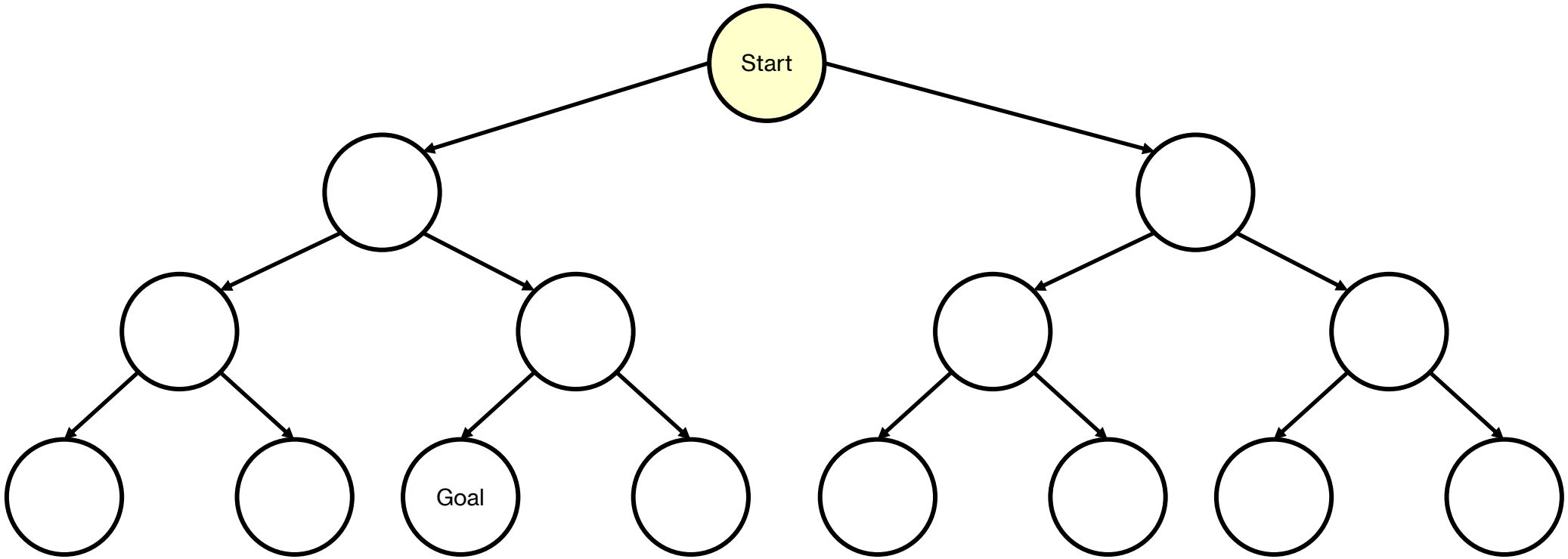
- Prioritize exploring shallower states (smallest to largest depth)
- Stop once a solution is found

