Agent

Problem-Solving Agent

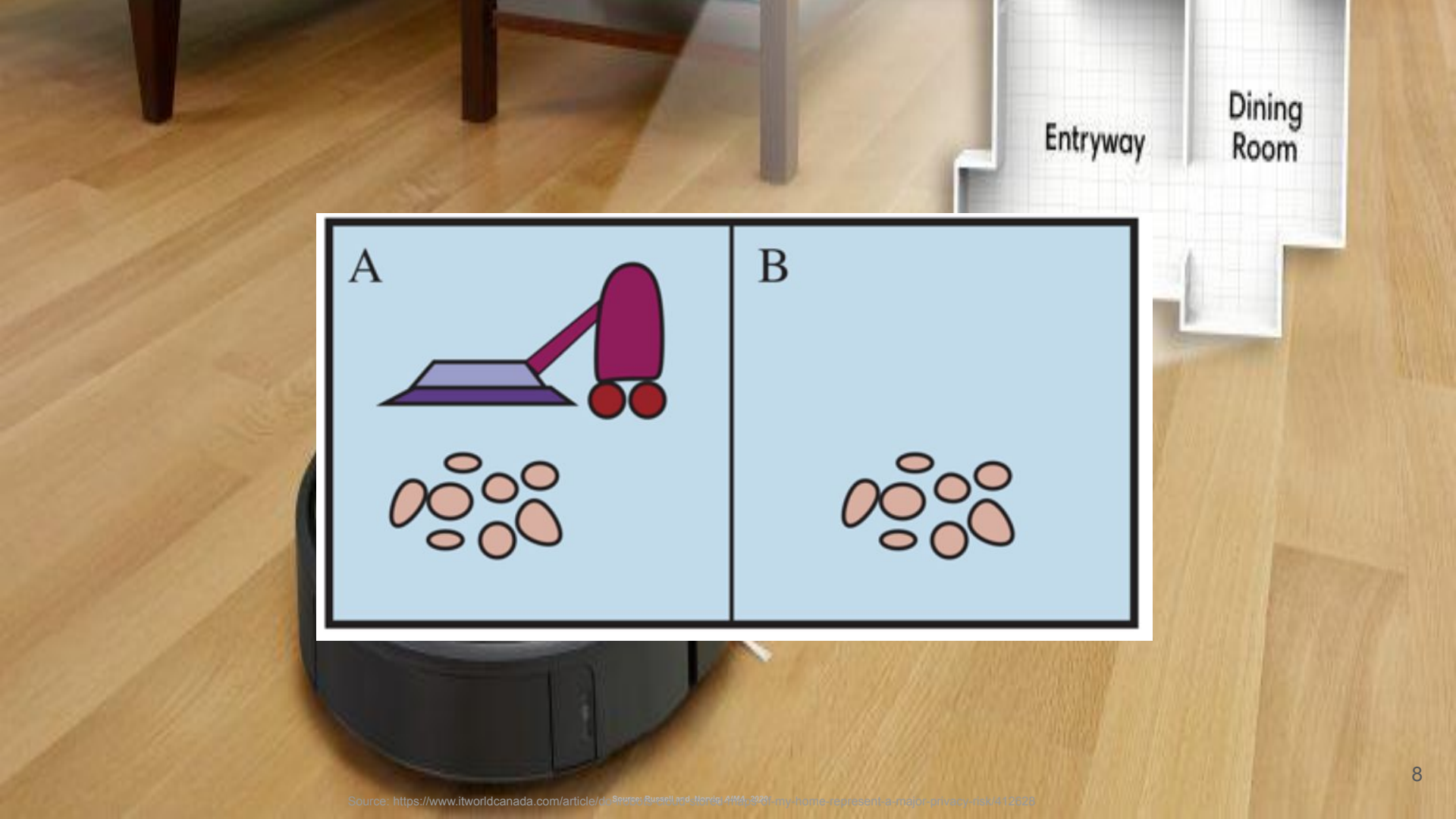

觀光景點
大眾運輸
停車場
藥局
自動提款機
台北捷運公司木柵機廠
魚衡子
抱子腳
動物園
貓纜動物園站
臺北市立動物園
大貓熊館
坡內坑
公館後
文山區
木柵
貓空
106
貓纜動物園南站
國立政治大學
新興
歐比咖啡
炮子崙
小坑
道南河濱公園
指南宮
猴山岳
木柵休息站
大坑
苧子園坑
石坡坑
樟山寺
岐山
貓空山水客景觀茶館
貓空觀光茶園
貓空站
貓空
found.your.tea
地圖資料 ©2023 Google
台灣
御櫻[櫻花林]

Source: https://www.itworldcanada.com/article/do-robots-cloud-detect-maps-of-my-home-represent-a-major-privacy-risk/412628

Goal Formulation

Problem Formulation

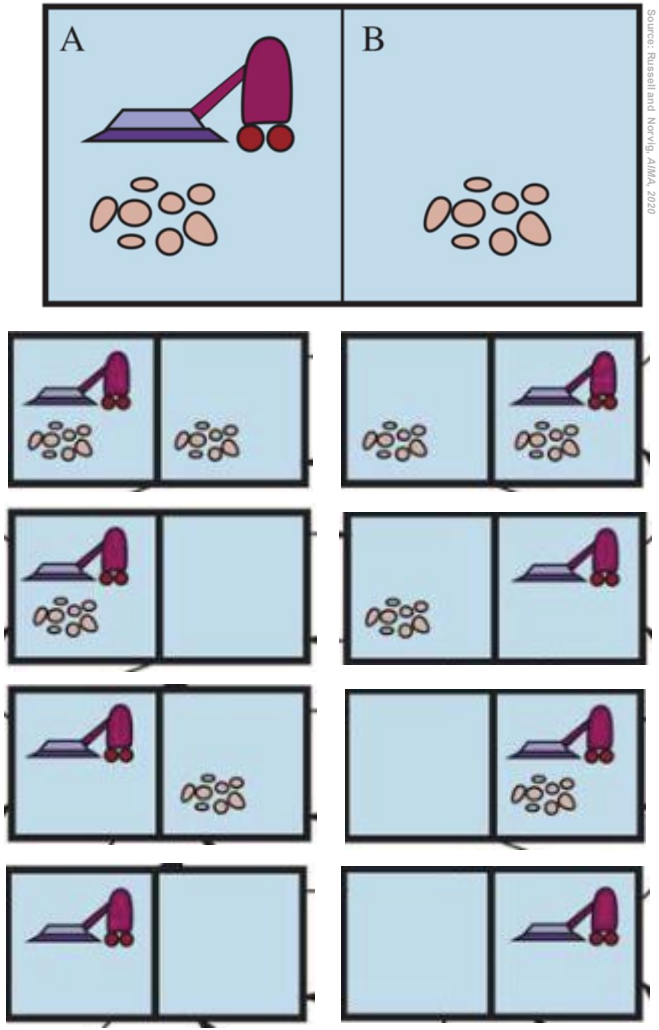

觀光景點
大眾運輸
停車場
藥局
自動提款機
動物園站
臺北市立動物園
國立政治大學
動物園南站
政治大學後門
臺北市文山區指南實驗國民小學
貓空纜車指南宮站
貓空站
台北捷運公司木柵機廠
魚衛子
公館後
抱子腳
萬芳路
文山區
木柵
貓空
新興
小坑
炮子崙
指南宮
猴山岳
石坡坑
岐山
樟山寺
歐比咖啡
木柵休息站
貓空山水客景觀茶館
貓空觀光茶園
貓空
found.your.tea
地圖資料 ©2023 Google
台灣
御櫻[櫻花林]



Source: https://www.itworldcanada.com/article/d... Source: Russell and Norvig, AIMA, 2020 ...my-home-represent-a-major-privacy-risk/412128

Example: Vacuum World

- Percepts
  - Location: A or B
  - Status: clean or dirty
- States
  - $2 \cdot 2^2 = 8$ states
- Actions
  - Move left
  - Move right
  - Suck

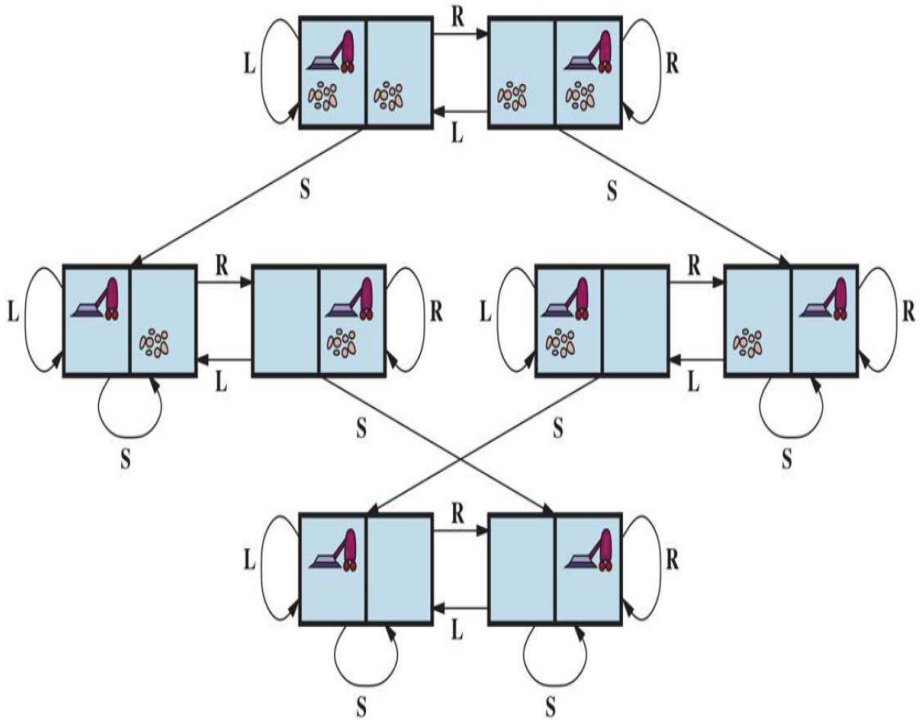


Source: Russell and Norvig, AIMA, 2020

Example: Vacuum World

- States
  - 8 states
- Actions
  - Move left, move right, suck
- Transition model
- Initial state
- Goal test
  - Every cell is clean

State Space Graph



Source: Russell and Norvig, *AIMA, 2020*

Agent plans **a sequence of actions**
that form a path to a goal state

Solving Problems by Searching

Course Topics

- Intelligent agents (AIMA Ch. 2)
- **Search (AIMA Ch. 3, 4, 6, 5)**
- Reasoning (AIMA Ch. 7-9, 12-15)
- Machine learning (AIMA Ch. 19-20)
- Deep learning (AIMA Ch. 22)
- Natural language processing (AIMA Ch. 24)
- Generative AI (Optional)

Search

L.-Y. Wei

Spring 2024

Search Topics

- Search problems (Ch. 3)
- Uninformed search (Ch. 3)
- Informed search (Ch. 3)
- Local search (Ch. 4)
- Adversarial search (Ch. 6)
- Constraint Satisfaction Problems (CSPs) (Ch. 5)

Search Topics

- **Search problems** (Ch. 3)
- Uninformed search (Ch. 3)
- Informed search (Ch. 3)
- Local search (Ch. 4)
- Adversarial search (Ch. 6)
- Constraint Satisfaction Problems (CSPs) (Ch. 5)

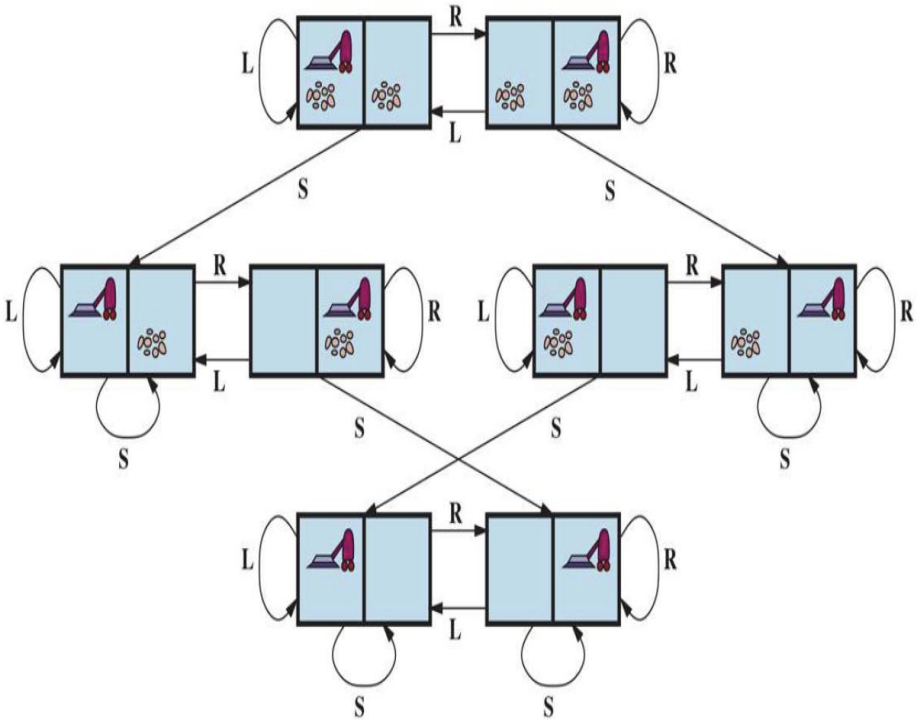
Search Problems

- A **search problem** consists of
  - State space
  - Initial state
  - Goal test (goal states)
  - Actions
  - Transition model
  - Action cost function
- A **solution** is a sequence of actions (path) which transforms the initial state to a goal state
- An **optimal solution** has the lowest path cost among all solutions

