CS221 Problem Workout

Week 3

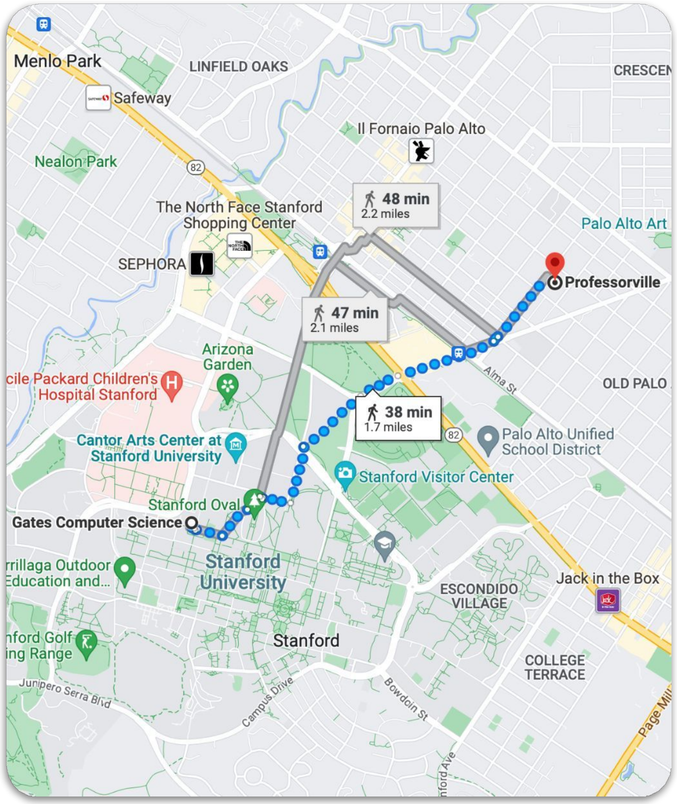
Stanford University

Search Problems

- Need to define:
  - States
    - Start State
  - Actions
  - Goals
  - Costs
  - Successors

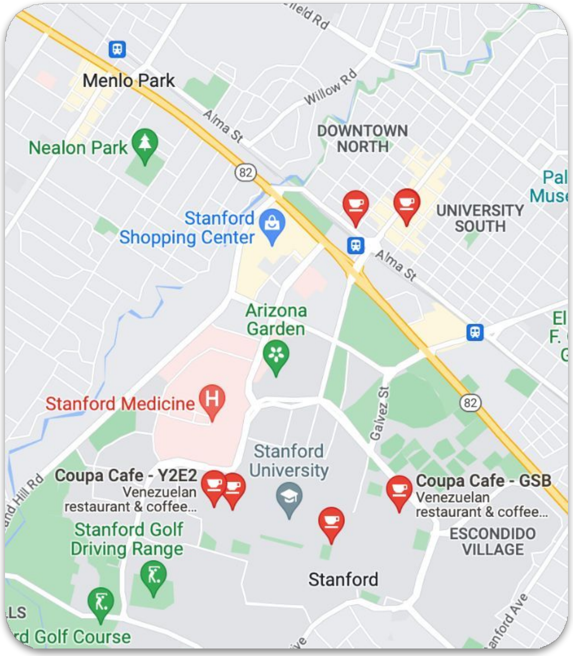
Objective?

Find a sequence of actions such that cost is minimized



States

- A state space contains all the possible configurations of the system.
- Each state tells you everything you need to know about "where you are" towards reaching your goal.



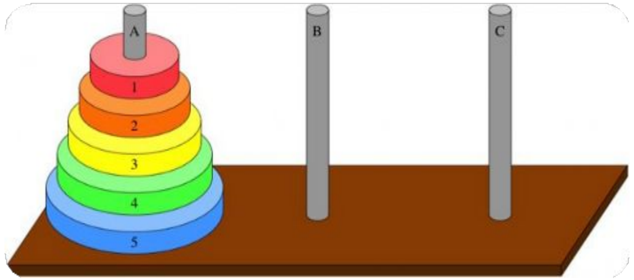


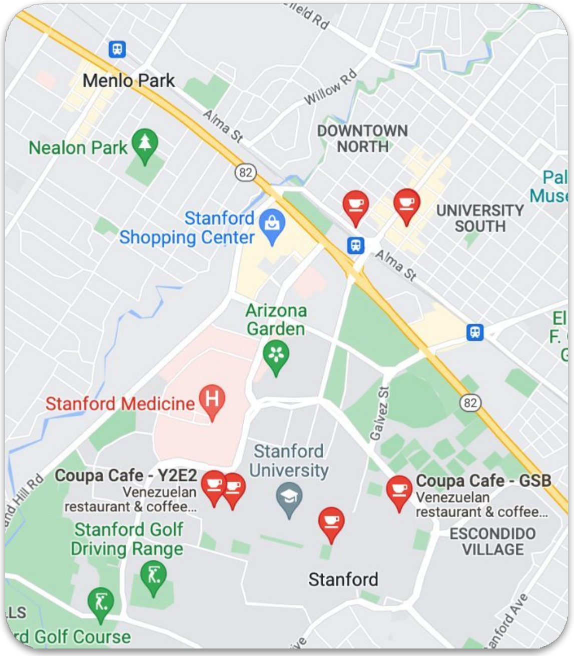
Image Credit: Khan Academy

States

- A state space contains all the possible configurations of the system.