Breadth-First Search



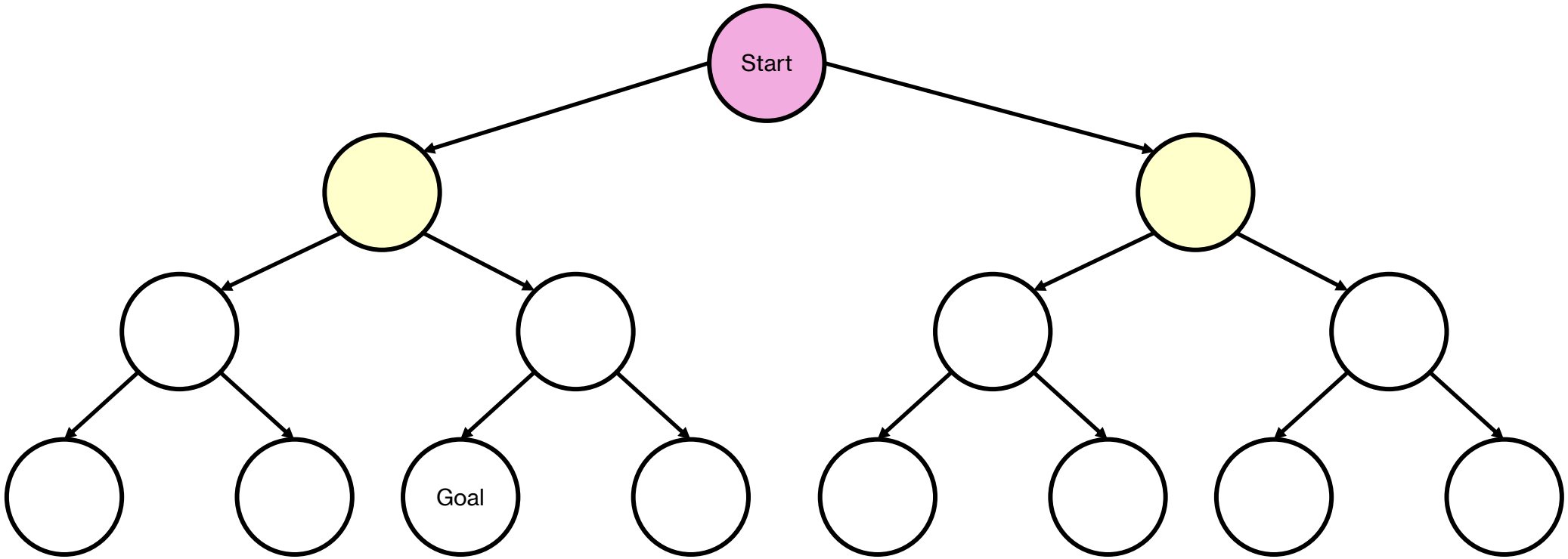
- Explored
- Frontier

Breadth-First Search



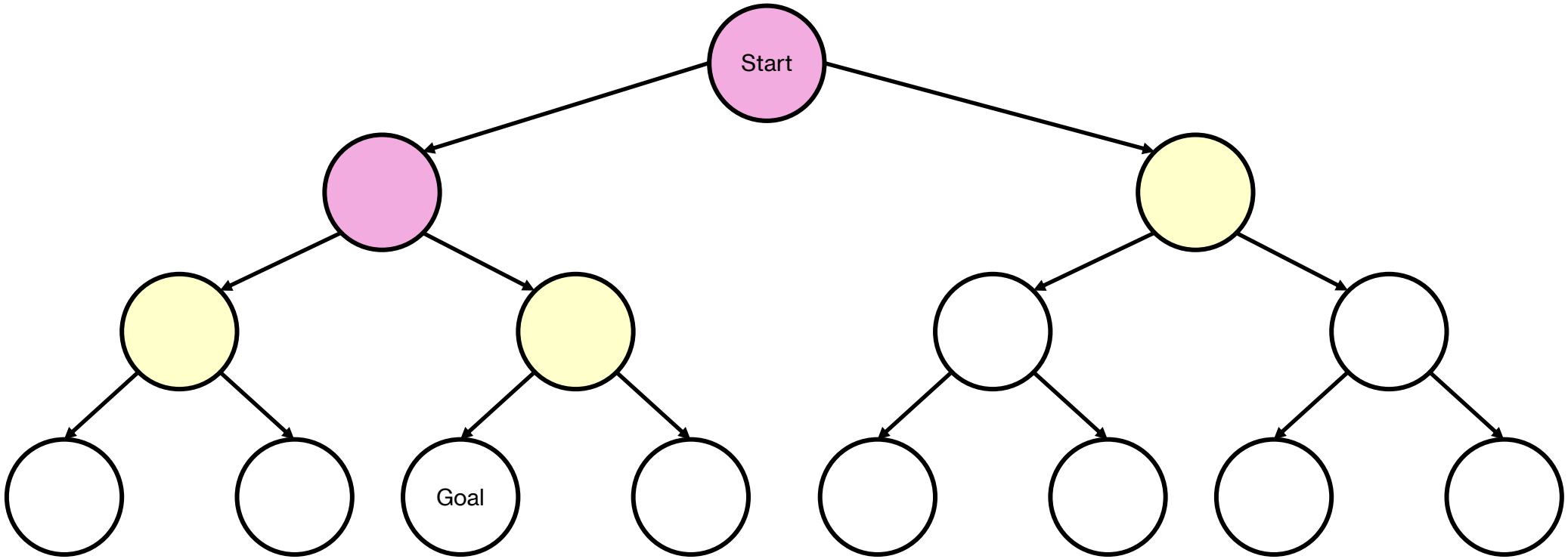
- Explored
- Frontier

Breadth-First Search



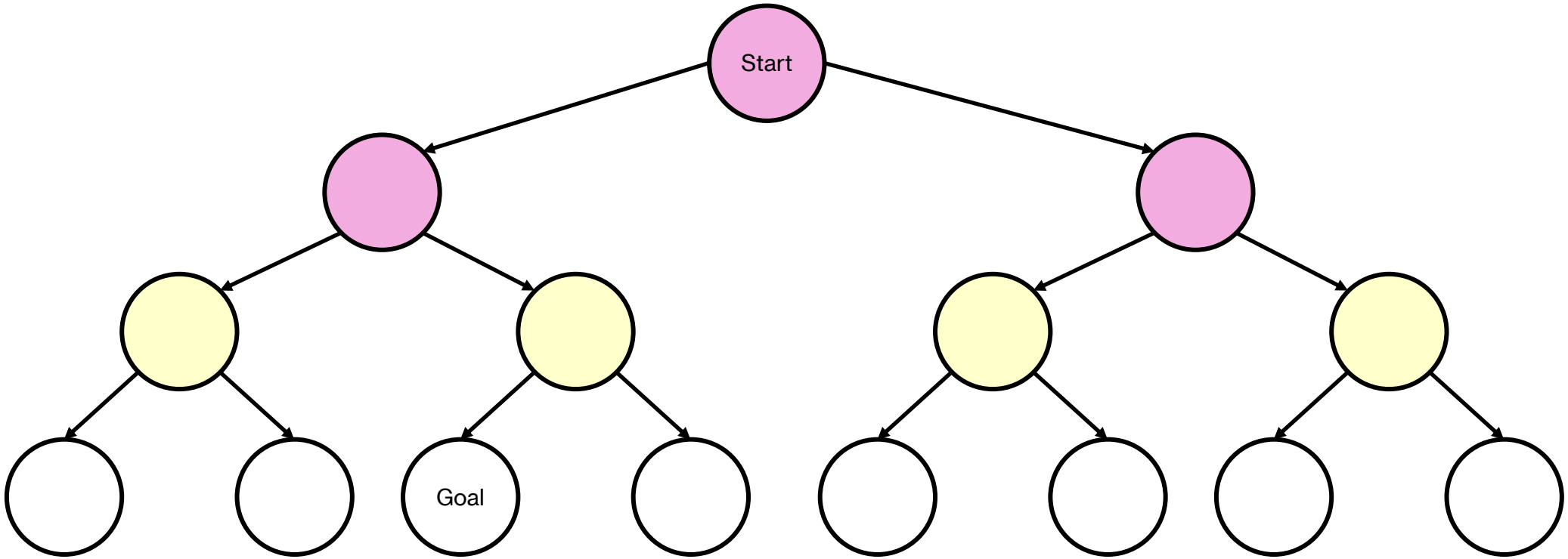
- Explored
- Frontier

Breadth-First Search



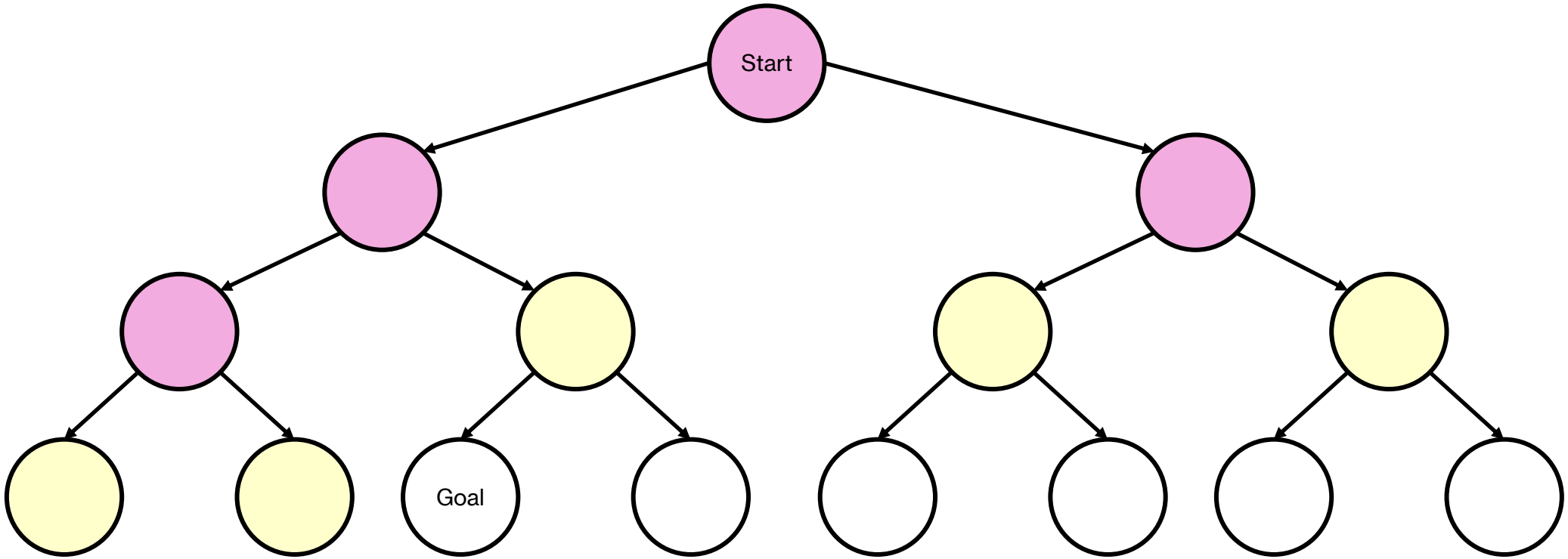
- Explored
- Frontier

Breadth-First Search



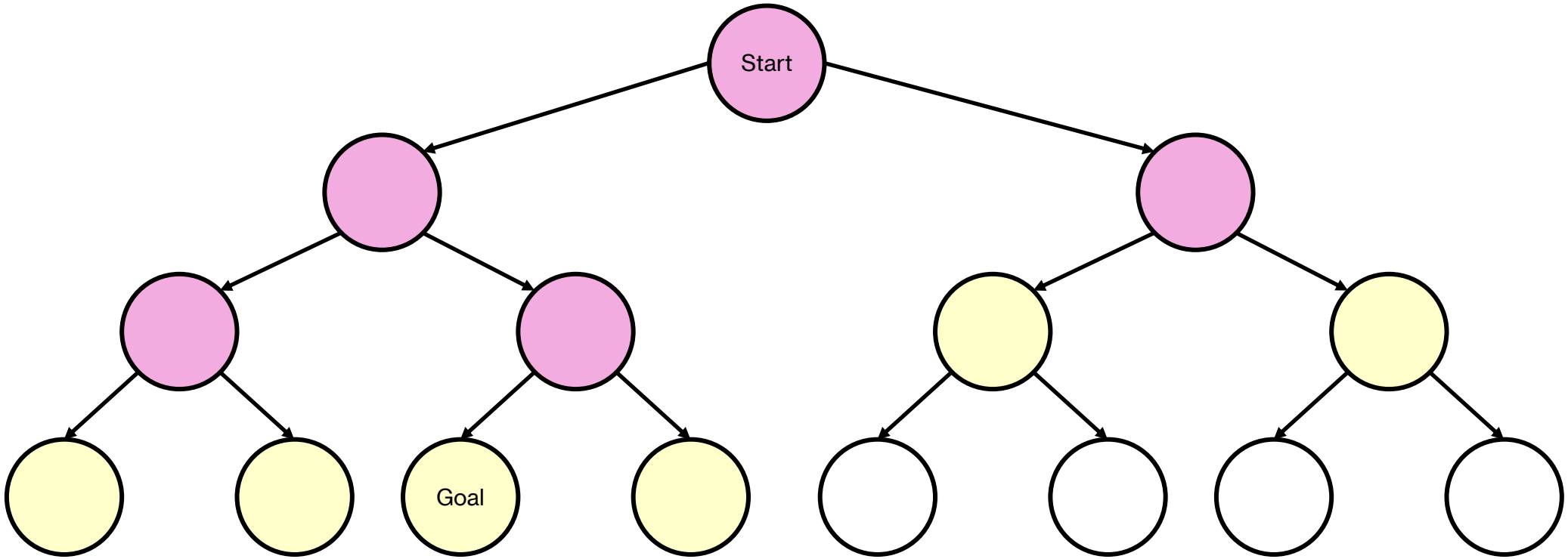
- Explored
- Frontier

Breadth-First Search



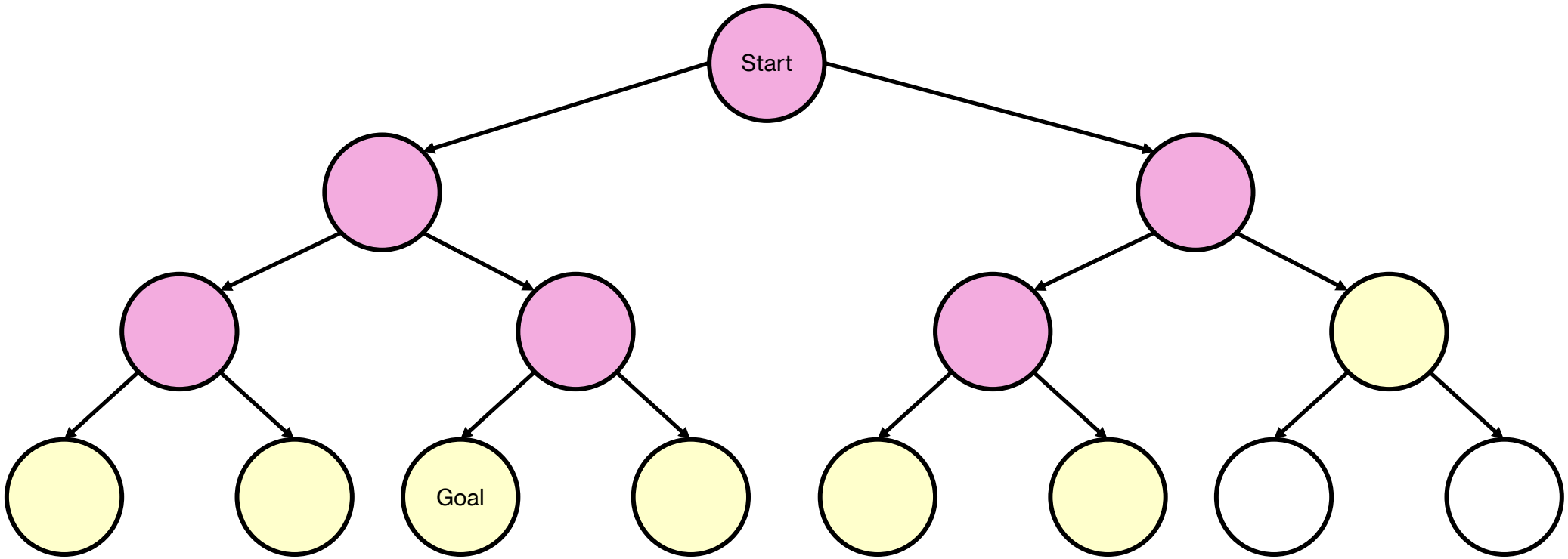
- Explored
- Frontier

Breadth-First Search



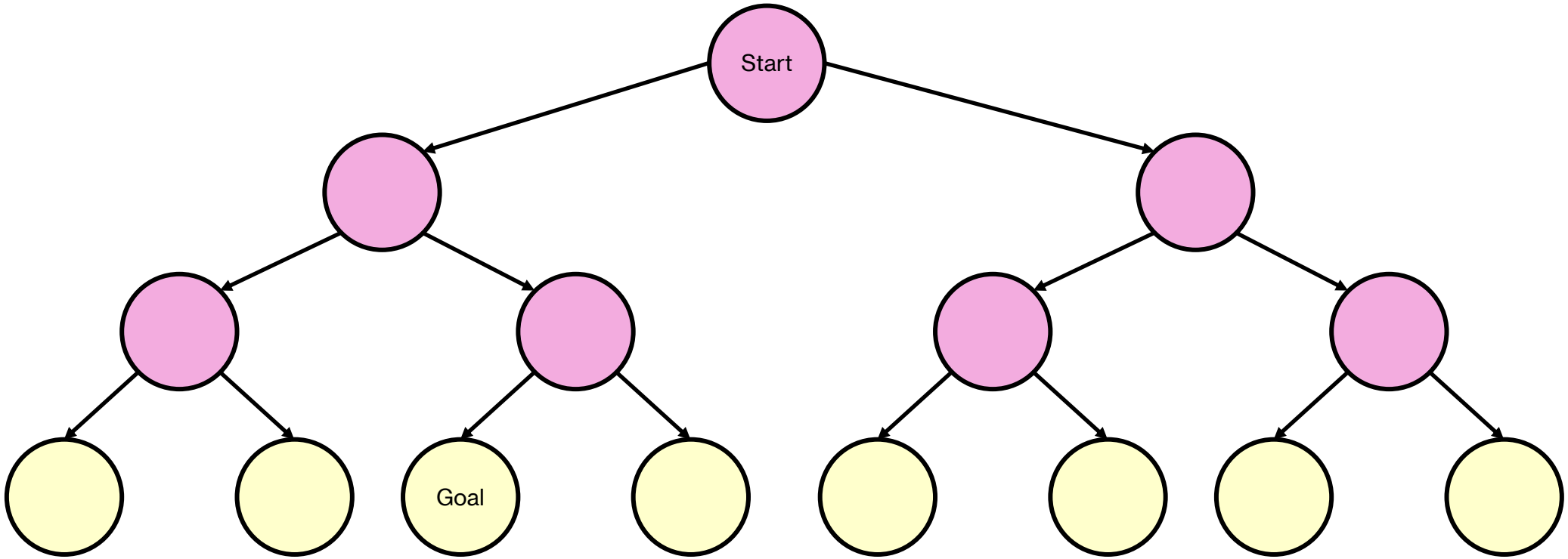
- Explored
- Frontier

Breadth-First Search



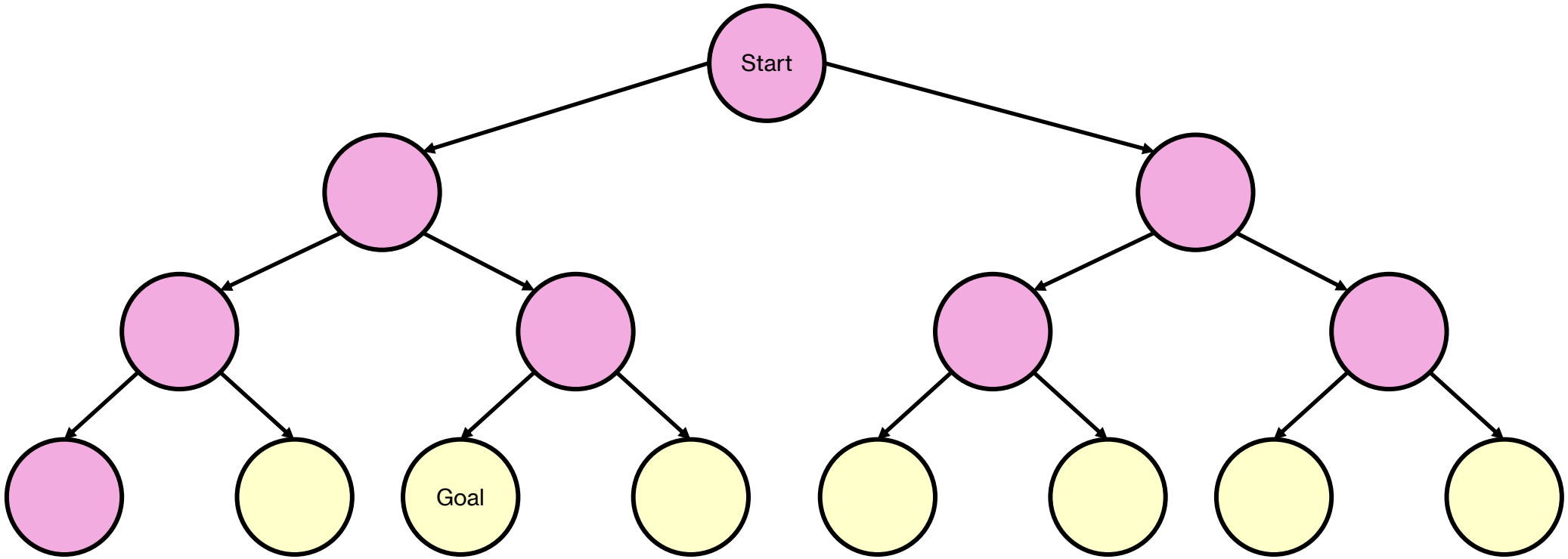
Explored
Frontier

Breadth-First Search