Example: Vacuum World

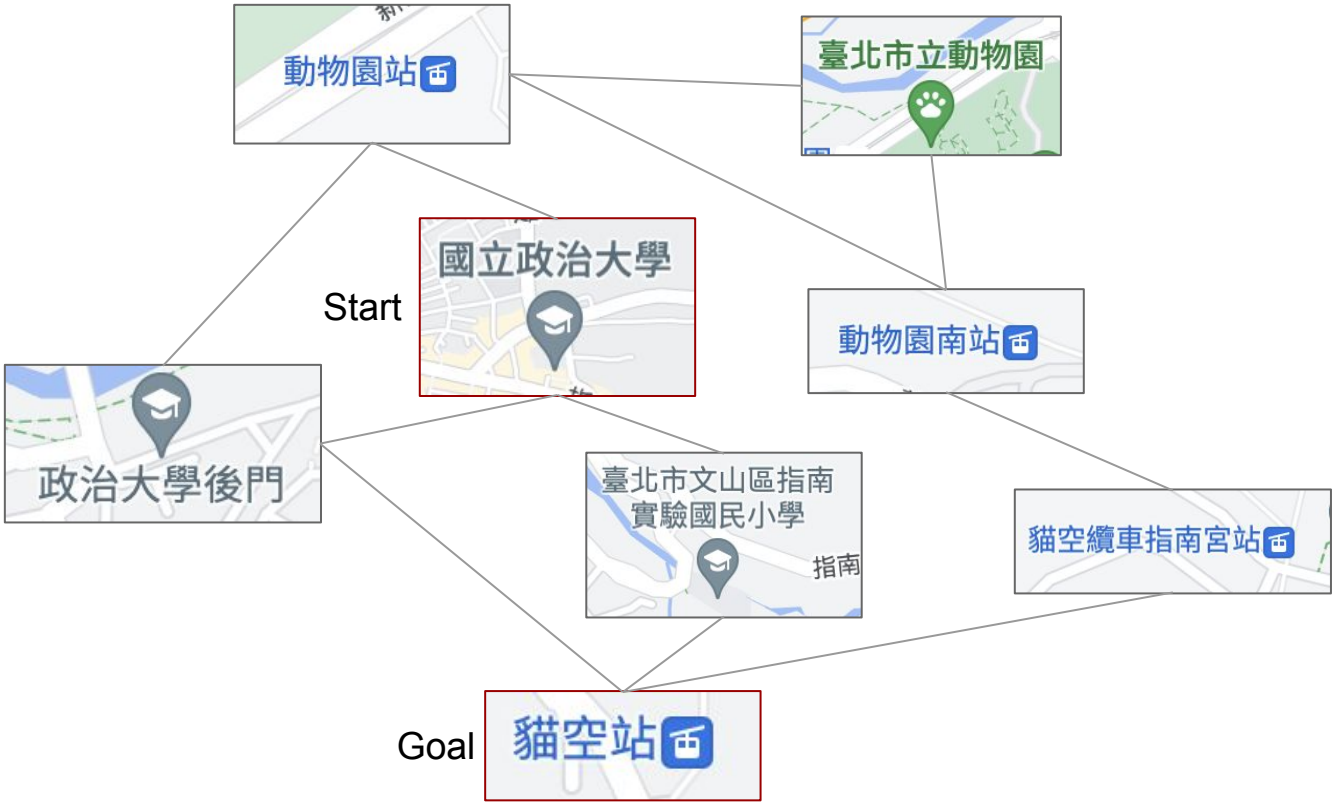
- States
  - 8 states
- Actions
  - Move left, move right, suck
- Transition model
- Initial state
- Goal test
  - Every cell is clean
- Action cost function
  - Each action costs 1