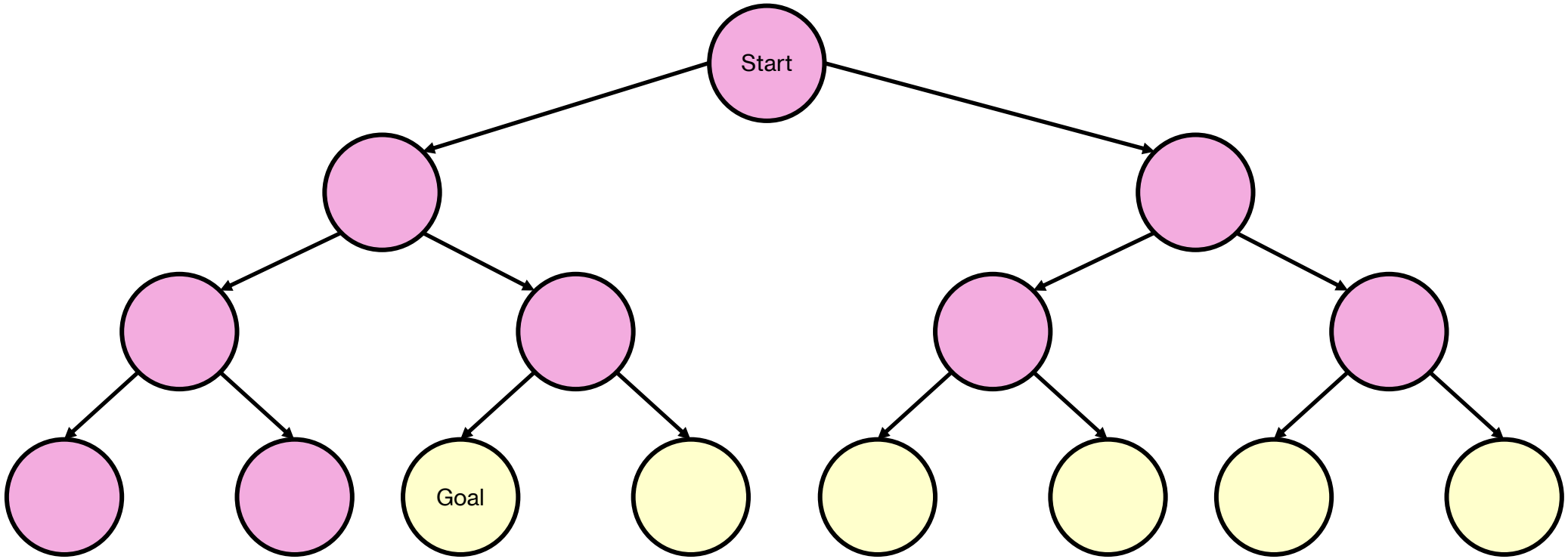
Explored
Frontier

Breadth-First Search



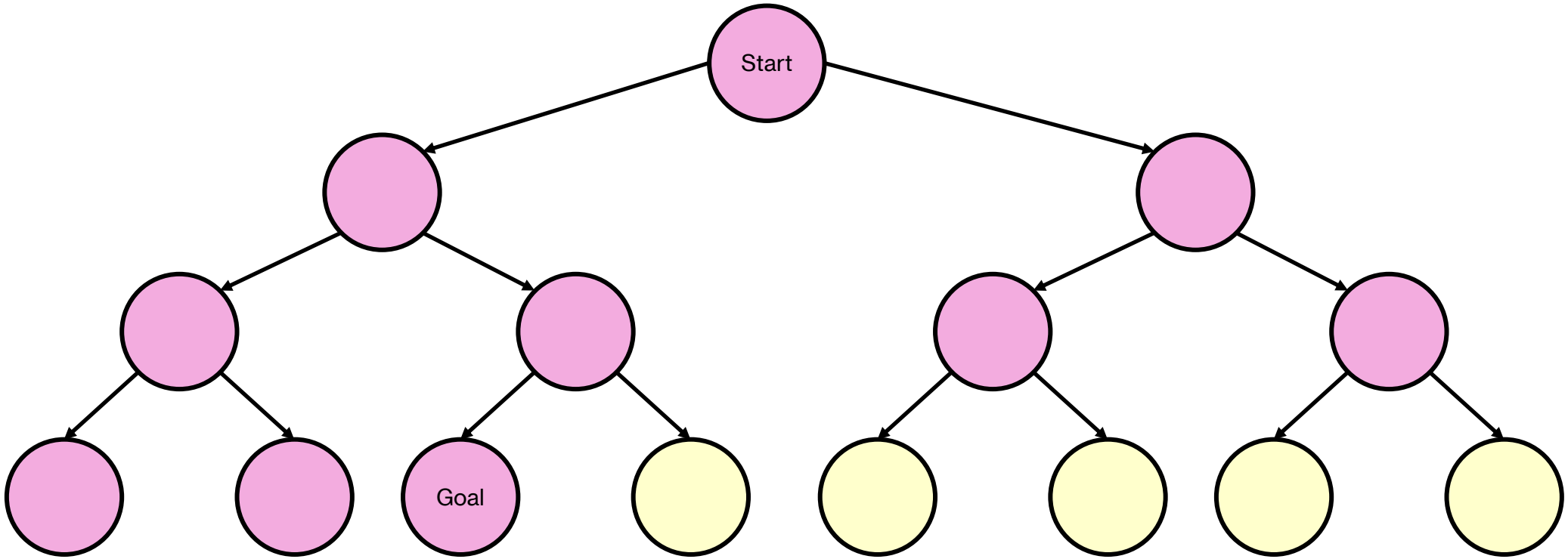
Explored
Frontier

Breadth-First Search



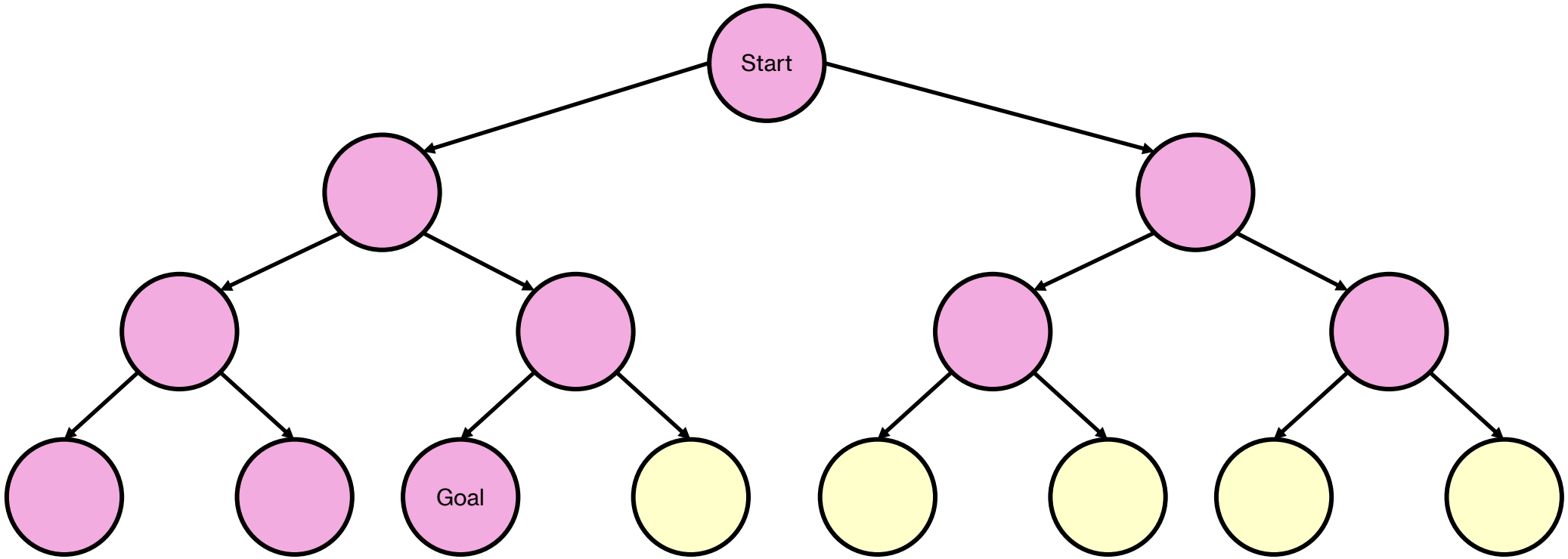
- Explored
- Frontier

Breadth-First Search



Explored
Frontier

Breadth-First Search



Return Solution!

- Explored
- Frontier

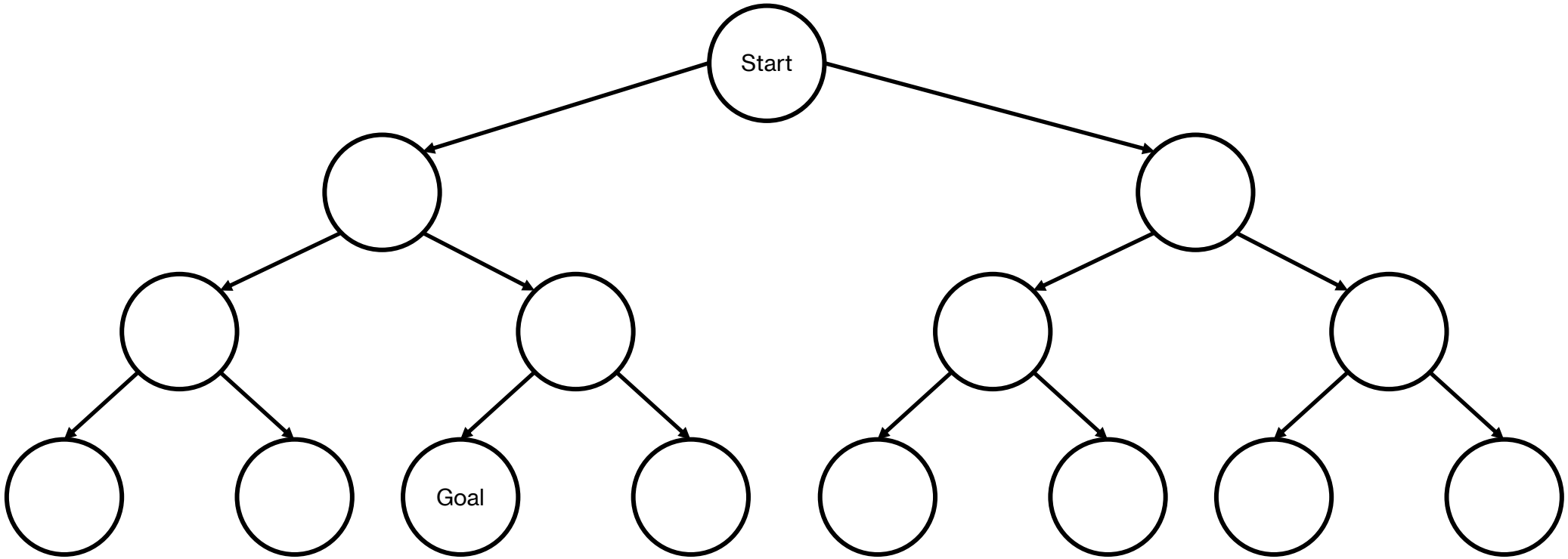
Breadth-First Search Characteristics

- It is **complete** even if the state space is infinite
- It is **optimal** only if the costs are uniform
- Time complexity:$O(b^d)$
- Space complexity: $O(b^d)$

DFS with Iterative Deepening

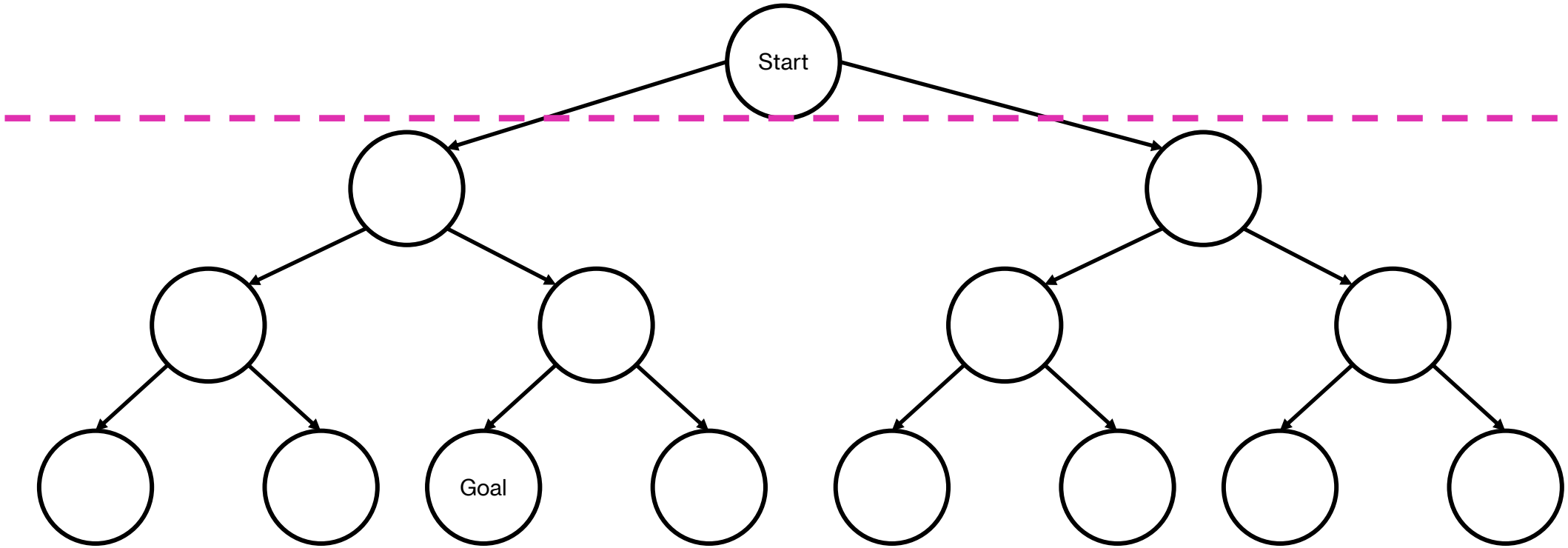
- Call DFS with a depth of limit of 1, 2, 3, ..., D
- Stop when a solution is found