State Space Graph

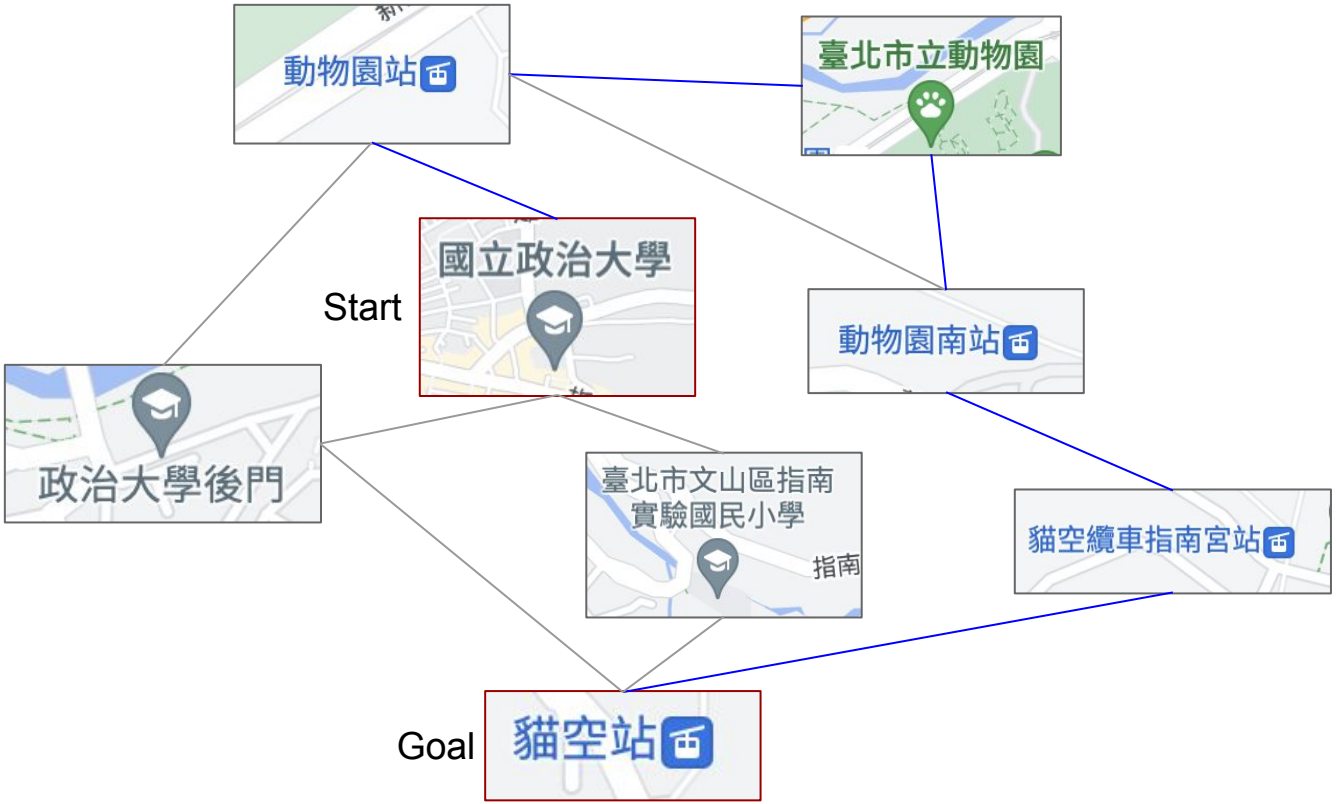


Source: Russell and Norvig, *AIMA, 2020*

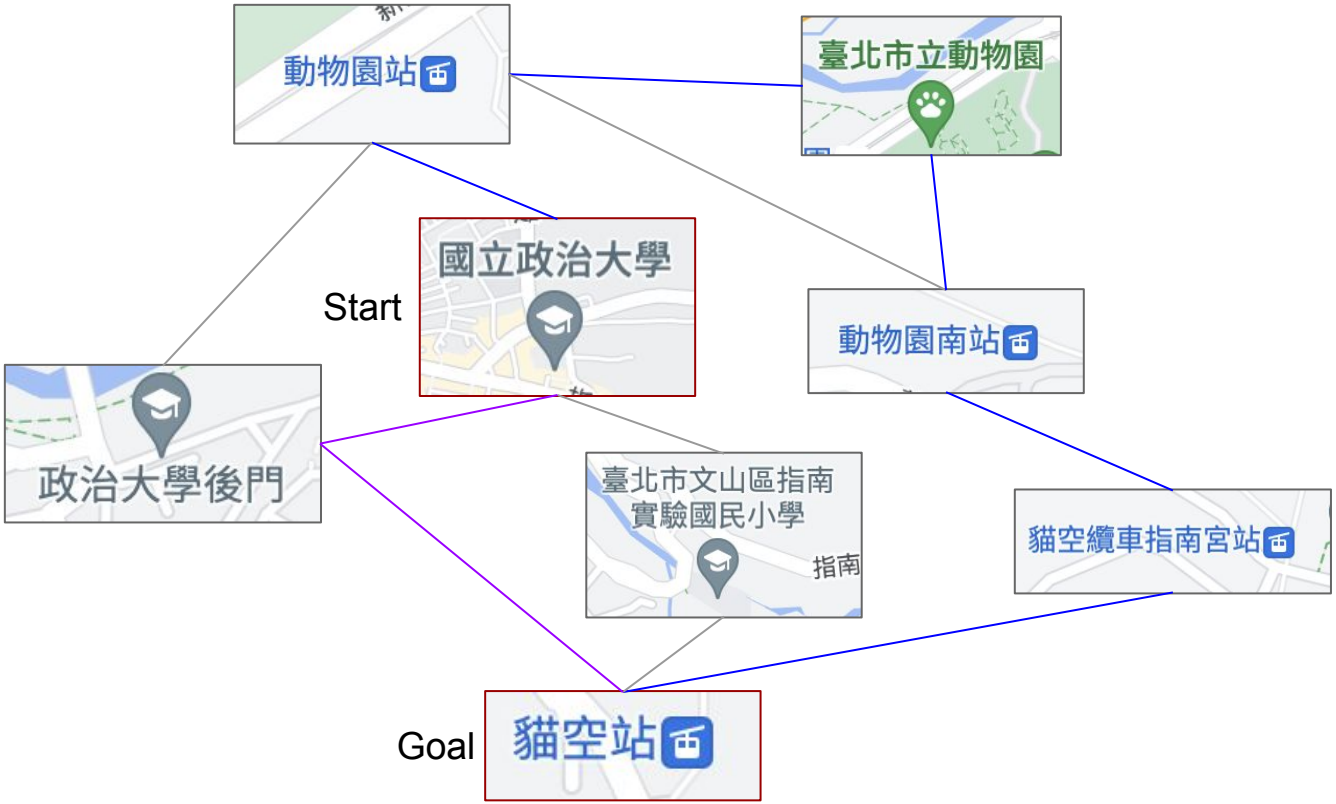
Example: Route-Finding Problem



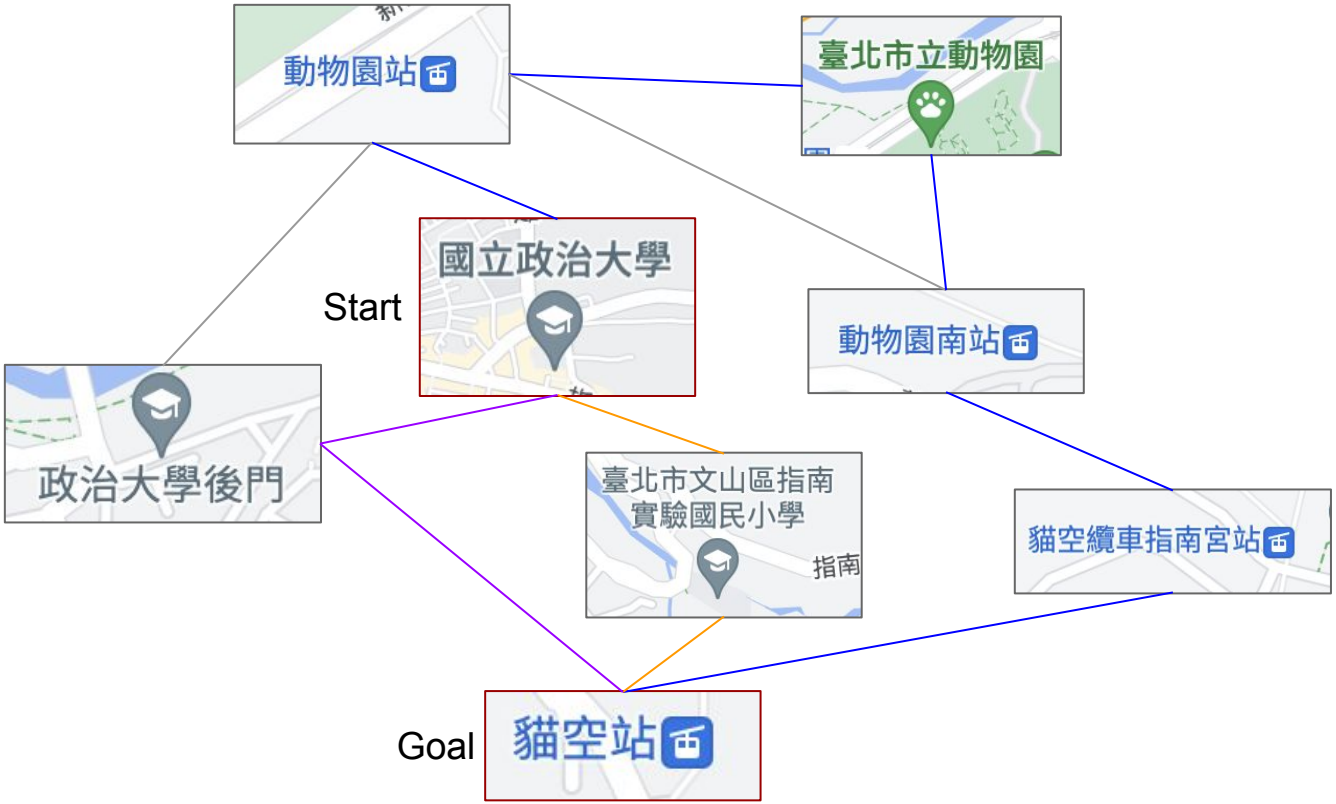
Example: Route-Finding Problem



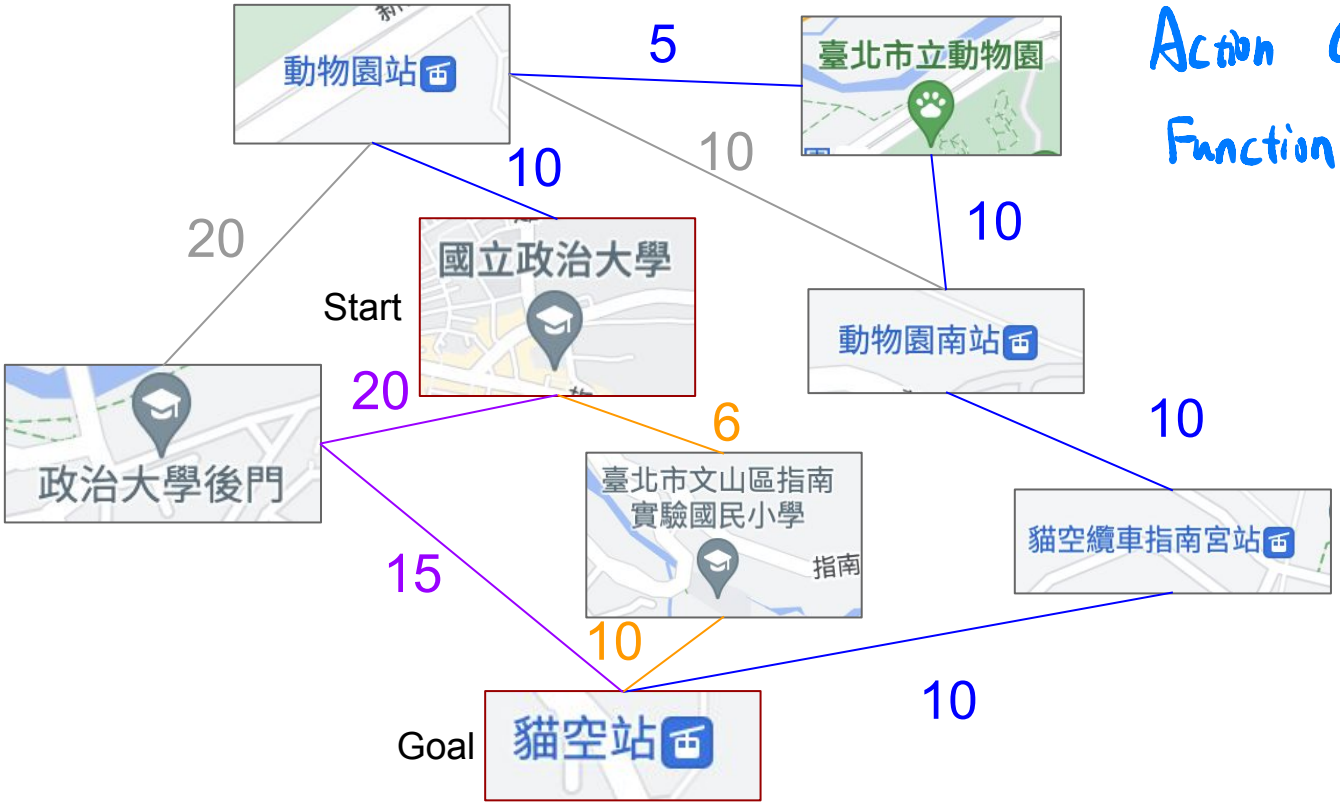
Example: Route-Finding Problem



Example: Route-Finding Problem



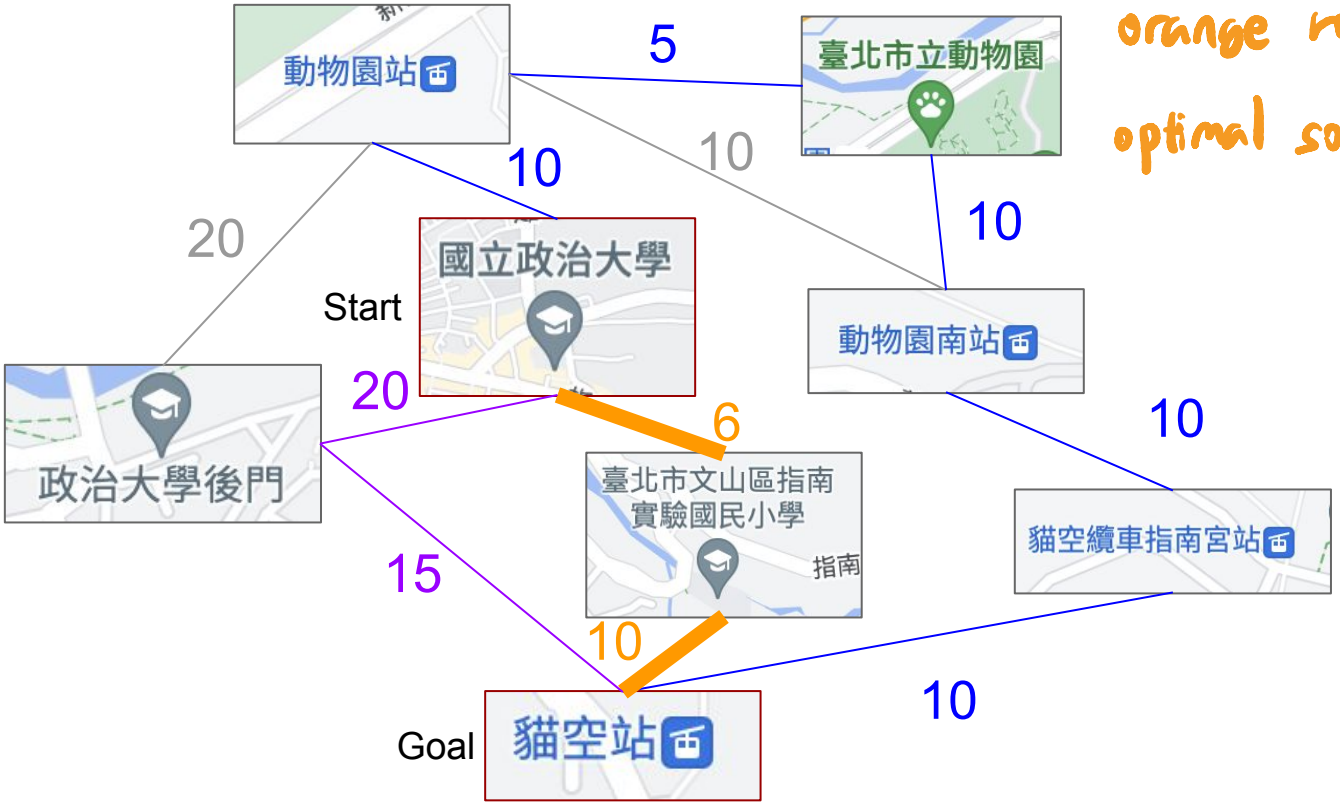
Example: Route-Finding Problem



Example: Route-Finding Problem

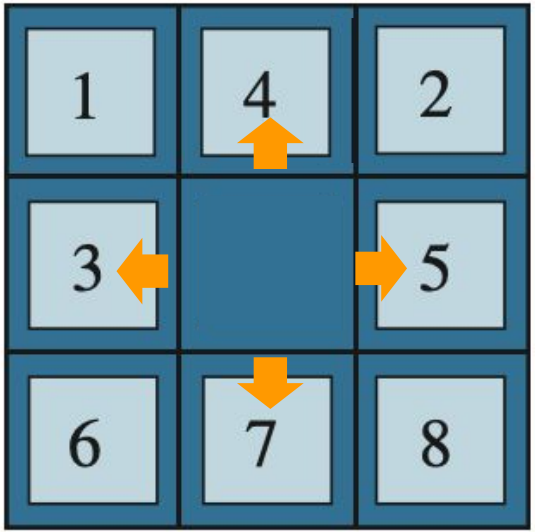
each route : solution

orange route : optimal solution

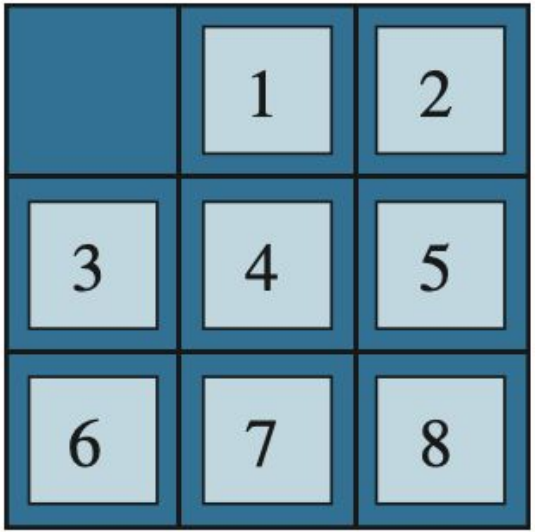


Example: The 8-Puzzle Game

Action

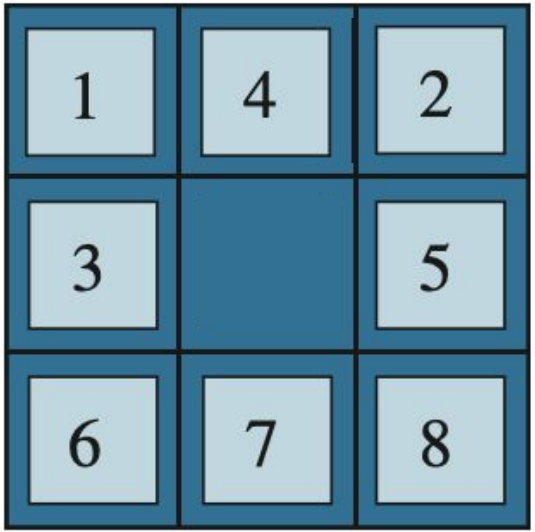


Start State



Goal State

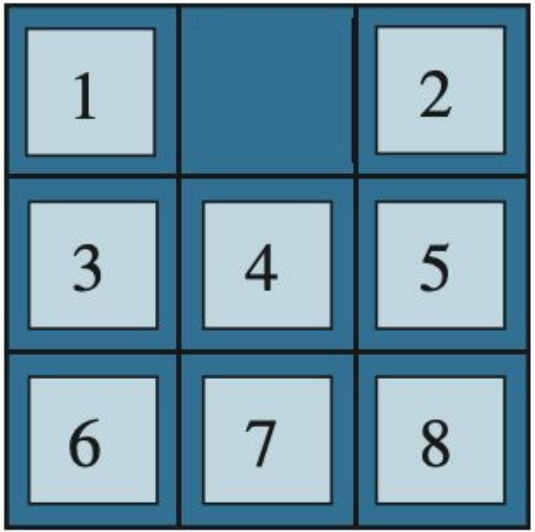
Example: The 8-Puzzle Game



Start State

+ Action: ↑

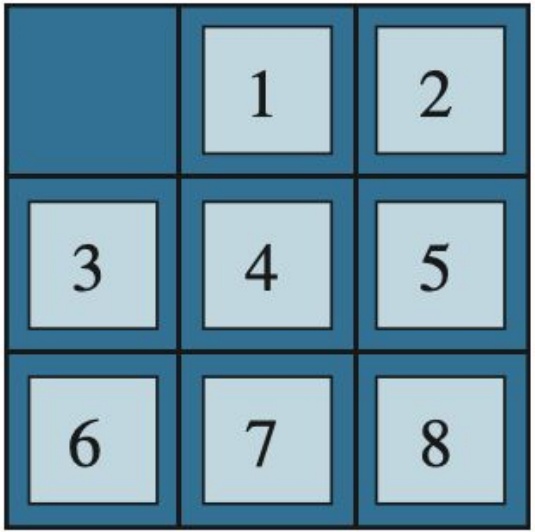
Example: The 8-Puzzle Game



+ Action: ←

Source: Russell and Norvig, *AIMA, 2020*

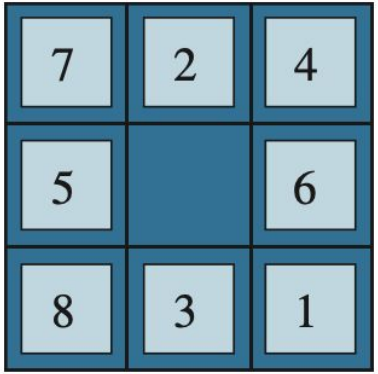
Example: The 8-Puzzle Game



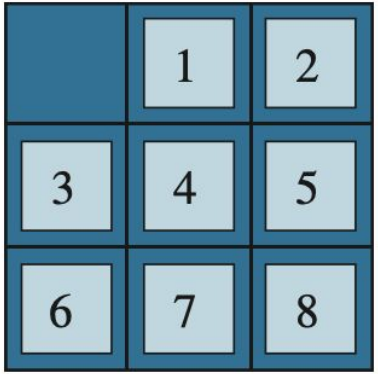
Goal State

Example: The 8-Puzzle Game

Source: Russell and Norvig, *AIMA, 2020*

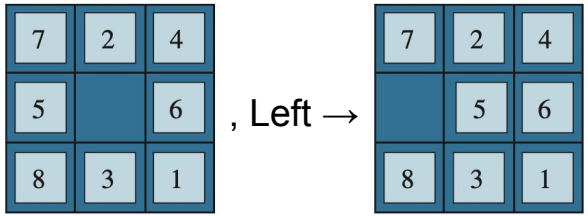


Start State



Goal State

- States
  - A state description specifies the location of each of the tiles
- Initial state
- Goal state
- Actions
  - The blank space moves Left, Right, Up, or Down
    (If the blank is at an edge or corner then not all actions will be applicable)
- Transition model
  - Maps a state and action to a resulting state
- Action cost
  - Each action costs 1



Search Problems

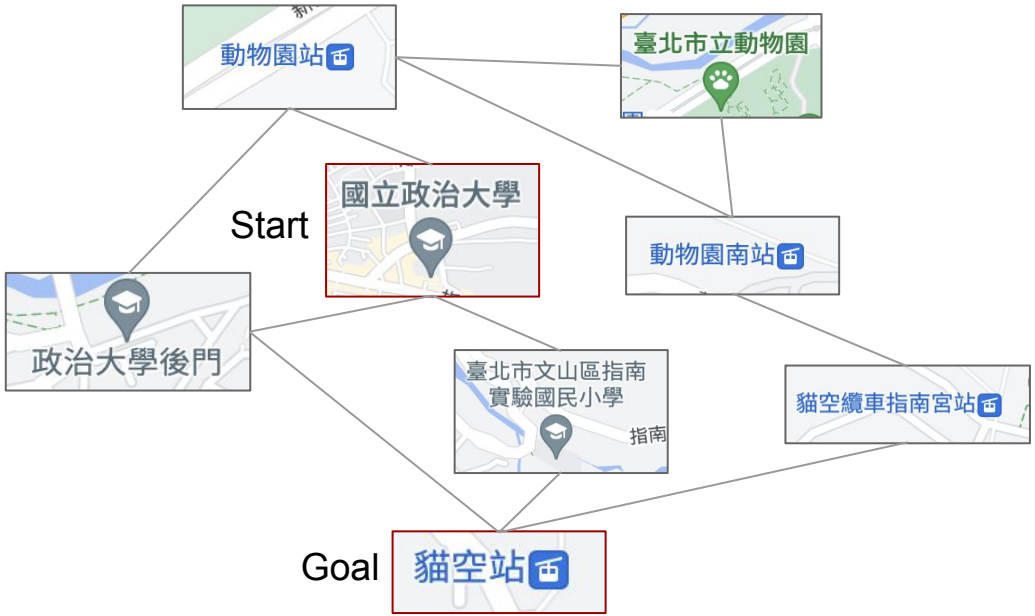
How to solve Search Problems?

Search Algorithm

- A search algorithm takes a search problem as input and returns a solution, or an indication of failure

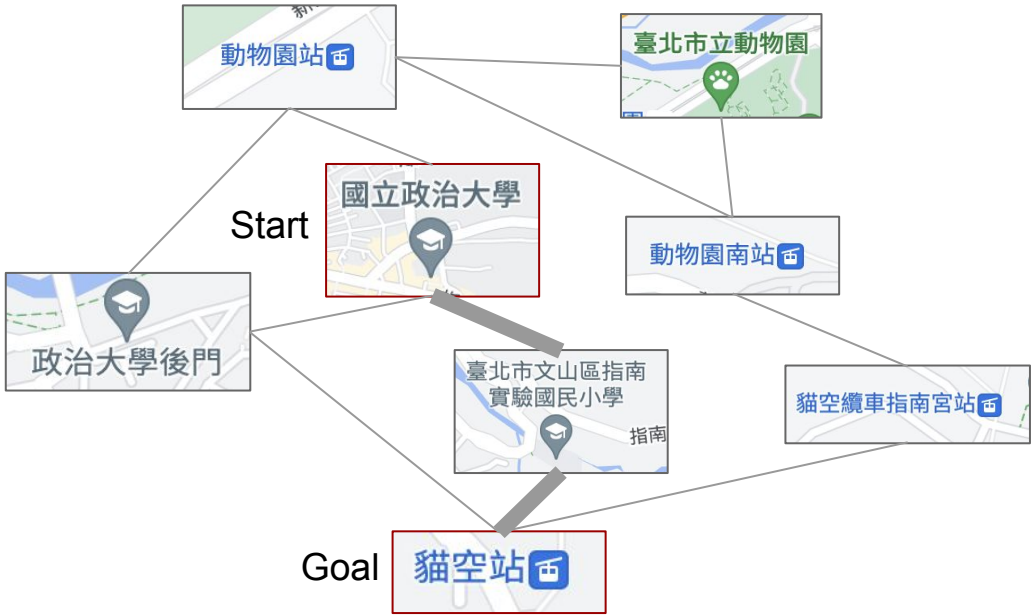
Search Algorithm

- A search algorithm takes a search problem as input and returns a solution, or an indication of failure

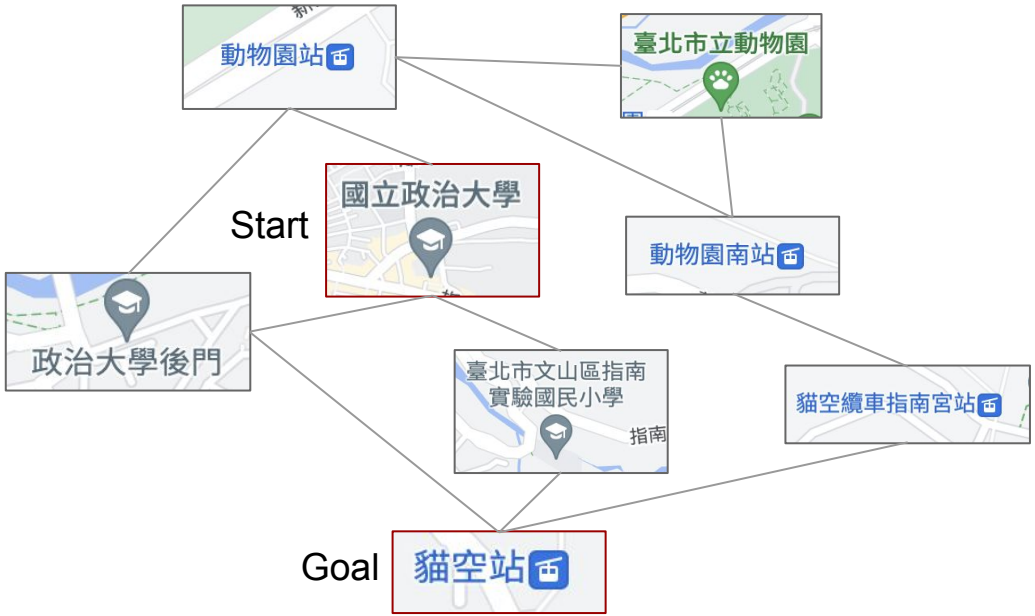


Search Algorithm

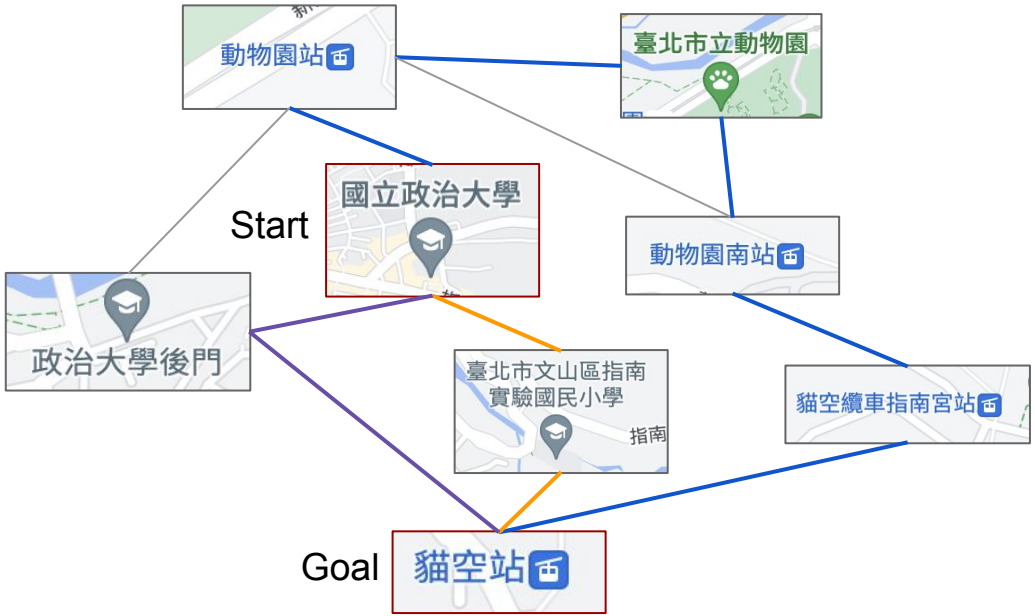
- A search algorithm takes a search problem as input and returns a solution, or an indication of failure