DFS-Iterative Deepening



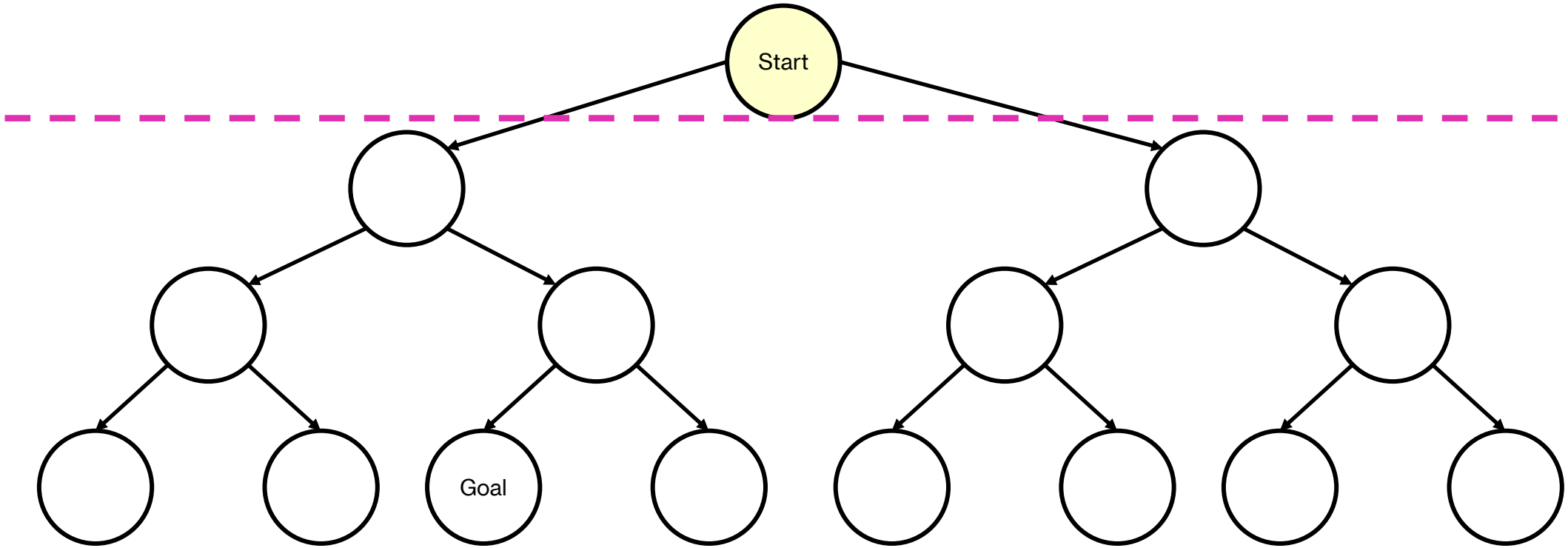
- Explored
- Frontier

DFS-Iterative Deepening



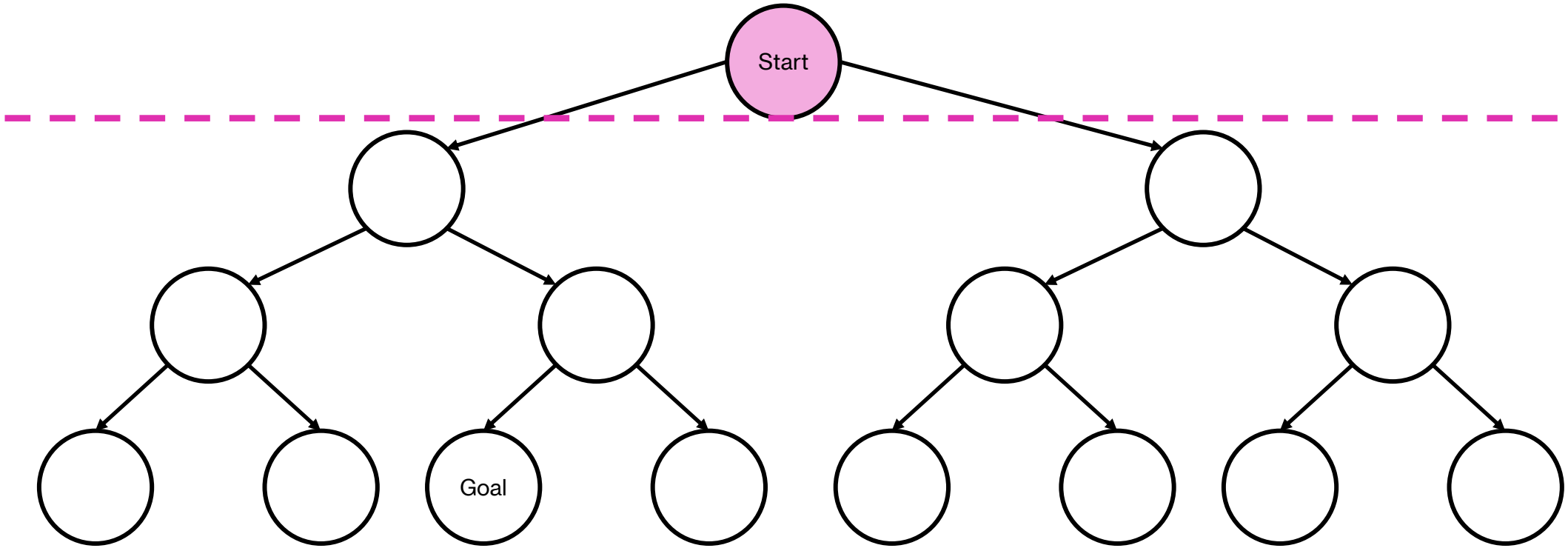
- Explored
- Frontier

DFS-Iterative Deepening



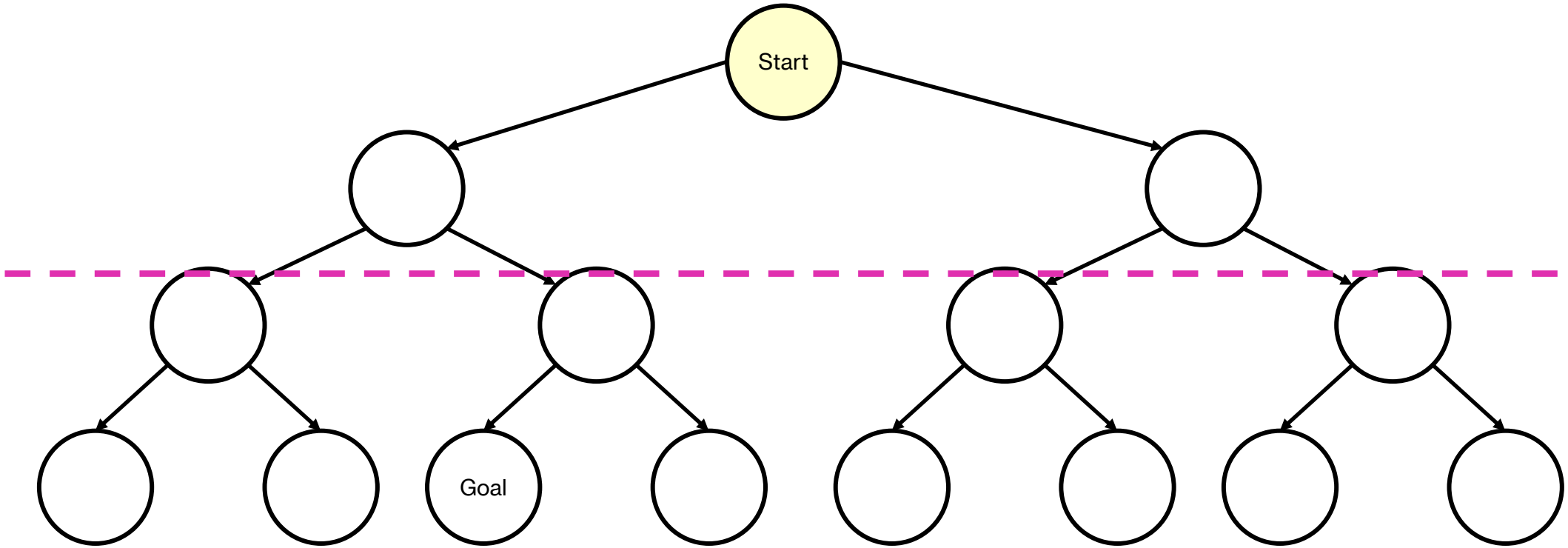
- Explored
- Frontier

DFS-Iterative Deepening



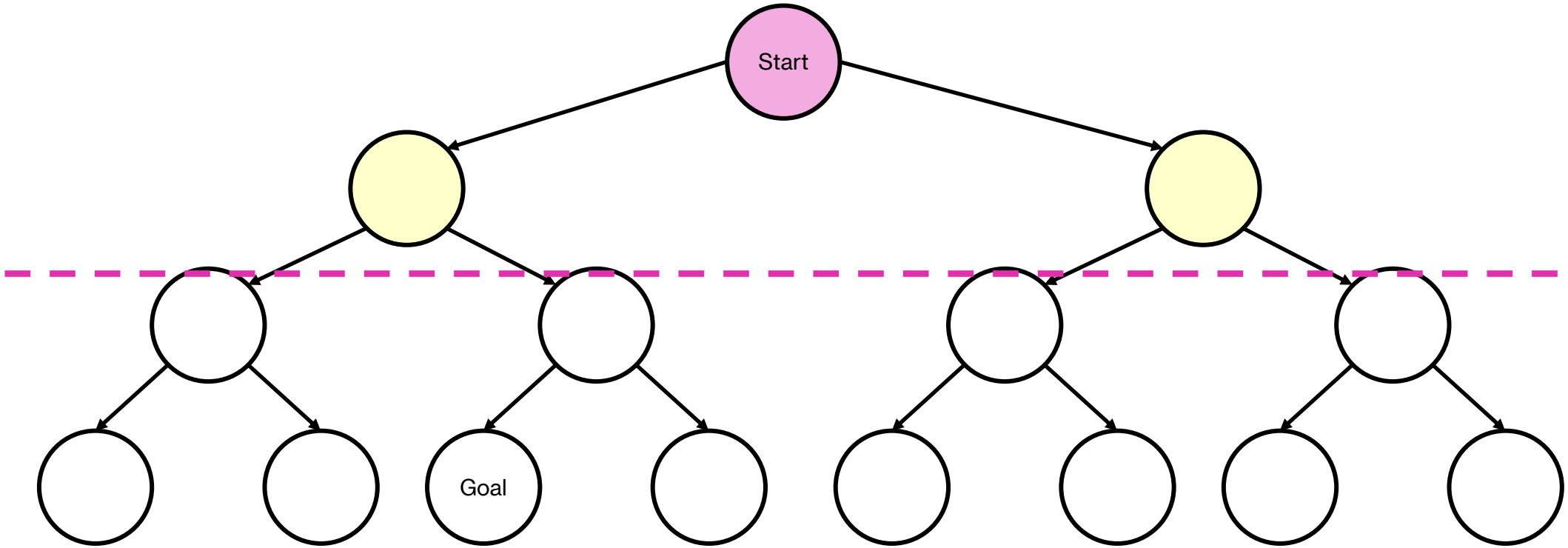
- Explored
- Frontier

DFS-Iterative Deepening



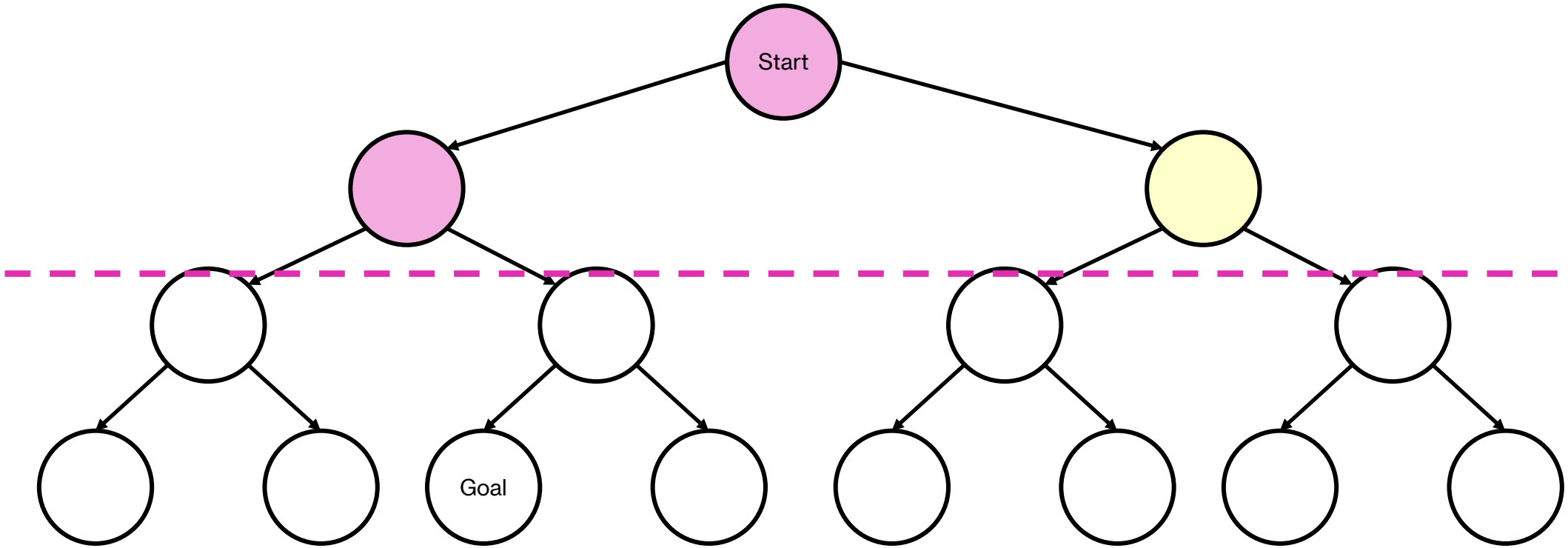
- Explored
- Frontier

DFS-Iterative Deepening

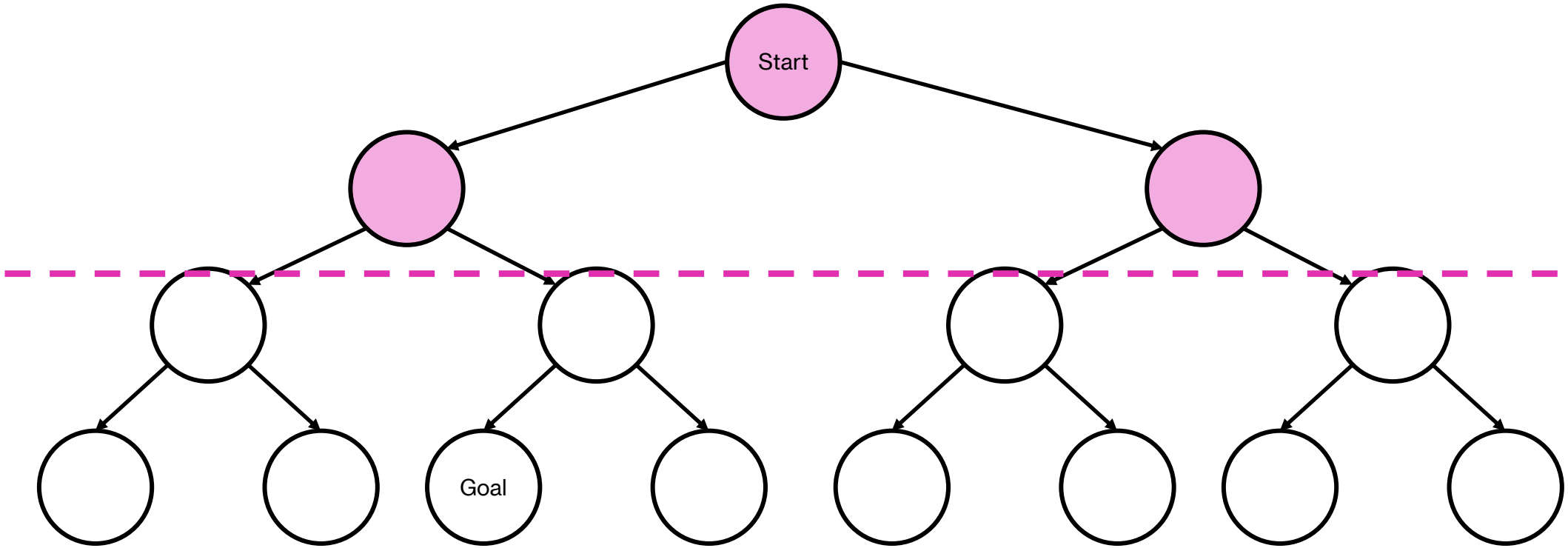


- Explored
- Frontier

DFS-Iterative Deepening