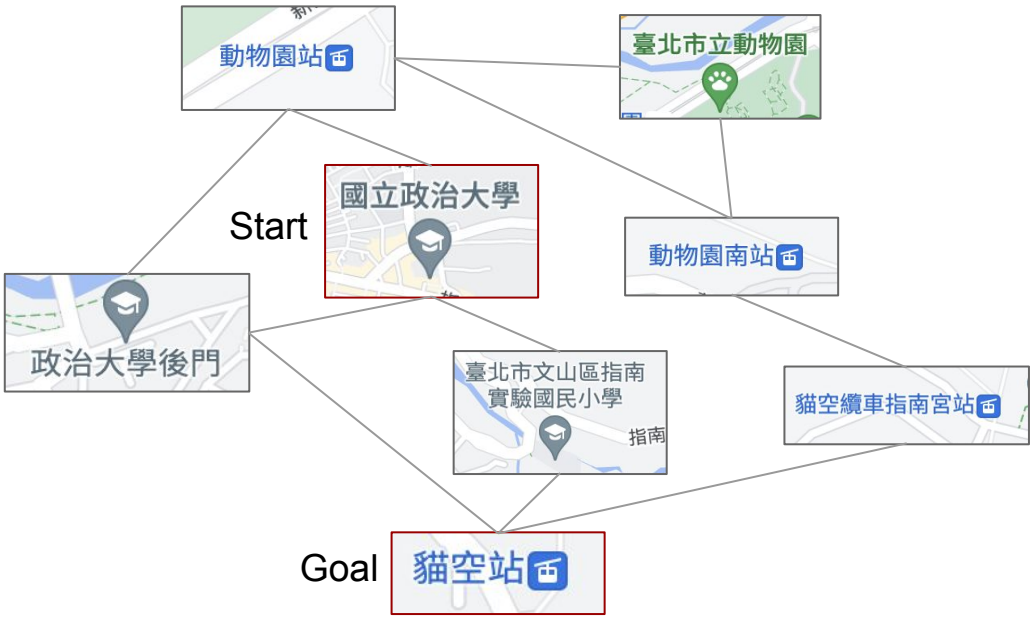
Redundant Path Issue



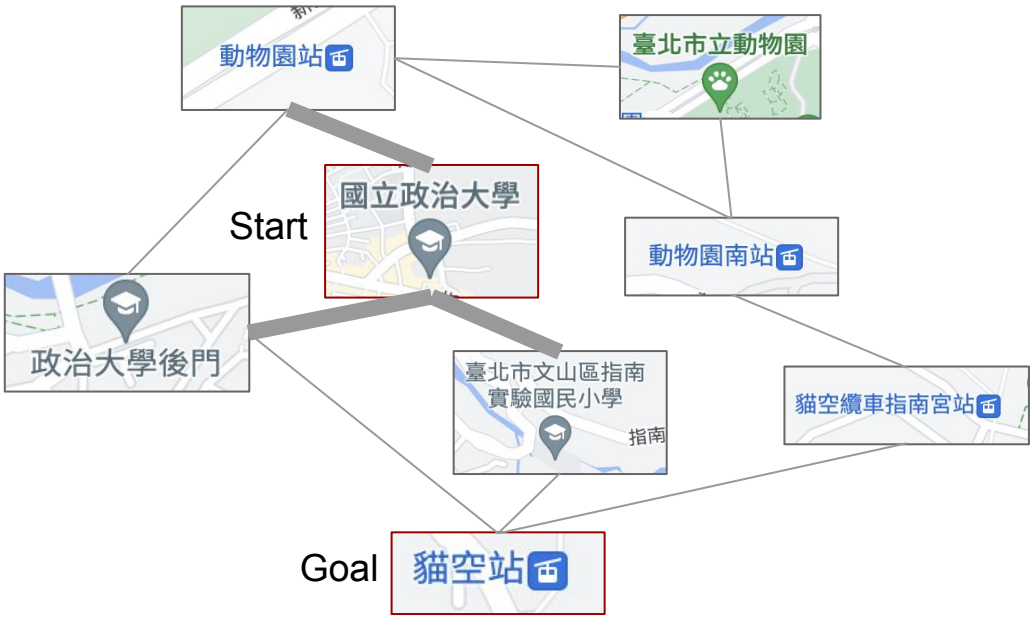
Redundant Path Issue I



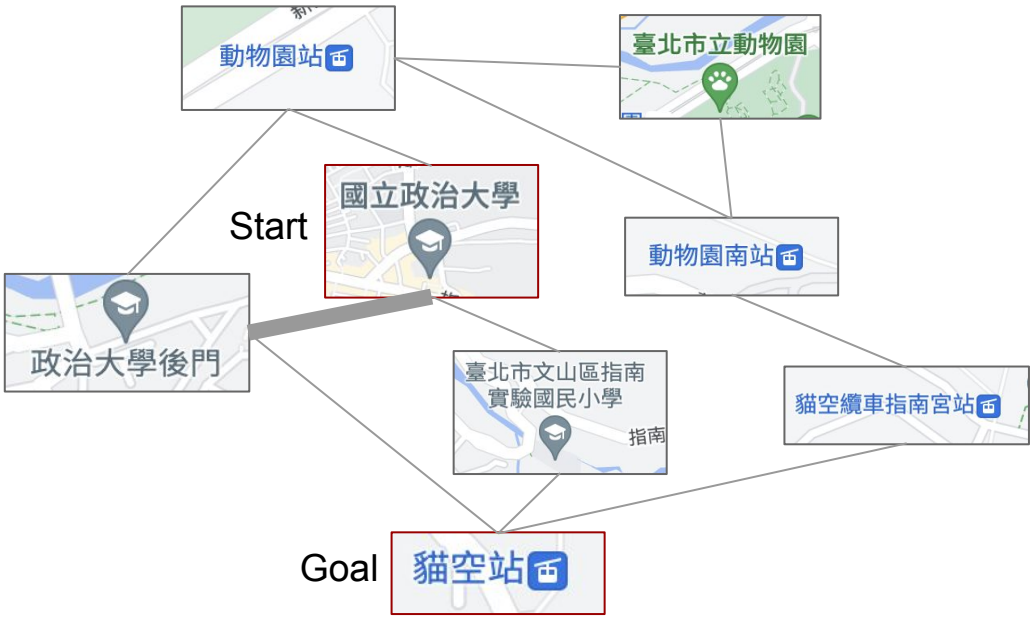
Redundant Path Issue II



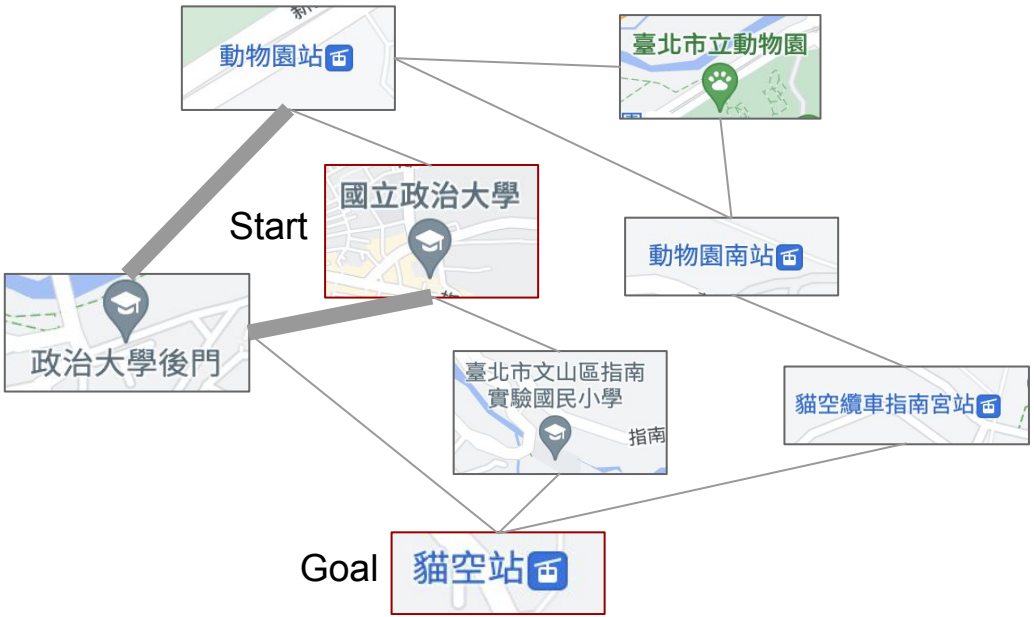
Redundant Path Issue II



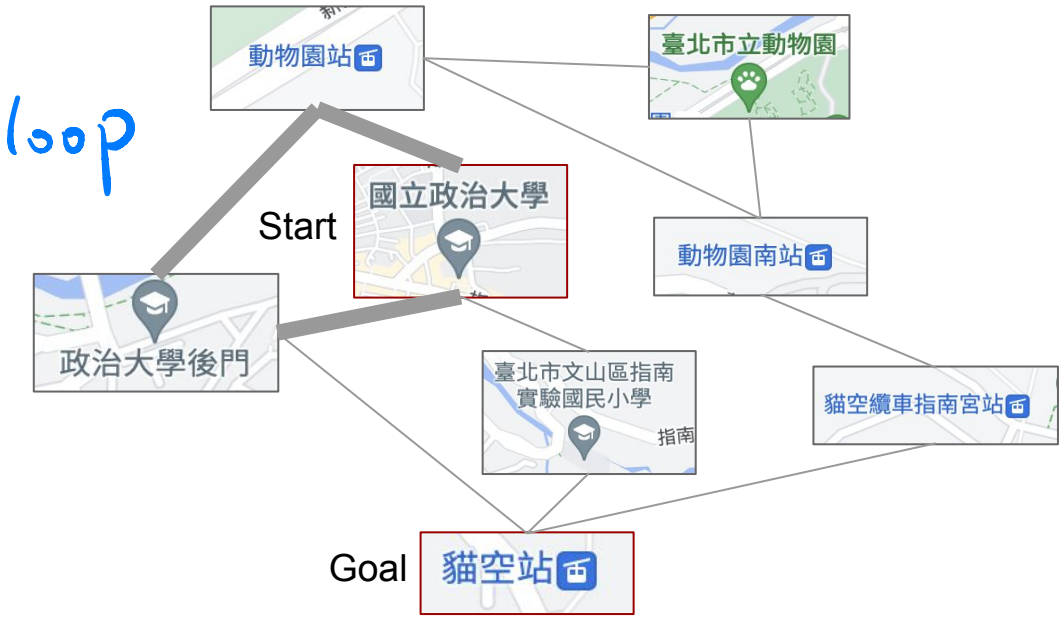
Redundant Path Issue II



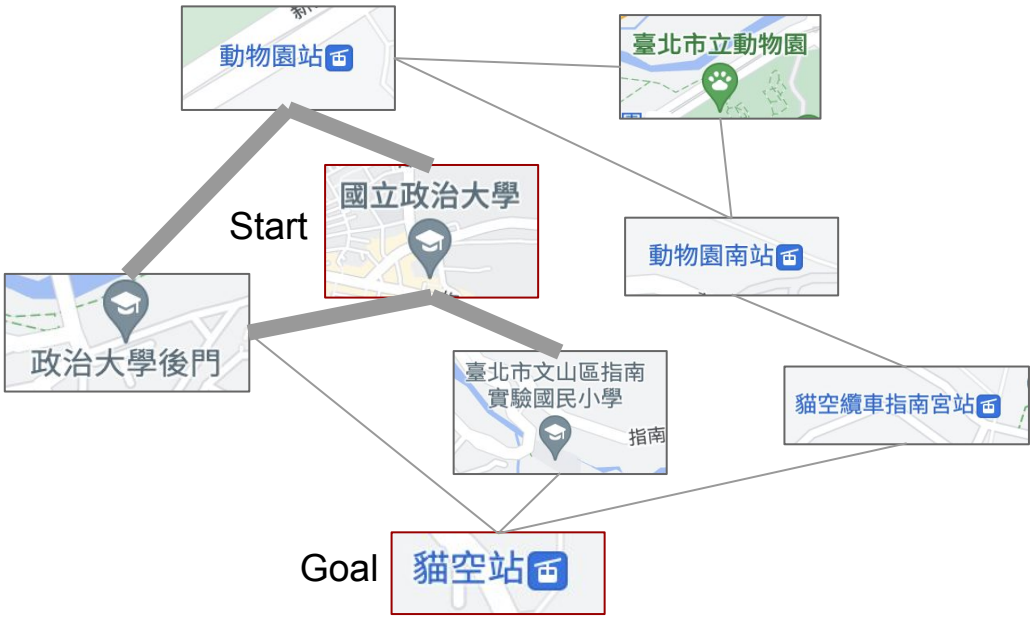
Redundant Path Issue II



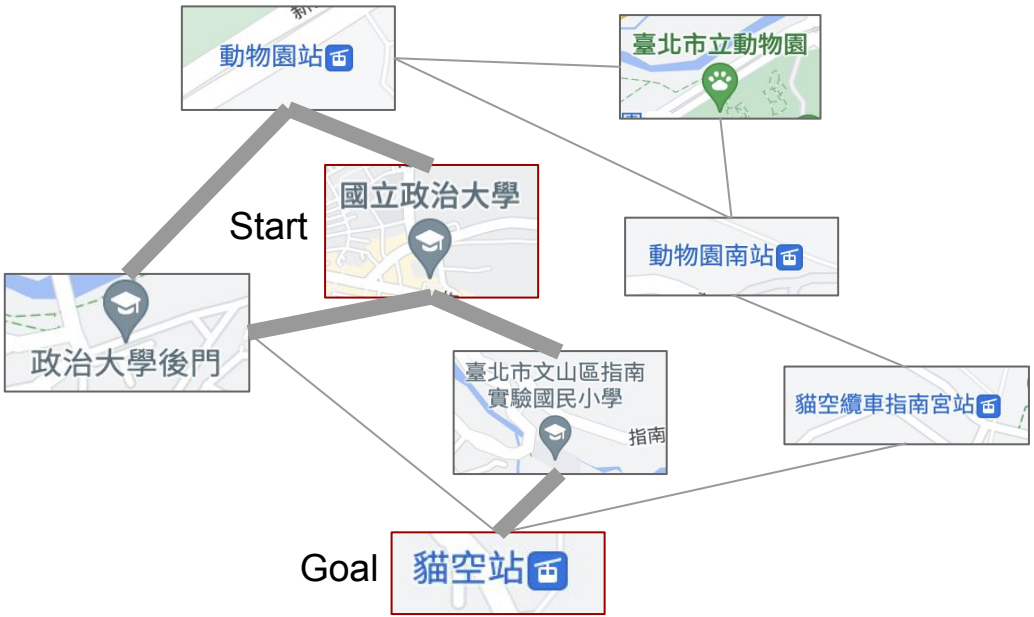
Redundant Path Issue II



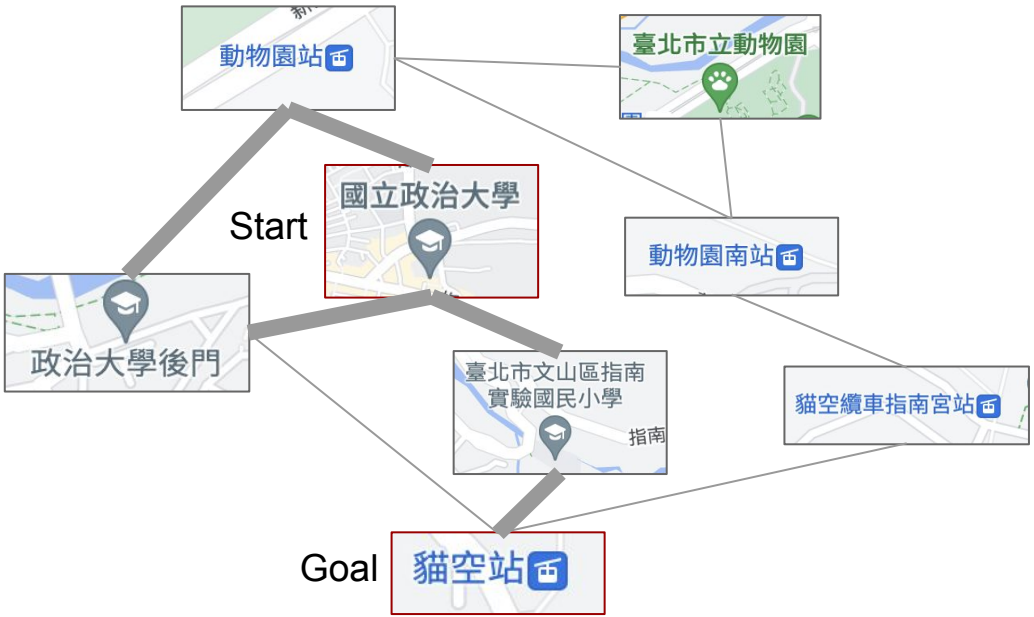
Redundant Path Issue II



Redundant Path Issue II



Redundant Path Issue II: Cycle (Loopy Path)



Three Approaches to the Redundant Path Issue

- Remember all previously **reached** states and keep only the best path to each state
- Check redundant paths
  - Graph search
    - A search algorithm checks for redundant paths
- Check for cycles but not for redundant paths
  - Since each node has a chain of parent pointers, we can check for cycles by following up the chain of parents to see if the state at the end of the path has appeared earlier in the path

Good Search?

Performance of a Search Algorithm

- Completeness
  - Guaranteed to find a solution when there is one
- Cost optimality
  - Guaranteed to find a solution with the lowest path cost
- Time complexity
  - How long does it take to find a solution?
- Space complexity
  - How much memory is needed to perform the search?

Search Topics

- **Search problems** (Ch. 3)
- **Uninformed search (Ch. 3)**
- Informed search (Ch. 3)
- Local search (Ch. 4)
- Adversarial search (Ch. 6)
- Constraint Satisfaction Problems (CSPs) (Ch. 5)

Uninformed Search

L.-Y. Wei

Spring 2024

Environment

Fully Observable
Deterministic
Static

Uninformed Search (Blind Search)

- An uninformed search algorithm is given **no clue** about how close a state is to the goal
  (i.e., only the information available in the problem definition)

Uninformed Search Strategies

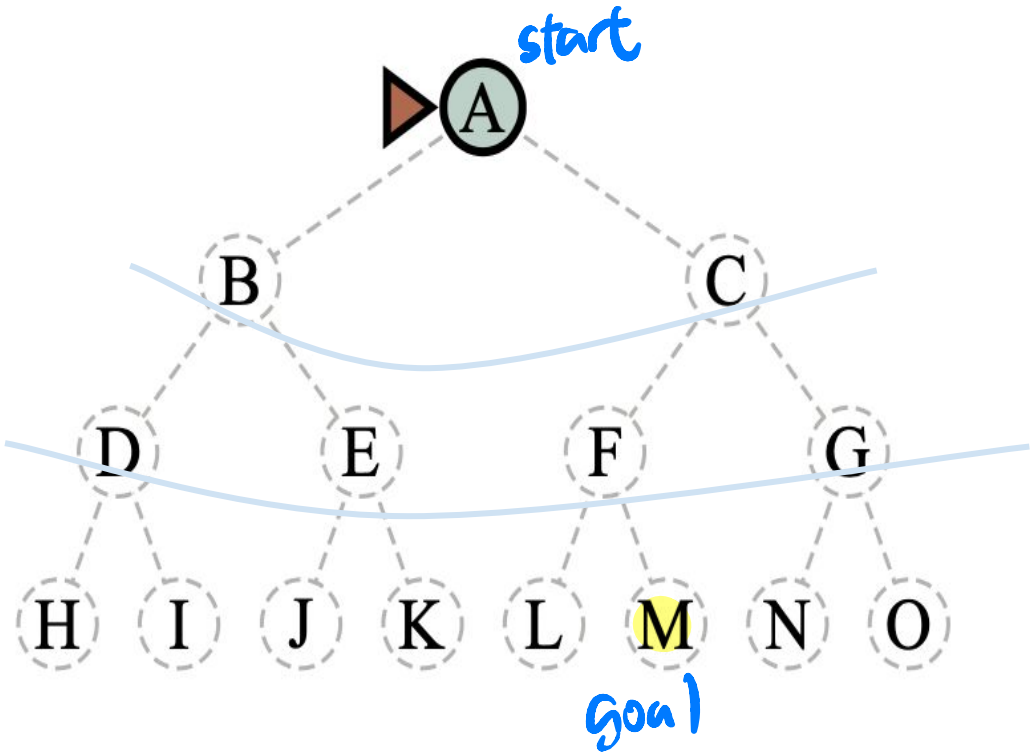
- Breadth-first search
- Depth-first search (DFS)
- Depth-limited search (DLS)
- Iterative deepening search (IDS)
- Uniform-cost search (UCS)
- Bidirectional search

Uninformed Search Strategies

- **Breadth-first search**
- Depth-first search (DFS)
- Depth-limited search (DLS)
- Iterative deepening search (IDS)
- Uniform-cost search (UCS)
- Bidirectional search

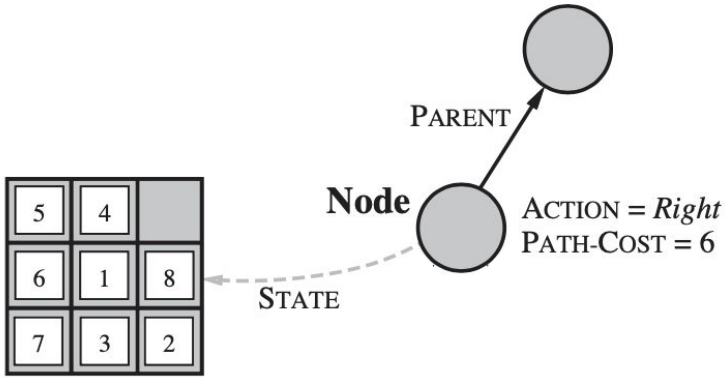
Example: Breadth-First Search on a Simple Binary Tree

- Start: A
- Goal: M



Implementation Issue

- A node in the tree is represented by a data structure with four components
  - node.STATE: the state to which the node corresponds
  - node.PARENT: the node in the tree that generated this node
  - node.ACTION: the action that was applied to the parent's state to generate this node
  - node.PATH-COST: the total cost of the path from the initial state to this node