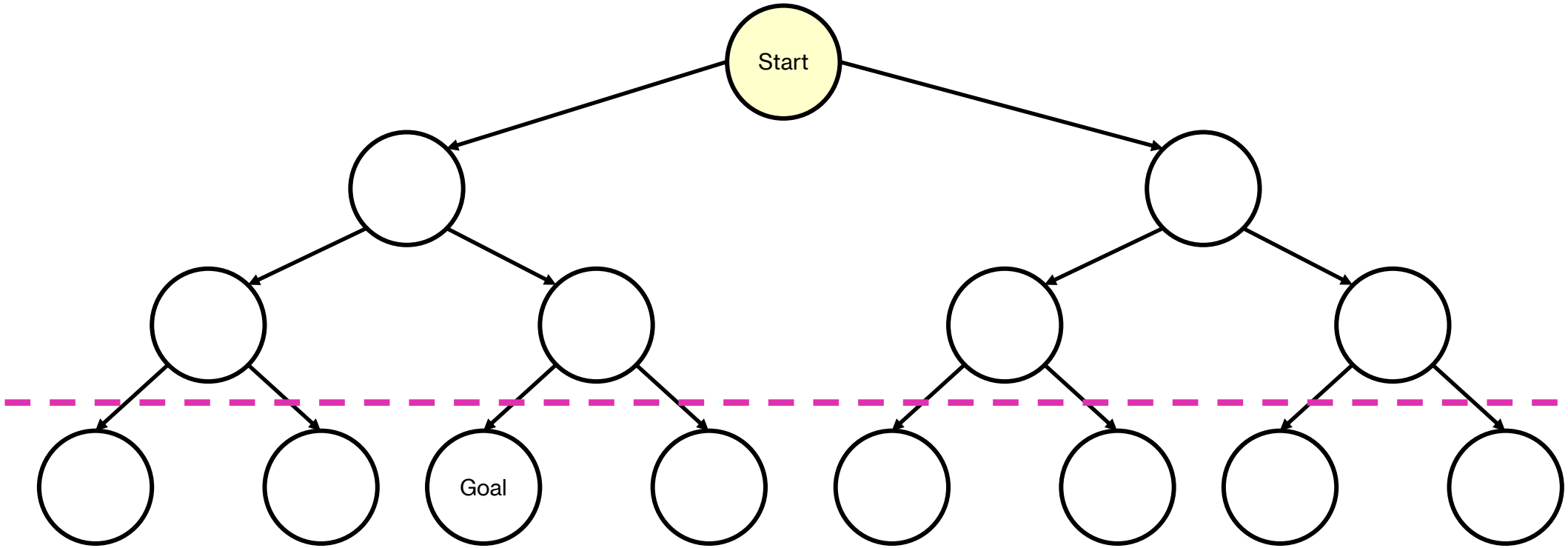


DFS-Iterative Deepening



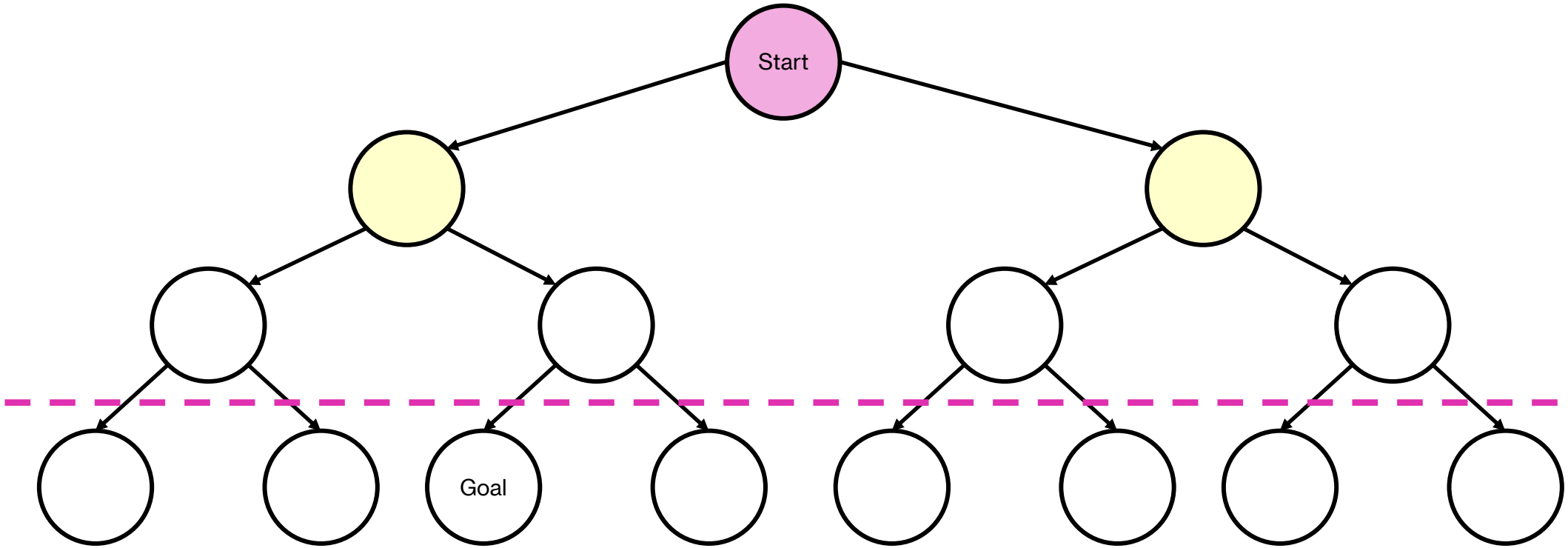
Explored
Frontier

DFS-Iterative Deepening

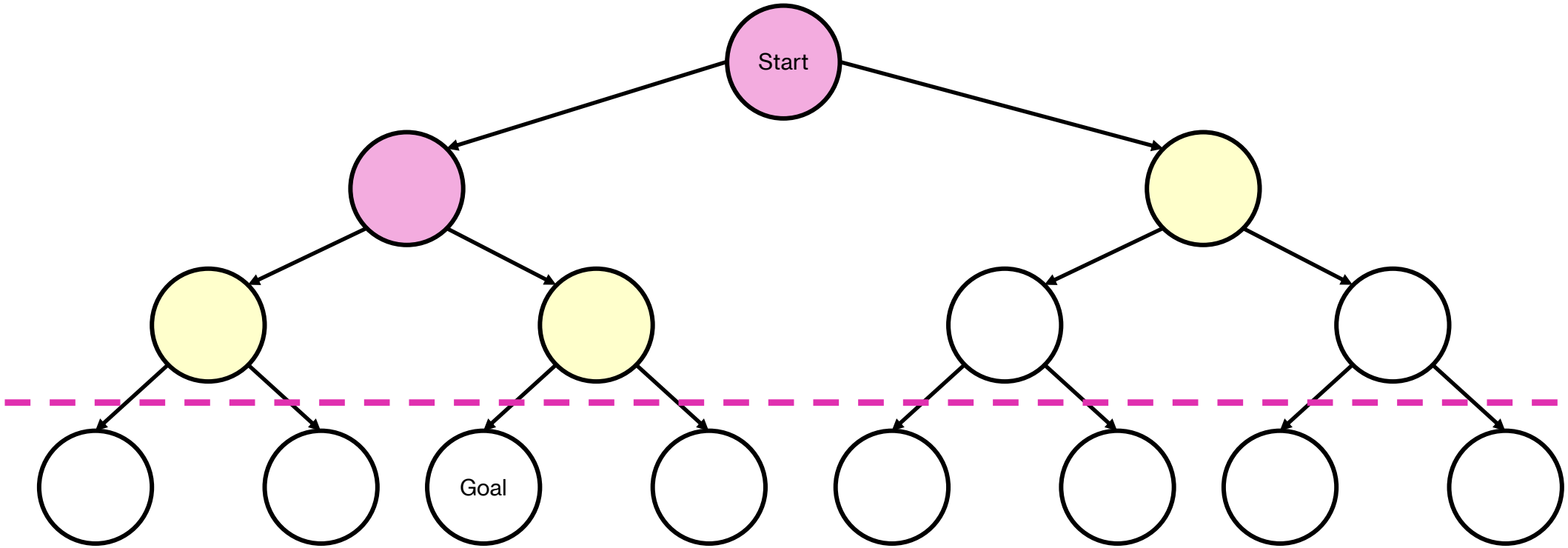


- Explored
- Frontier

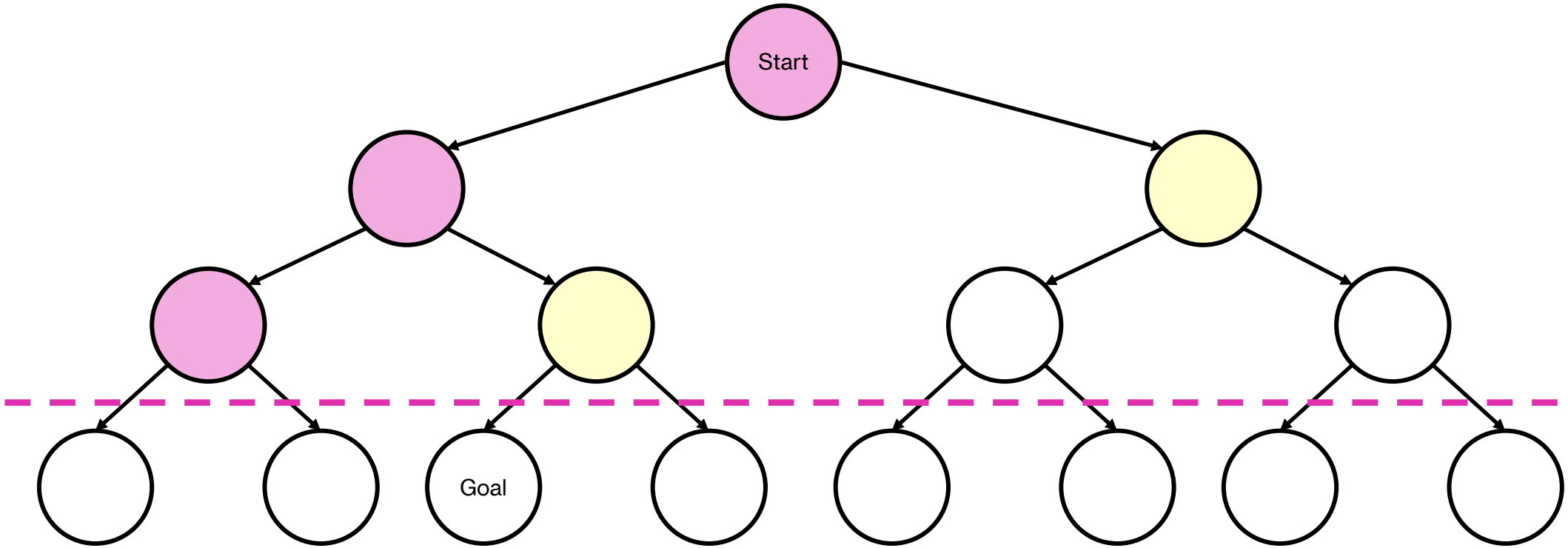
DFS-Iterative Deepening



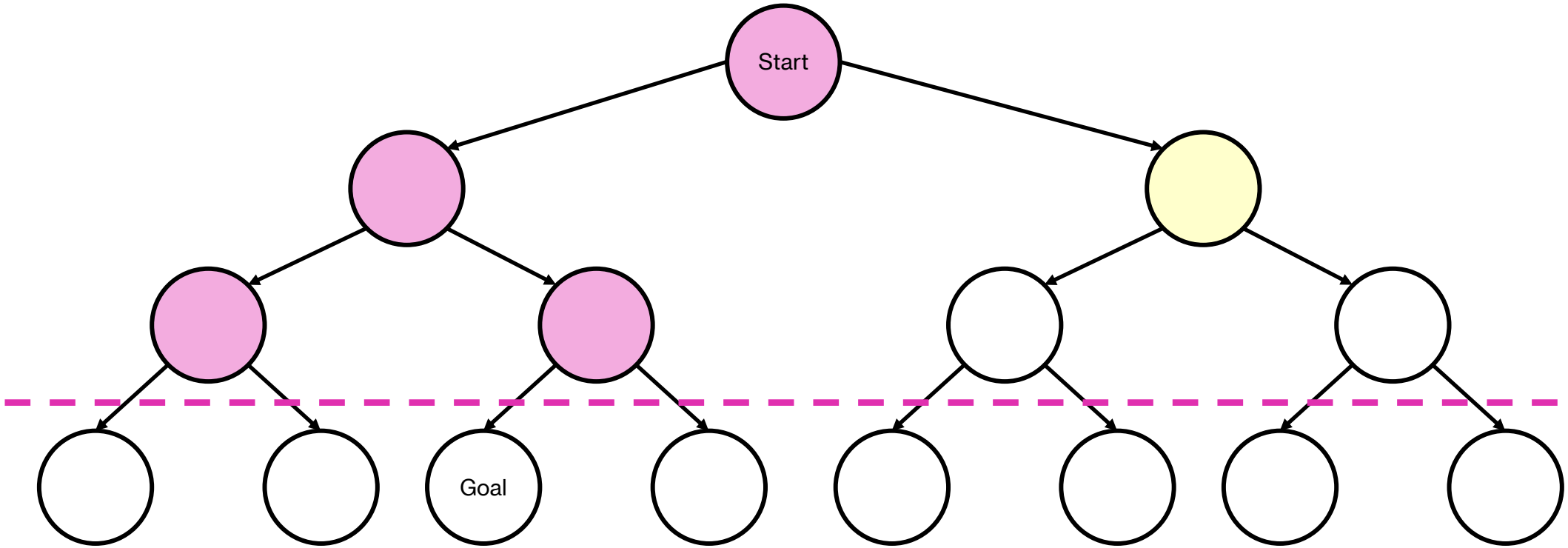
DFS-Iterative Deepening



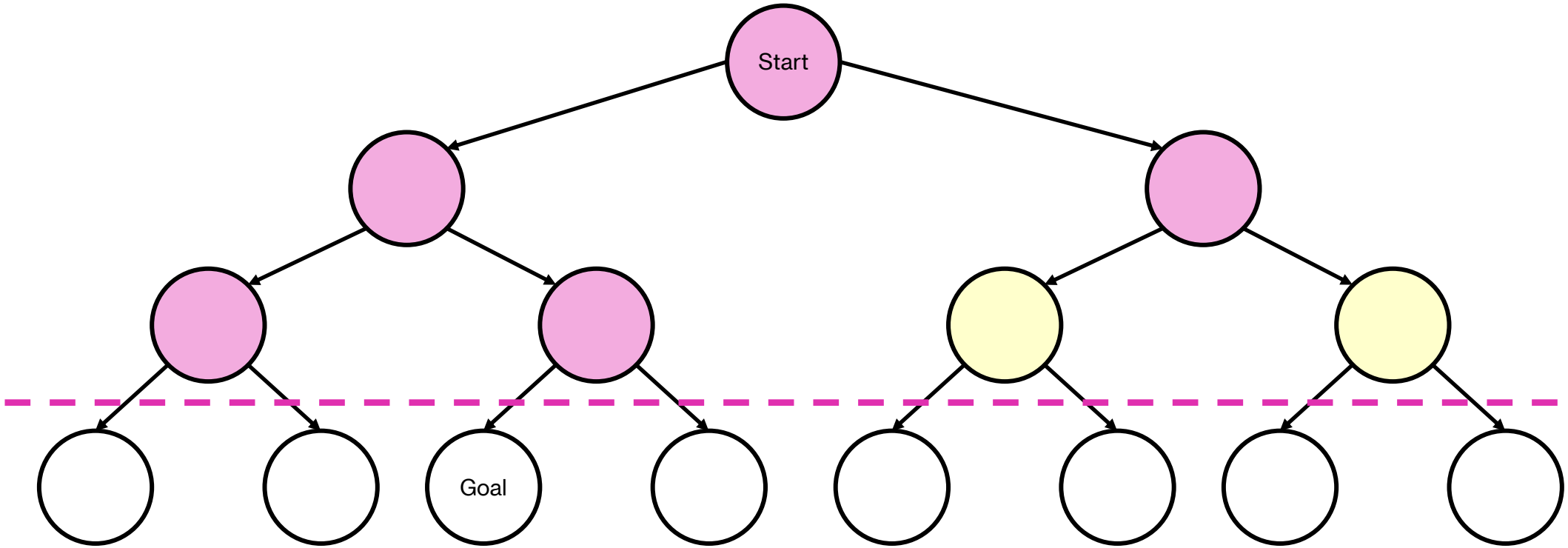
DFS-Iterative Deepening



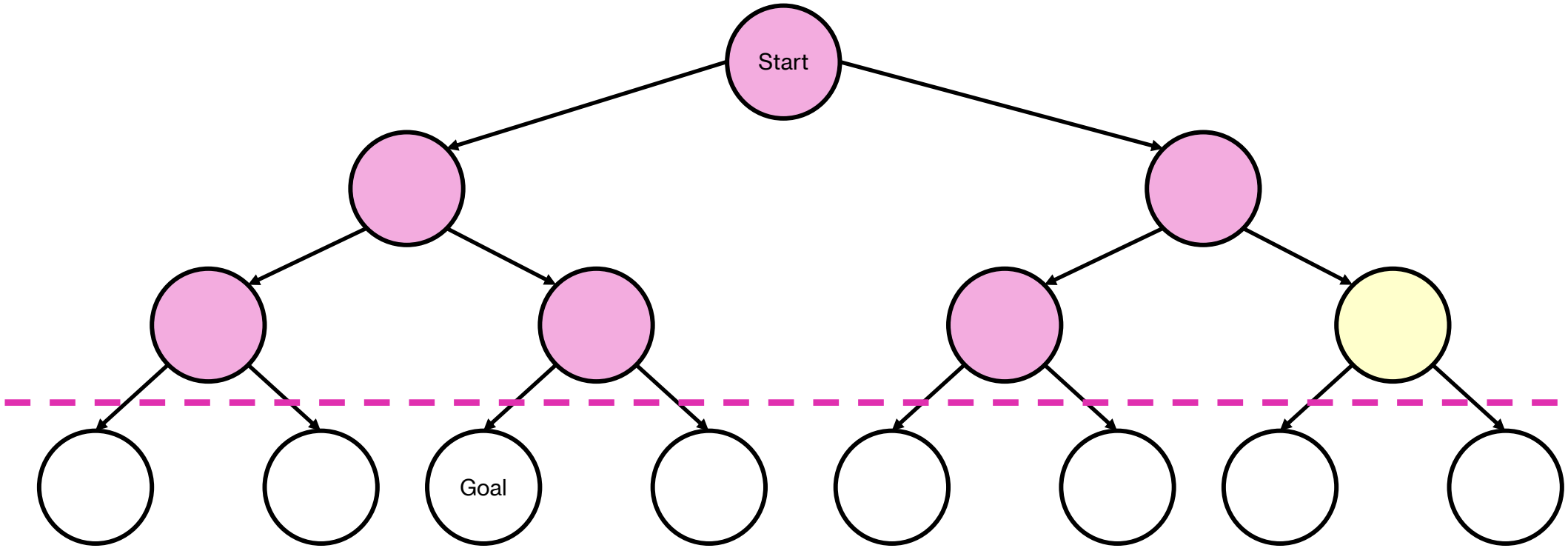
DFS-Iterative Deepening



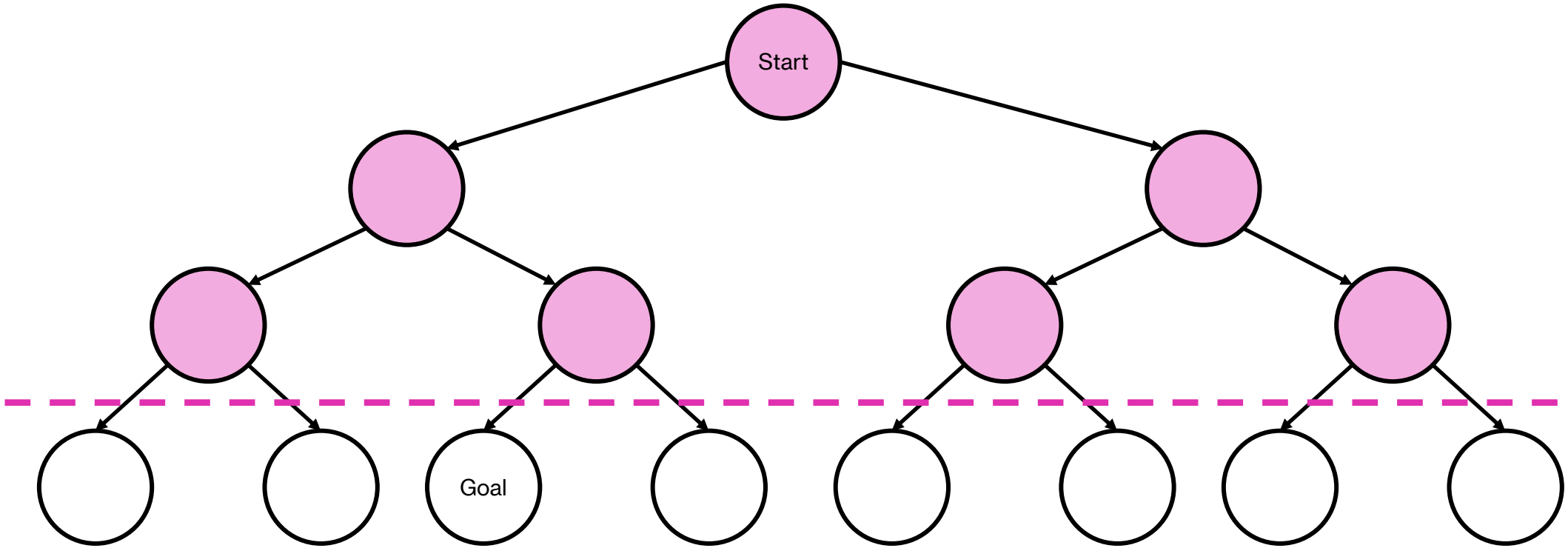
DFS-Iterative Deepening



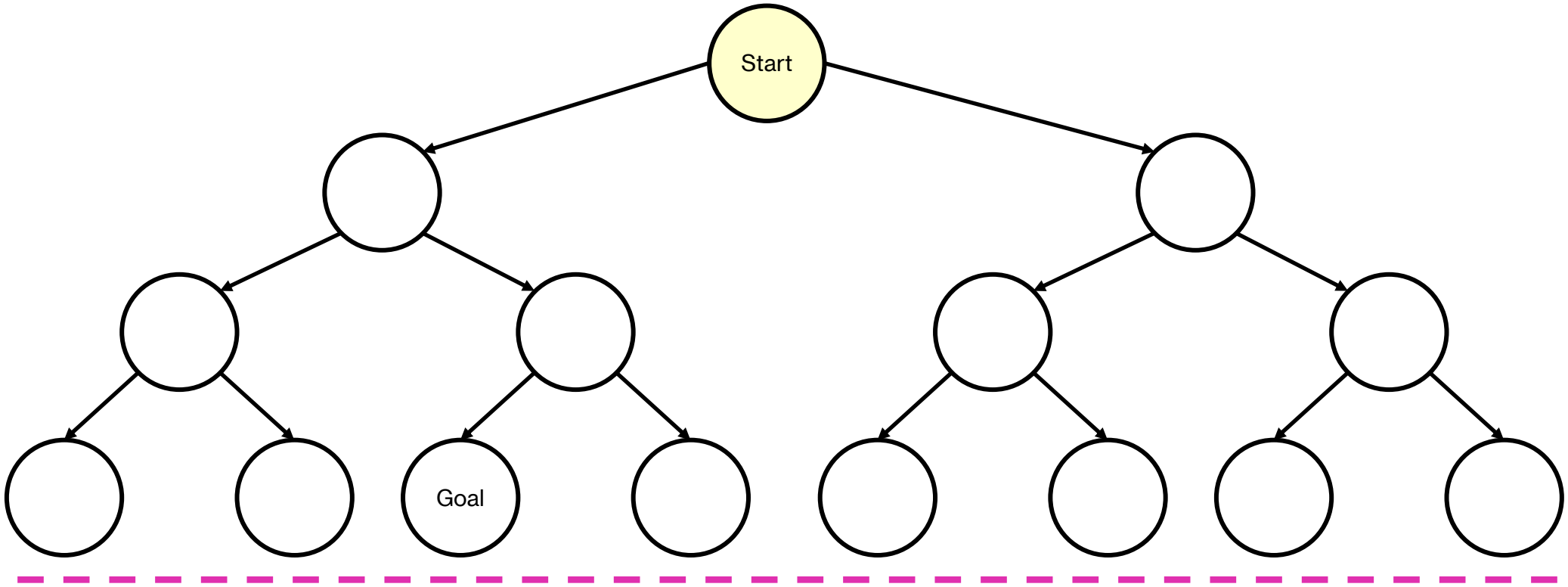