Implementation Issue (cont.)

```
function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

// apply all actions

// new cost

A function used to generate all successor nodes by applying each possible actions to the current state

Breadth-First Search Algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

edge case

: queue

: keeps track of the reached states

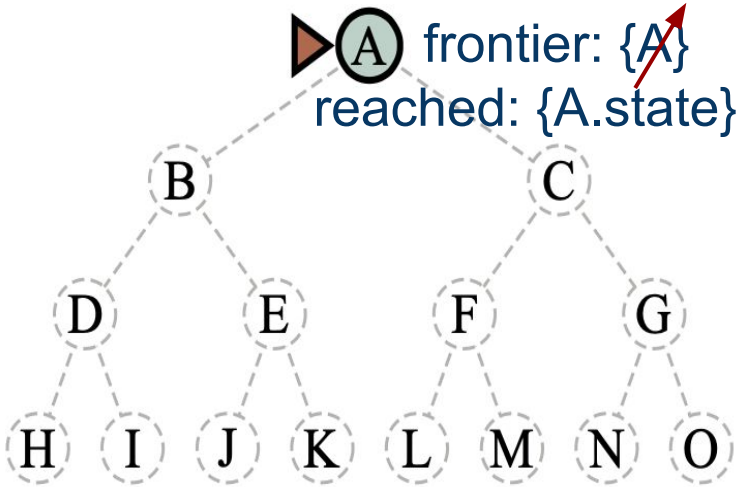
See the previous slide (try each action)

: goal reached

: fails to reach the goal

Example: Breadth-First Search on a Simple Binary Tree

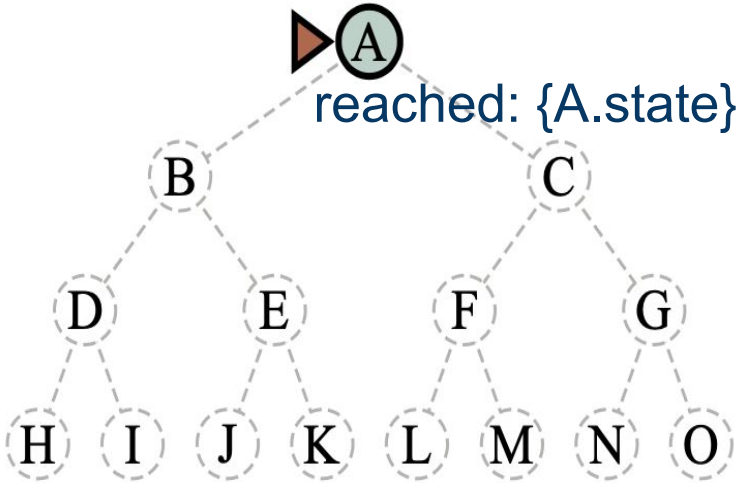
- Start: A
- Goal: M



Frontier: {}

Example: Breadth-First Search on a Simple Binary Tree

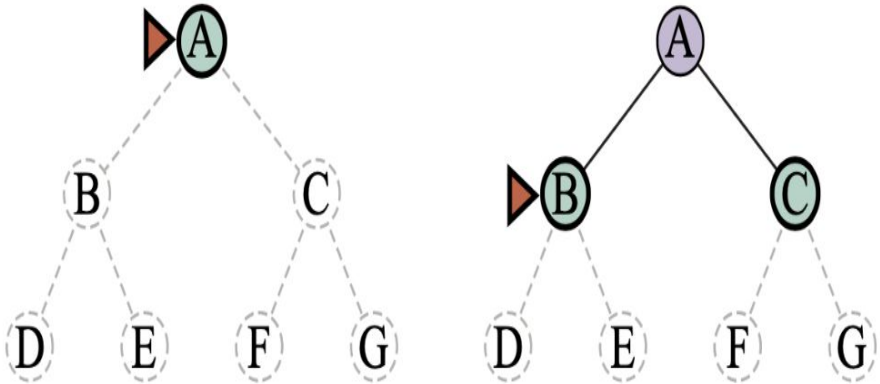
- Start: A
- Goal: M



reached: {A.state,B.state,

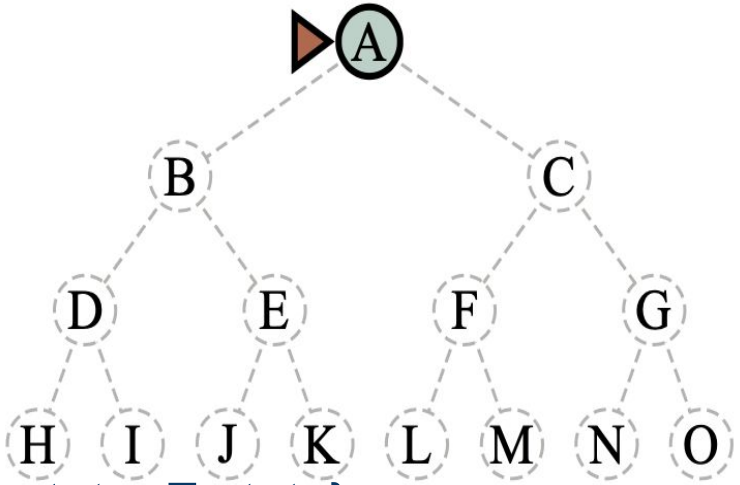
Frontier: {}

frontier: {B }



Example: Breadth-First Search on a Simple Binary Tree

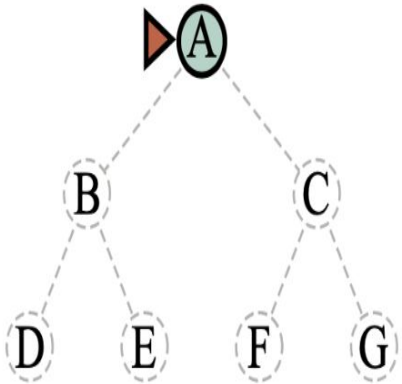
- Start: A
- Goal: M

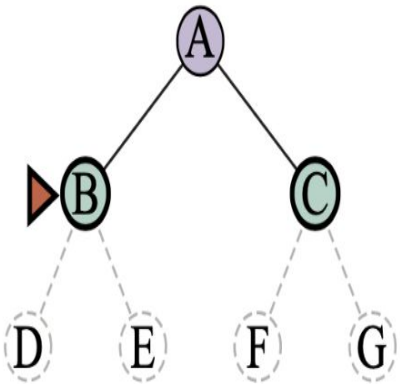


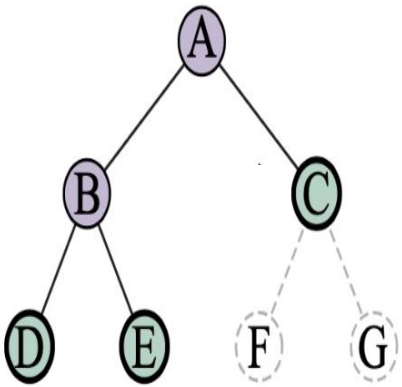
reached: {A.state,B.state,C.state, D.state, E.state}

frontier: {~~B~~,C}

Frontier: {~~C~~,D,E}

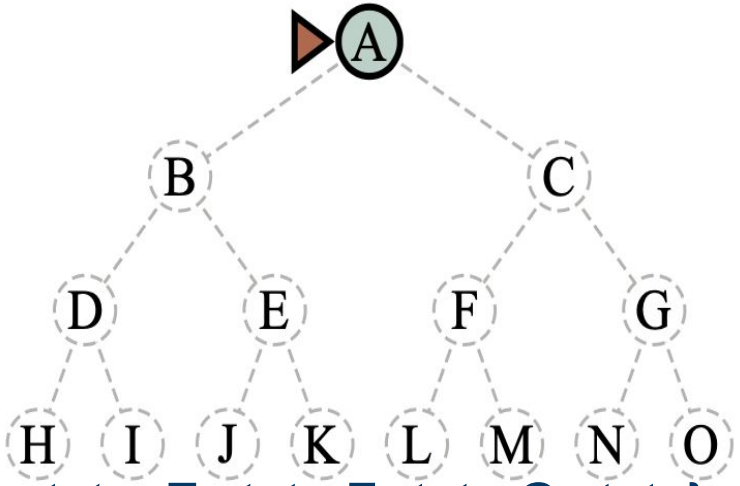






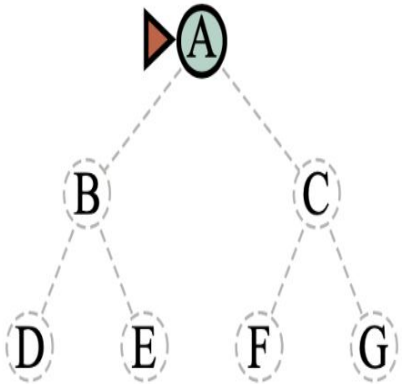
Example: Breadth-First Search on a Simple Binary Tree

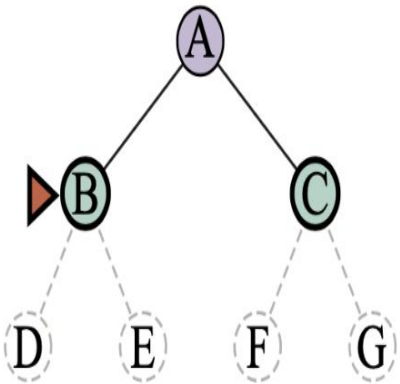
- Start: A
- Goal: M

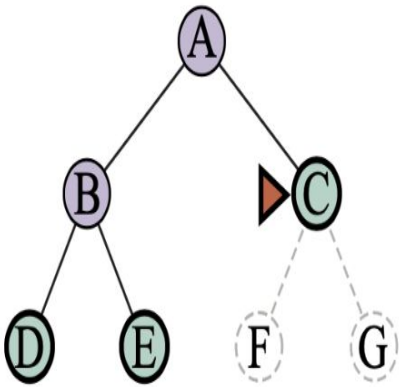


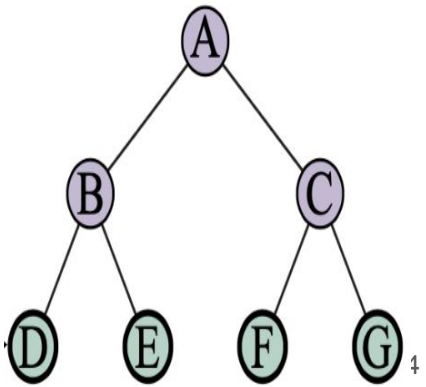
reached: {A.state,B.state,C.state, D.state, E.state,F.state,G.state}

Frontier: {C,D,E} Frontier: {D,E,F,G}









Properties of Breadth-First Search

Note. b is the branching factor and d is the depth of the shallowest solution

- Complete?
  - Yes (if b is finite and the state space either has a solution or is finite)
- Optimal cost?
  - Yes (if action costs are all identical)
- Time complexity?
  - $O(b^d)$
    - $b^0 + b^1 + b^2 + \cdots + b^d = O(b^d)$
- Space complexity?
  - $O(b^d)$

∵ BFS performs search level by level

In the worst case, BFS travels all nodes at each level until it finds the shallowest solution

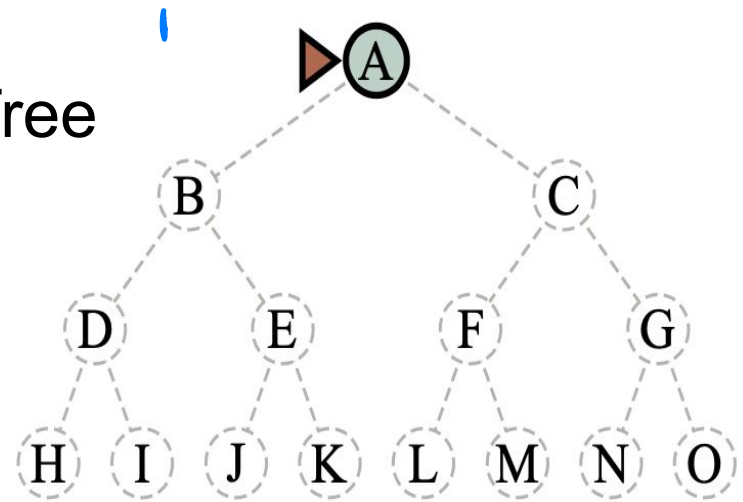
$b^0 + b^1 + \ldots + b^d$ # of nodes in the last level

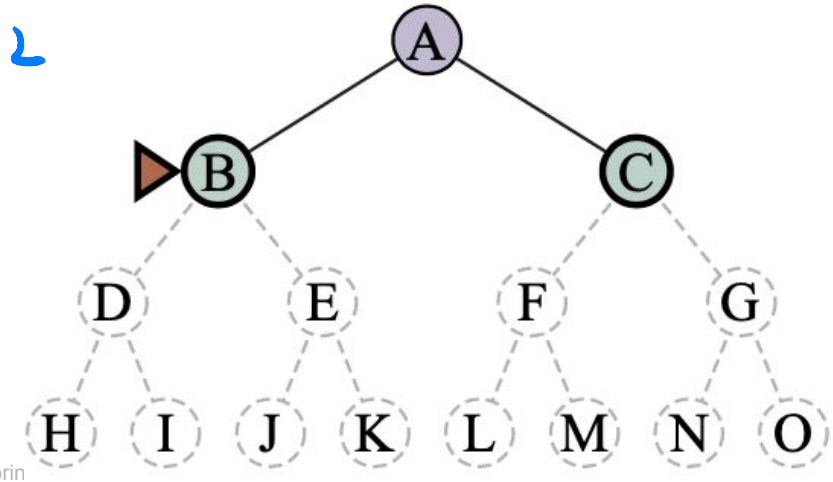
Uninformed Search Strategies

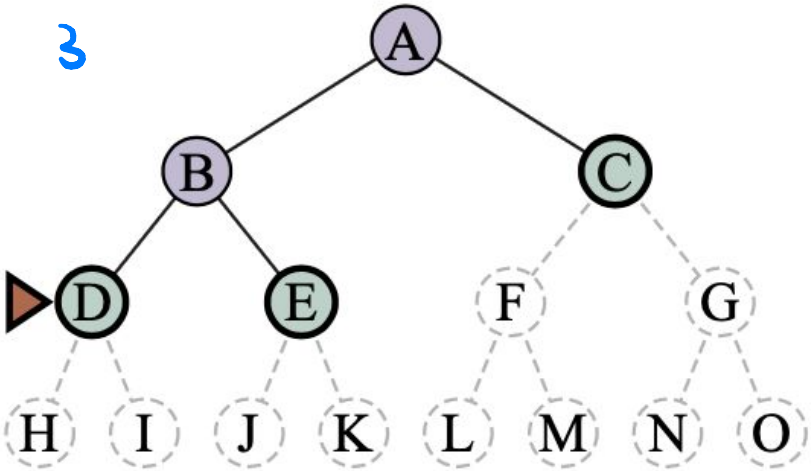
- Breadth-first search
- **Depth-first search (DFS)**
- Depth-limited search (DLS)
- Iterative deepening search (IDS)
- Uniform-cost search (UCS)
- Bidirectional search

Example: DFS on a Simple Binary Tree

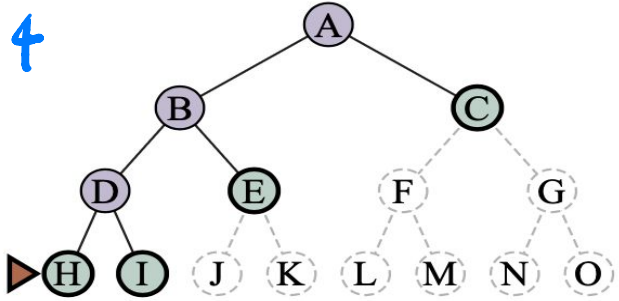
- Start: A
- Goal: M



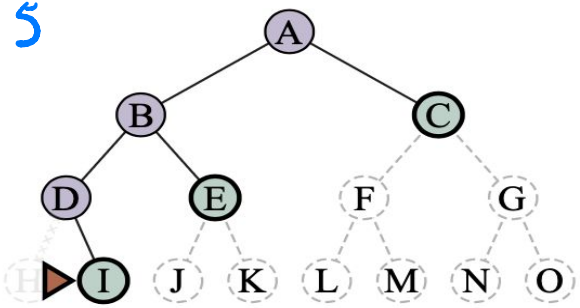




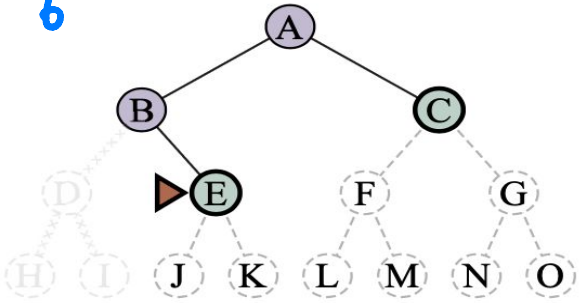
4



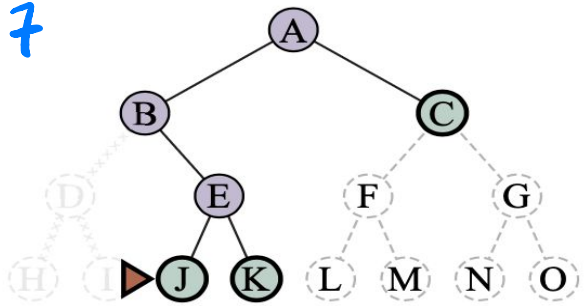
5



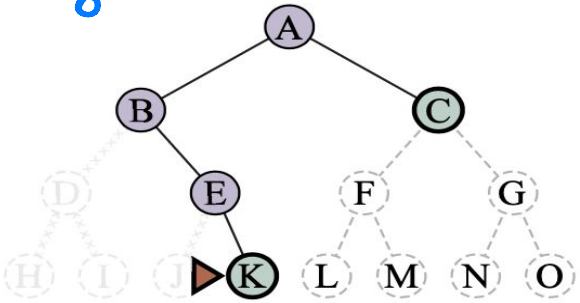
6



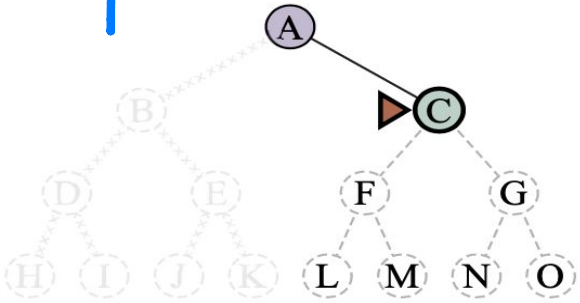
7



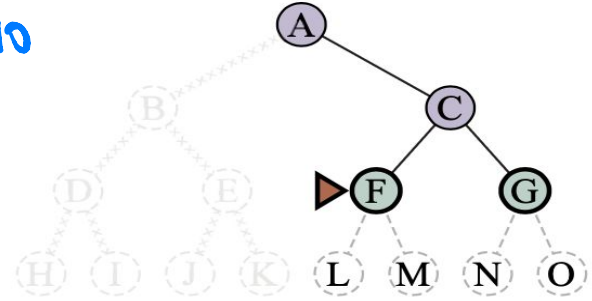
8



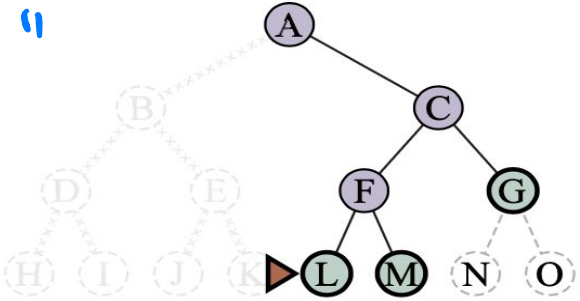
9



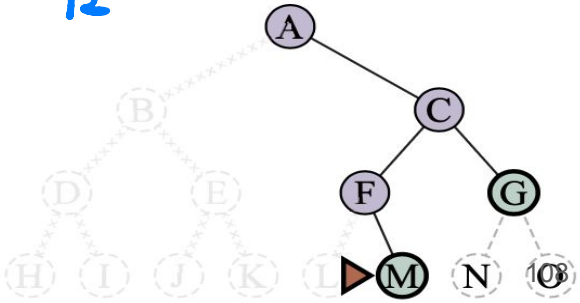
10



11



12



Pseudocode for DFS in Graph

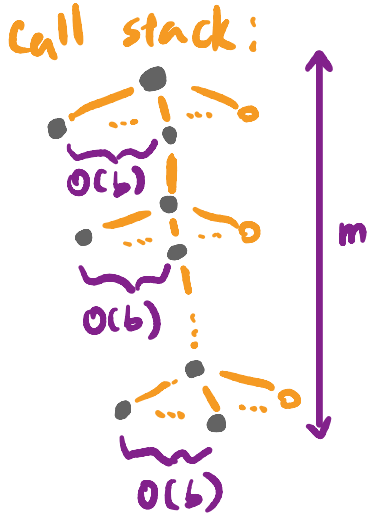
```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)