DFS-Iterative Deepening



DFS-Iterative Deepening

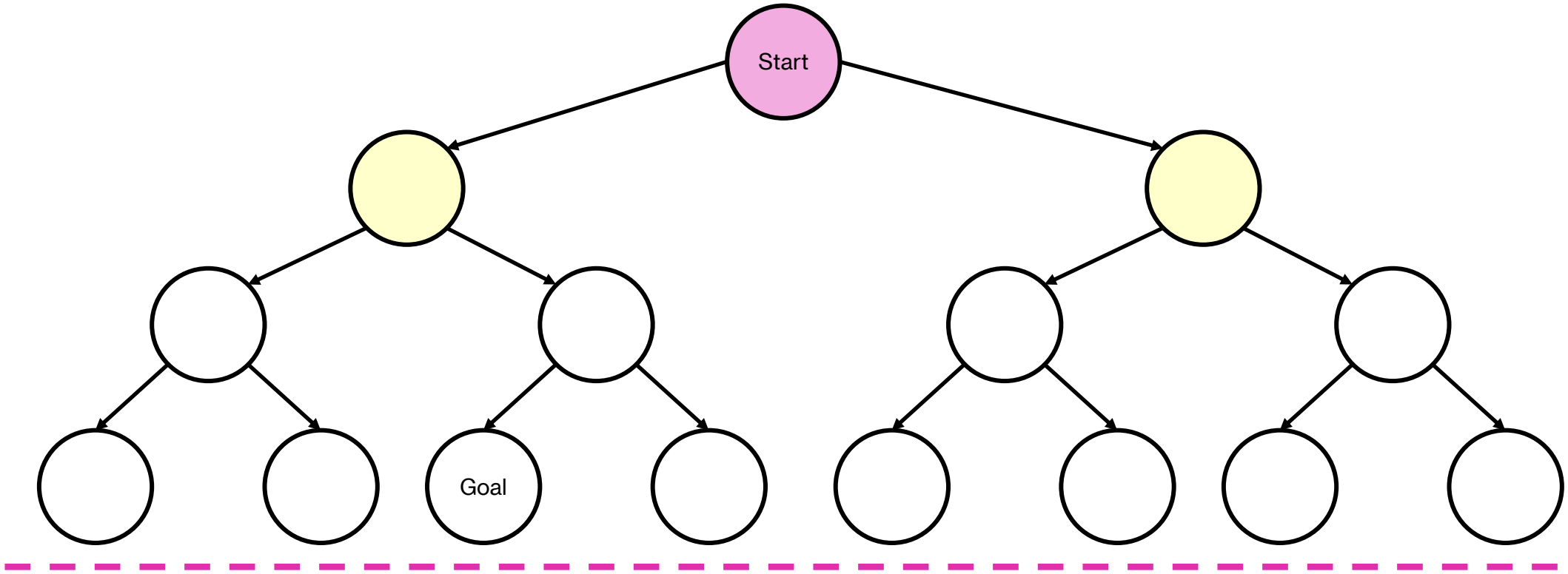


DFS-Iterative Deepening

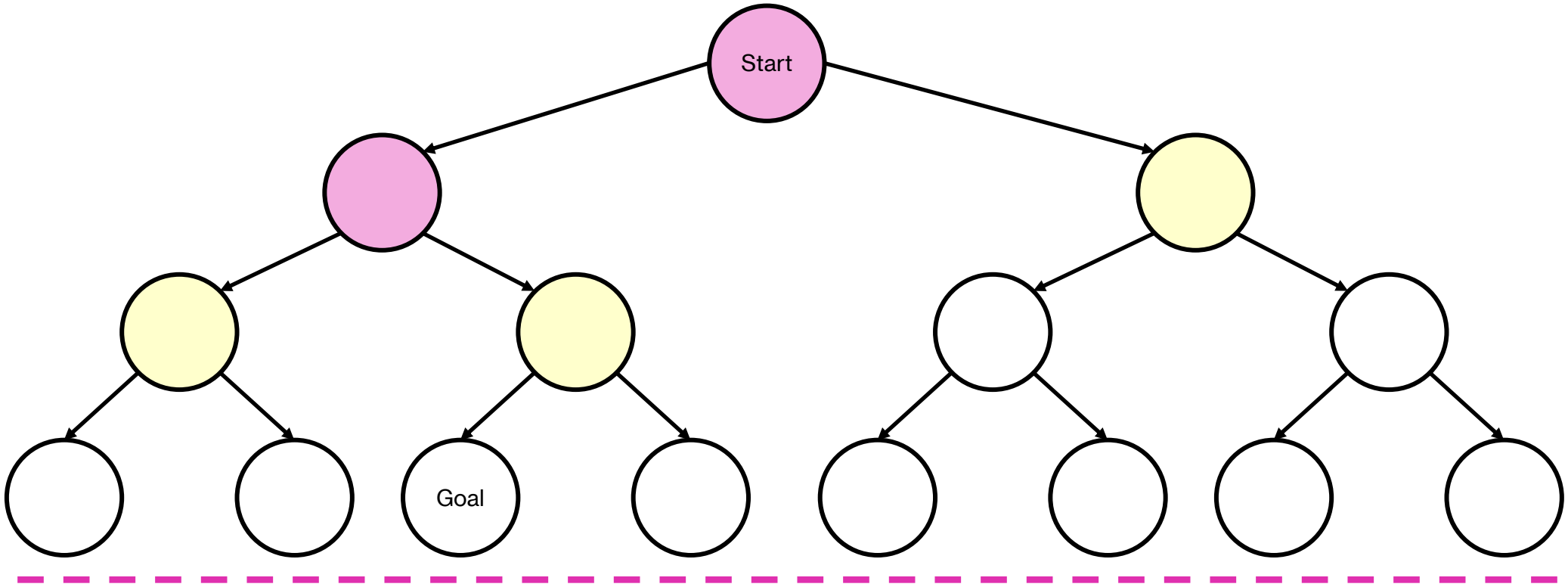


Explored
Frontier

DFS-Iterative Deepening

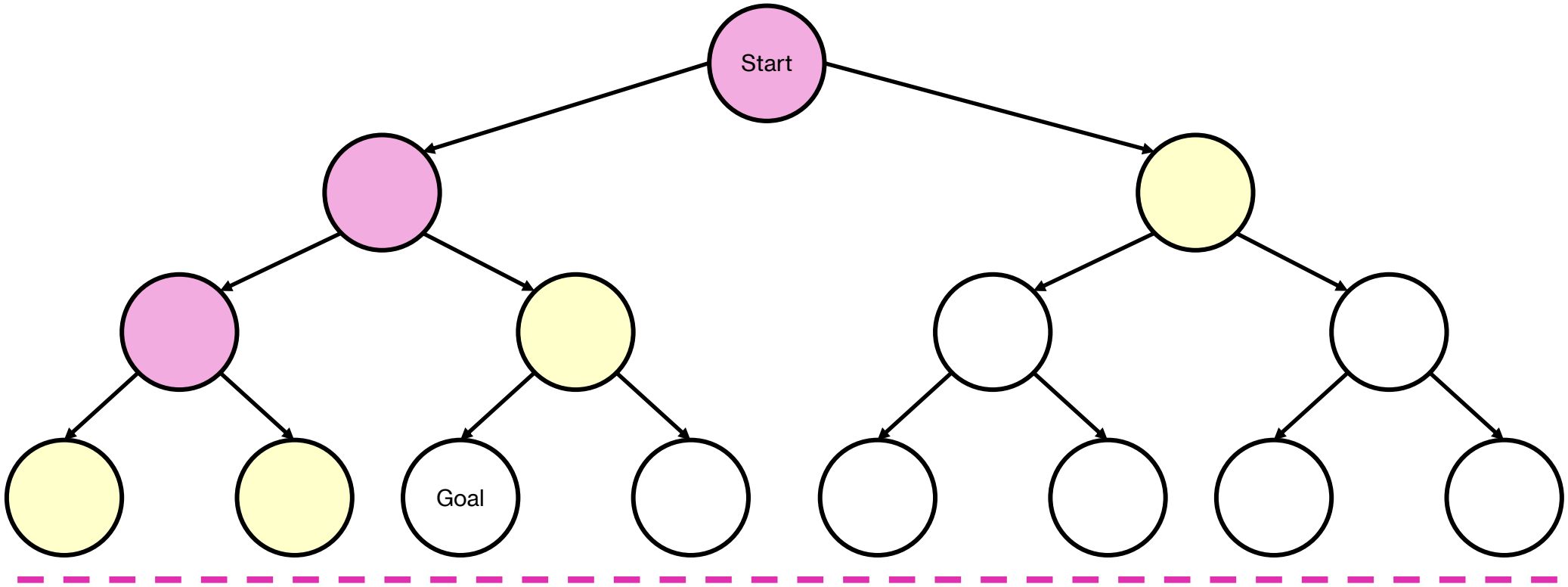


DFS-Iterative Deepening

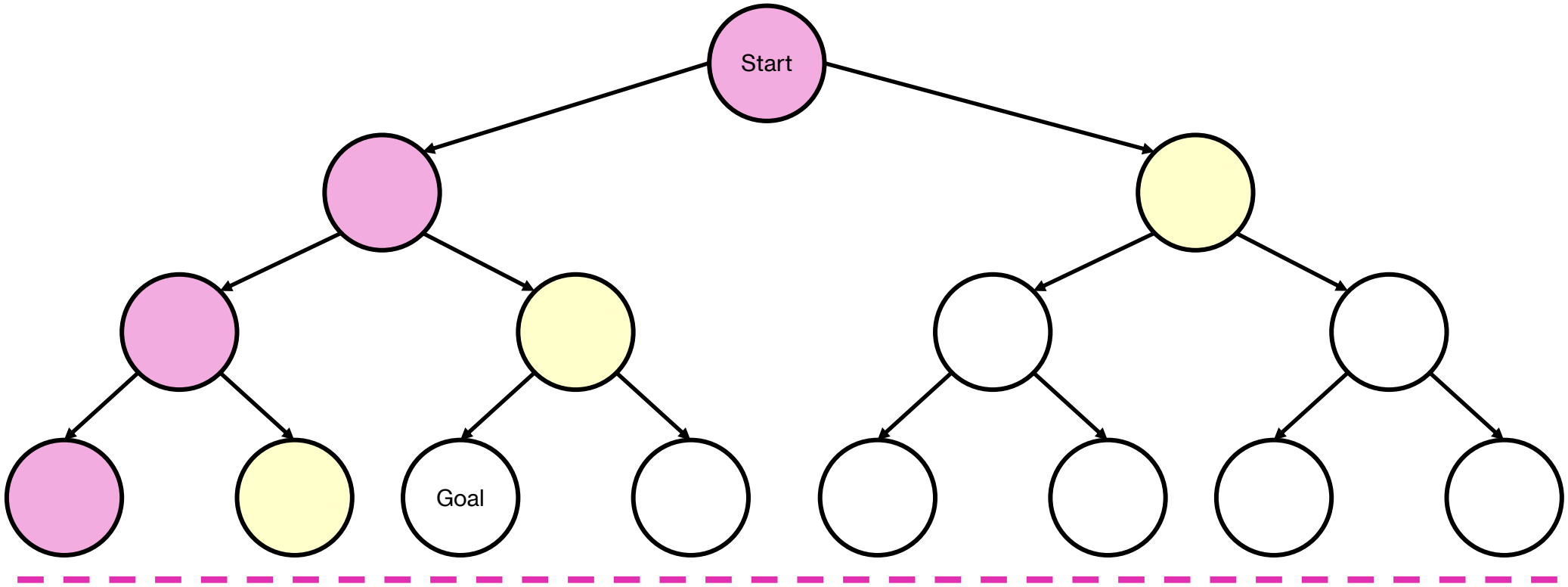


Explored
Frontier

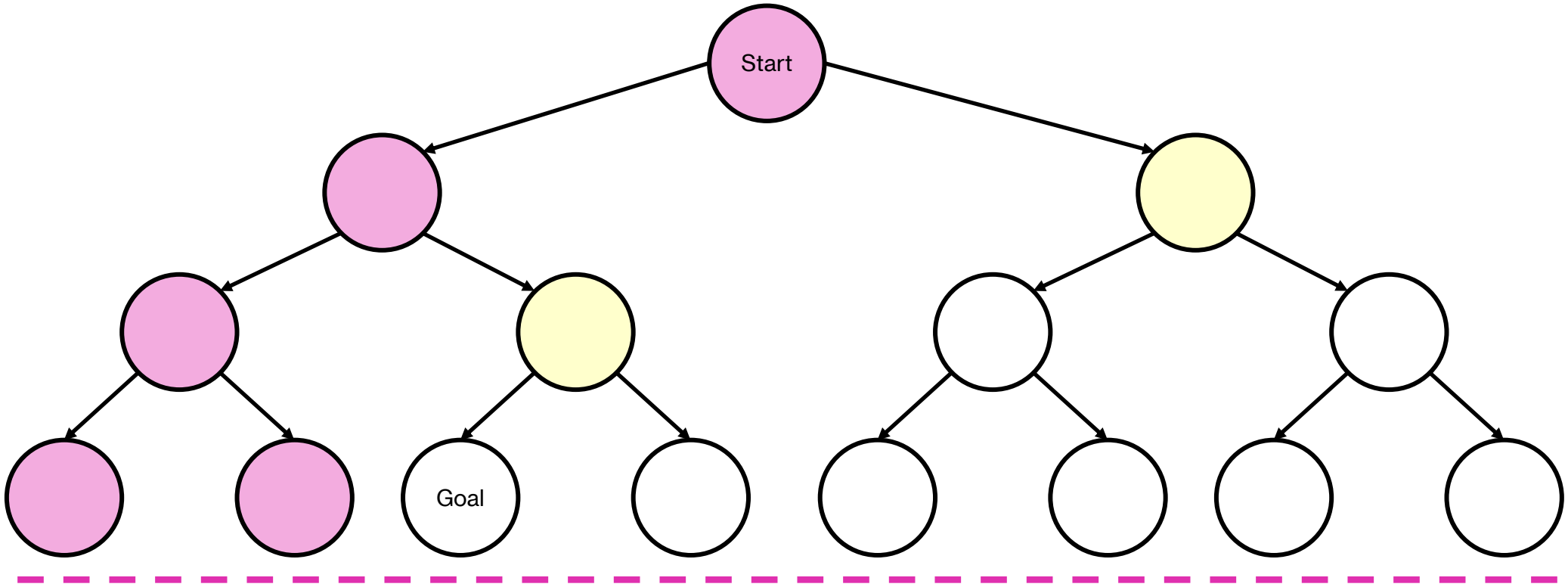
DFS-Iterative Deepening



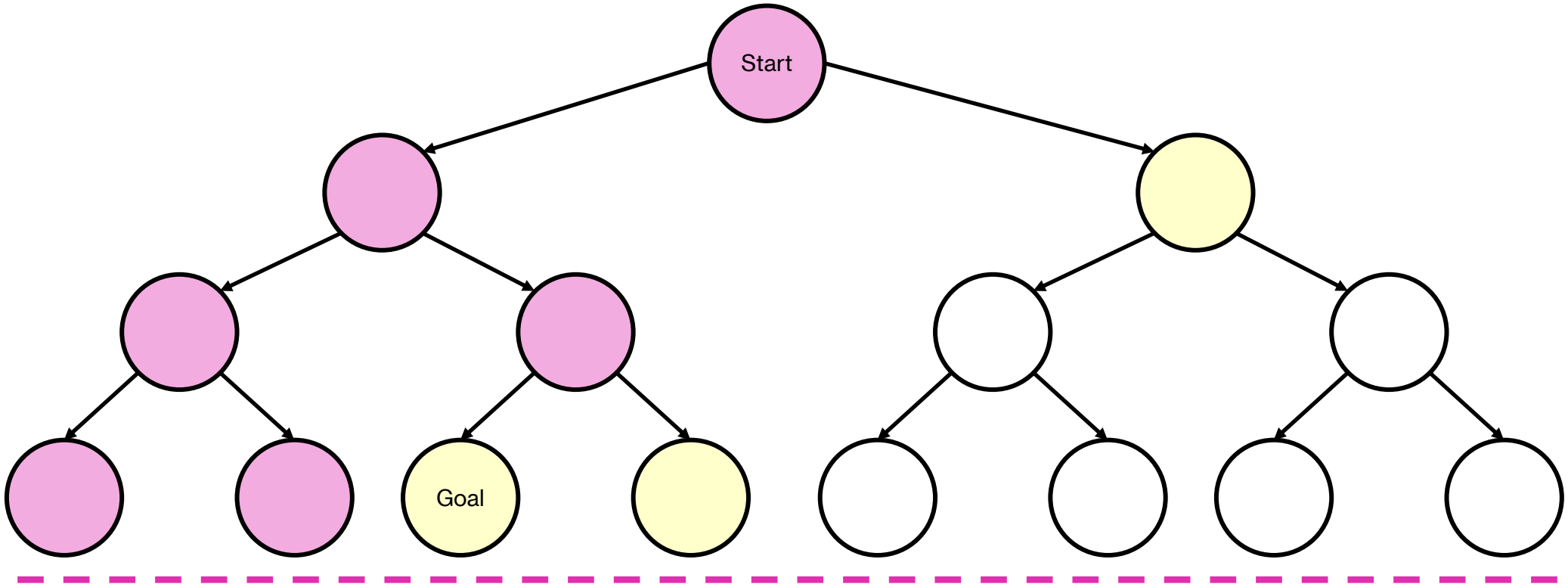
DFS-Iterative Deepening



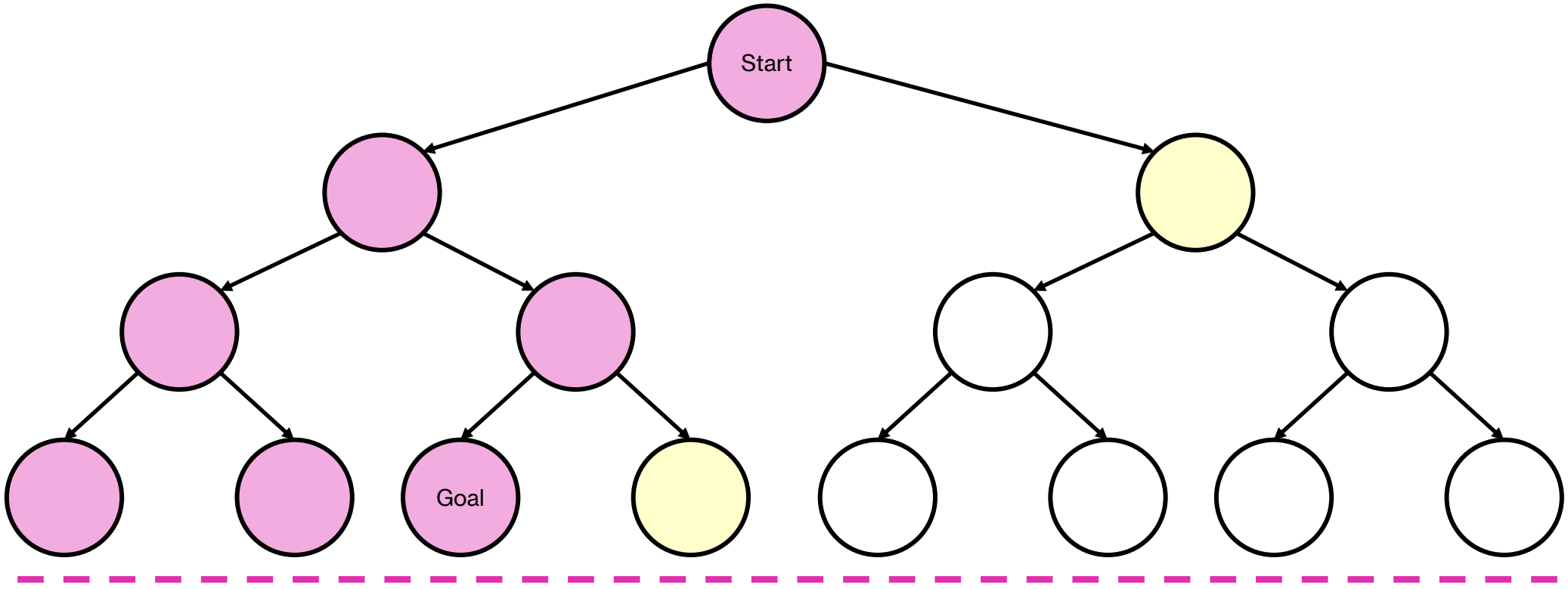
DFS-Iterative Deepening



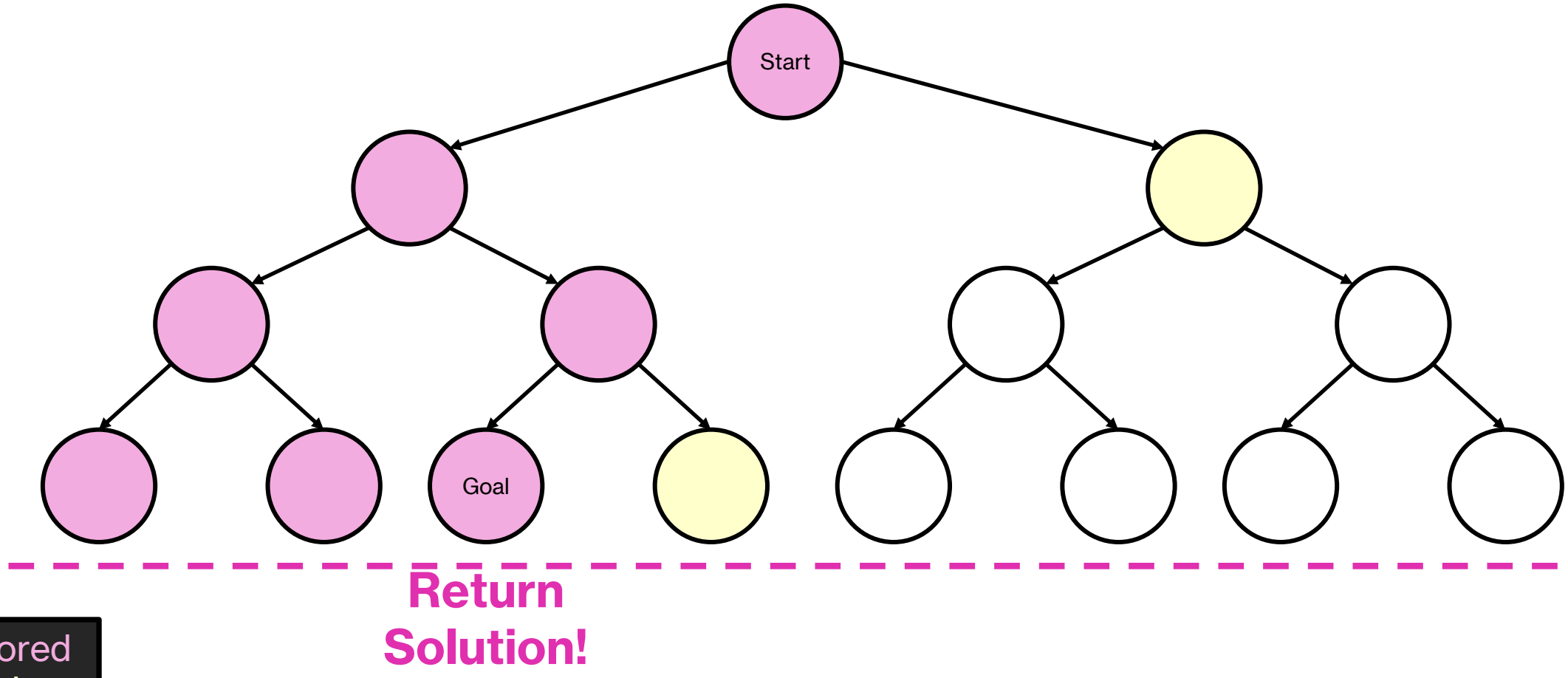
DFS-Iterative Deepening



DFS-Iterative Deepening



DFS-Iterative Deepening



DFS-Iterative Deepening Characteristics

- It is **complete** even if the state space is infinite
- It is **optimal** only if the costs are uniform
- Time complexity:$O(b^d)$
- Space complexity: $O(d)$

Tree Search Algorithms Summary

Algorithm	Optimal when action costs are...	Time Complexity	Space Complexity
backtracking search	any	$O(b^D)$	$O(D)$
DFS	all 0 (irrelevant)	$O(b^D)$	$O(D)$
BFS	constant ≥ 0	$O(b^d)$	$O(b^d)$
DFS-ID	constant ≥ 0	$O(b^d)$	$O(d)$

Acknowledgments

- Stanford University CS221 Autumn 2021 course. Available online at: https://stanford-cs221.github.io/autumn2021
- Previous CSINTSY slides by the following instructors:
  - Raymund Sison, PhD
  - Judith Azcarraga, PhD
  - Merlin Suarez, PhD
  - Joanna Pauline Rivera

Readings

- https://www.cs.miami.edu/home/geoff/Courses/COMP6210-10M/Content/StateSpaceSearch.shtml
- https://www.geeksforgeeks.org/search-algorithms-in-ai/
- https://www.educative.io/answers/what-is-uninformed-search-algorithm-in-ai
- https://www.educative.io/answers/what-is-iterative-deepening-search