init() {
    For each u ∈ G
        u.visited = false
     For each u ∈ G
       DFS(G, u)
}
```

Properties of DFS

Note. where b is the branching factor and m is the maximum depth of the search tree

- Complete?
  - No (in infinite-depth spaces, spaces with loops)

    Modify to avoid repeated states along path
    ⇒ complete in finite spaces
- Optimal cost?
  - No
- Time complexity?
  - $O(b^m)$ → In the worst-case, DFS may traverse all the way down to the maximum depth of the tree before backtracking
- Space complexity?
  - $O(bm)$



Uninformed Search Strategies

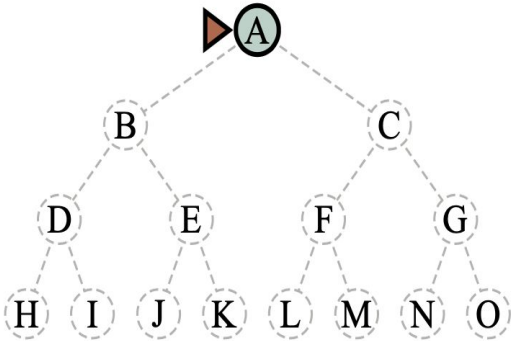
- Breadth-first search
- Depth-first search (DFS)
- **Depth-limited search (DLS)**
- Iterative deepening search (IDS)
- Uniform-cost search (UCS)
- Bidirectional search

Depth-Limited Search (DLS)

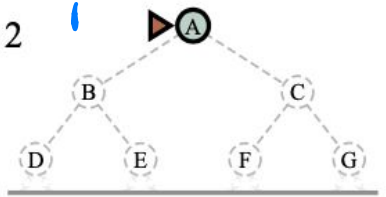
- A version of **depth-first search** in which we supply a **depth limit ℓ** and treat all nodes at the depth ℓ as if they had no successors

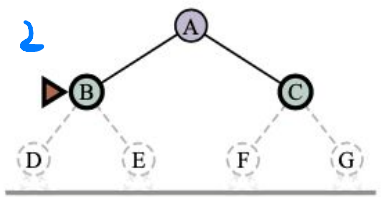
Example: DLS on a Simple Binary Tree

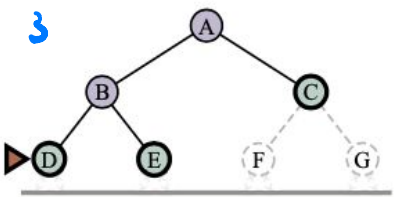
- Start: A
- Goal: M
- ℓ: 2

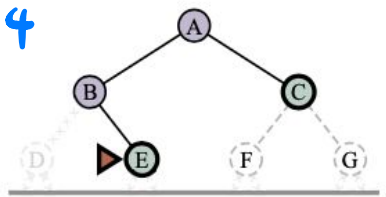


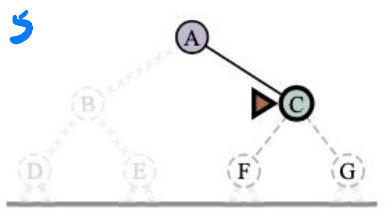
limit: 2

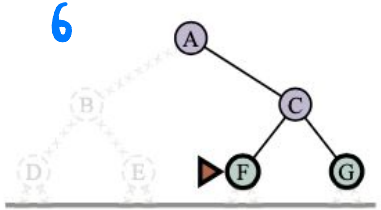


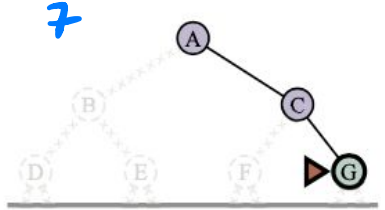












Depth-Limited Search (DLS) Algorithm

```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

: solution found

) limit search depth

Properties of DLS

Note. b is the branching factor and ℓ is the depth limit

- Complete?
  - No
- Optimal cost?
  - No
- Time complexity?
  - $O(b^{\ell})$
- Space complexity?
  - $O(b\ell)$

∵ Depth Limit

$O(1 + b + b^2 + \dots + b^{\ell})$

Refer to DFS

Uninformed Search Strategies

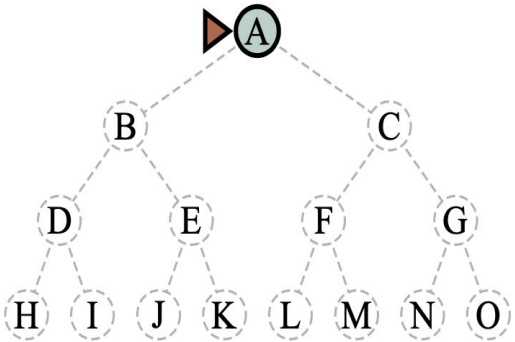
- Breadth-first search
- Depth-first search (DFS)
- Depth-limited search (DLS)
- **Iterative deepening search (IDS)**
- Uniform-cost search (UCS)
- Bidirectional search

Iterative-Deepening Search (IDS)

- Iterative deepening search solves the problem of picking a good value for ℓ by trying all values: first 0, then 1, then 2, and so on — until either **a solution is found**, or the depth-limited search returns the **failure** value

Example: IDS on a Simple Binary Tree

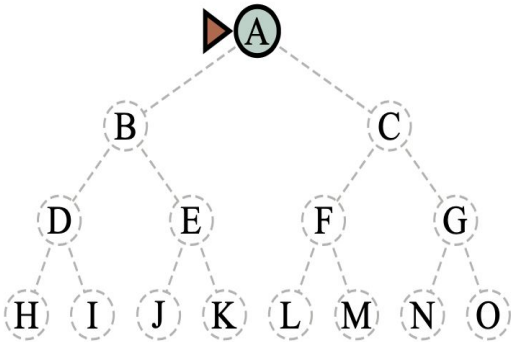
- Start: A
- Goal: M



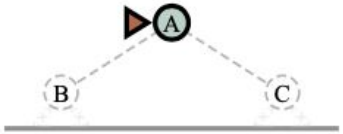
limit: 0

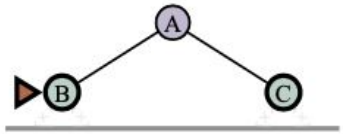
Example: IDS on a Simple Binary Tree

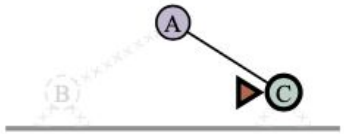
- Start: A
- Goal: M



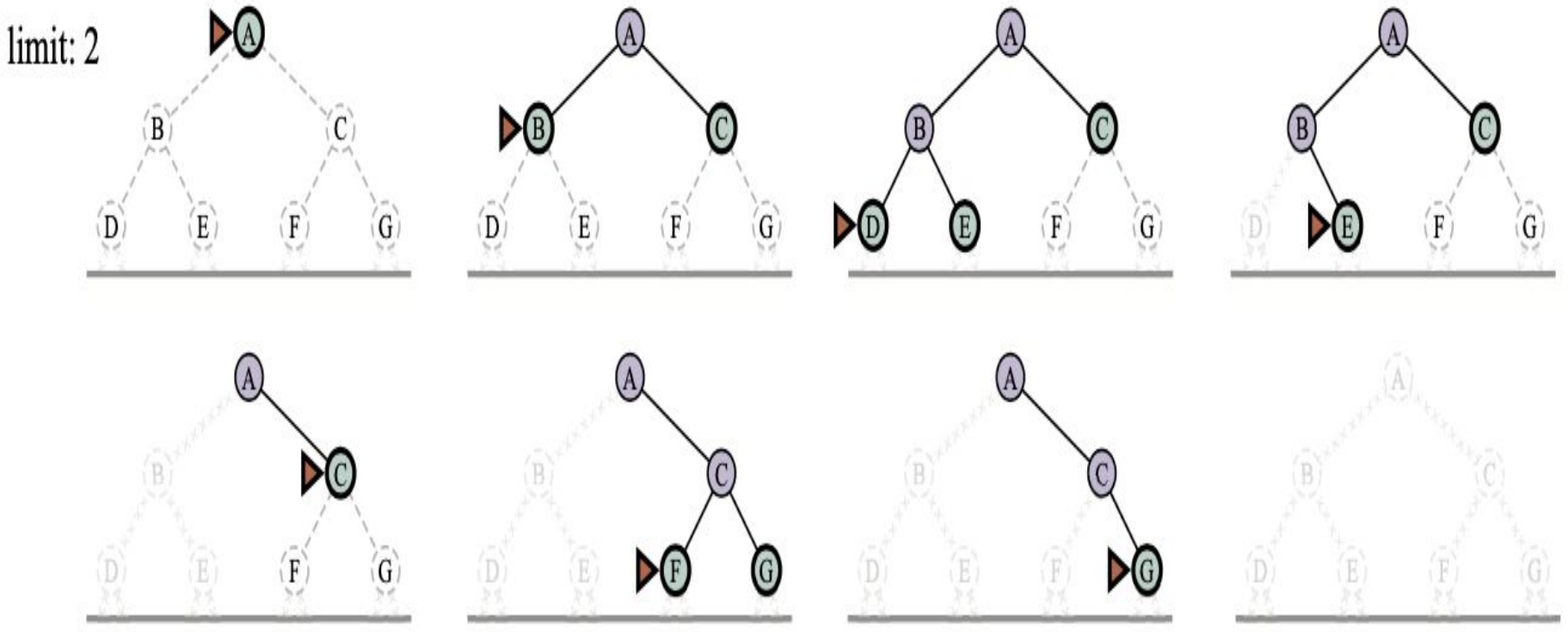
limit: 1



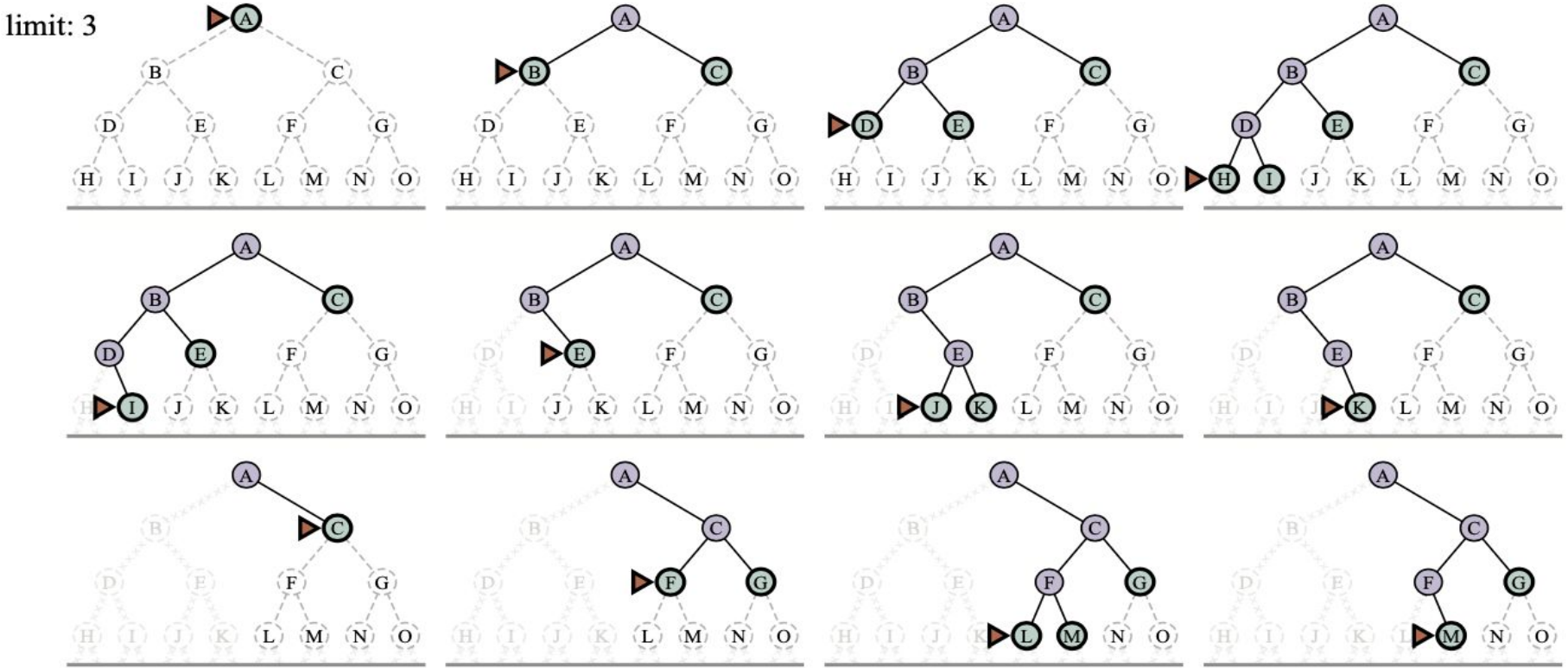




Example: IDS on a Simple Binary Tree



Example: IDS on a Simple Binary Tree



Iterative-Deepening Search (IDS) Algorithm

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Properties of IDS

Note. b is the branching factor and d is the depth of the shallowest solution

- Complete?
  - Yes if the branching factor b is finite
- Optimal cost?
  - Yes if action costs are all identical
- Time complexity?
  - $O(b^d)$
    - $(d + 1)b^0 + (d)b^1 + (d - 1)b^2 + (d - 2)b^3 + \cdots + b^d = O(b^d)$

depth = 0

depth = d

(depth = 0) is searched up to (d+1) times

- Space complexity?
  - $O(bd)$

Refer to DFS

Pros of IDS

- (Like DFS) Its memory↓ DFS: $O(bm)$ IDS: $O(bd)$ $(d \leq m)$

- (Like BFS) It is **optimal** for problems where all actions have the same cost and is **complete** on finite acyclic state spaces (or on any finite state space when we check nodes for cycles all the way up the path)

- IDS is the preferred uninformed search method when
  1) the search state space is ~~large~~ larger than can fit in memory and
  2) the depth of the solution is not known