- Each state tells you everything you need to know about "where you are" towards reaching your goal.

- Ex) Trying to visit every Coupa Cafe on campus.
  - State = (longitude, latitude, visitedY2E2?, visitedGSB?, visitedGreenLibrary?, …)



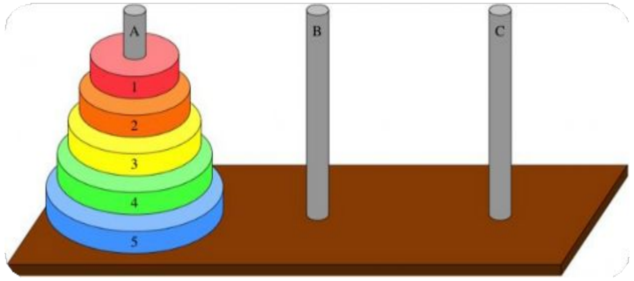
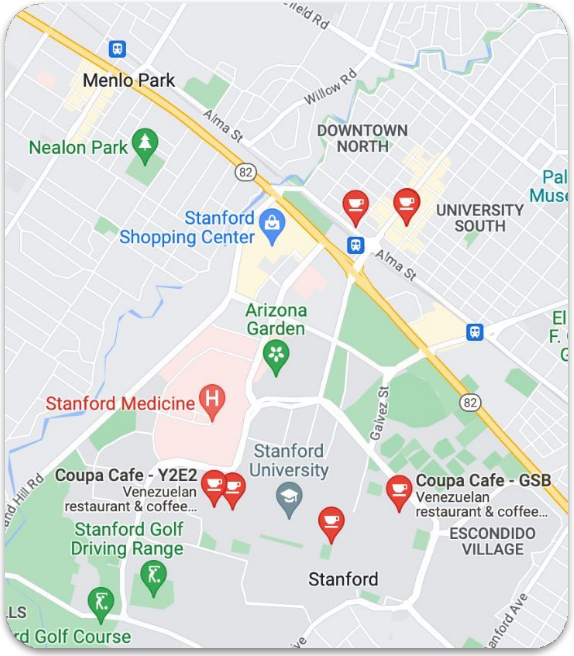


Image Credit: Khan Academy

States

- A state space contains all the possible configurations of the system.
- Each state tells you everything you need to know about “where you are” towards reaching your goal.
- Ex) Trying to visit every Coupa Cafe on campus.
    - State = (longitude, latitude, visitedY2E2?, visitedGSB?, visitedGreenLibrary?, …)
- Ex) Trying to solve towers of Hanoi
    - State = position and ordering of each of the blocks



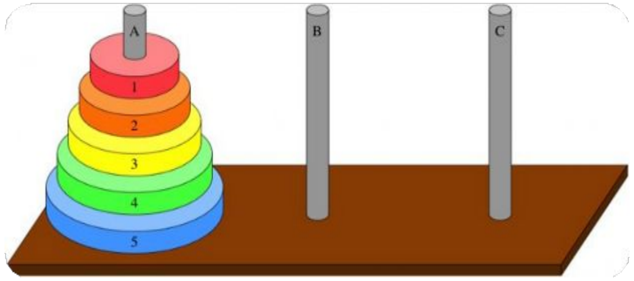


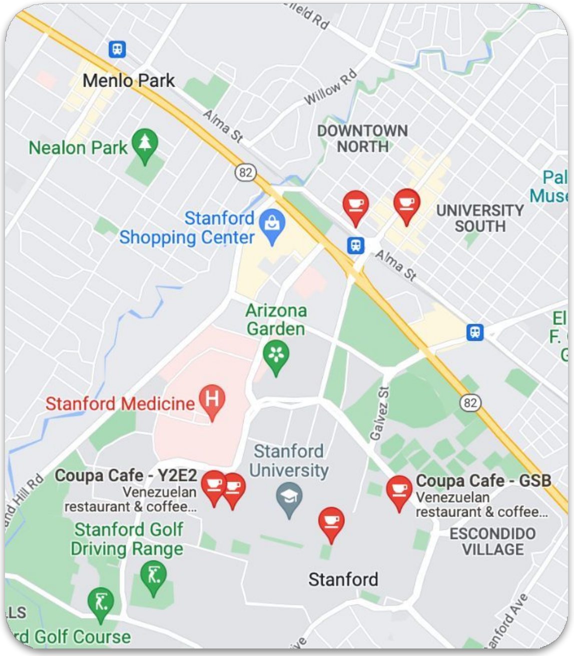
Image Credit: Khan Academy

Actions

- The action space describes all the possible things you can do to move from one state to another

- Ex) Trying to visit every Coupa Cafe on campus.
    - Walk North, Walk South, Walk East, Walk West, take the Margueritte from stop A to stop B, etc.

- Ex) Trying to solve towers of Hanoi
    - Move block from one pole to another



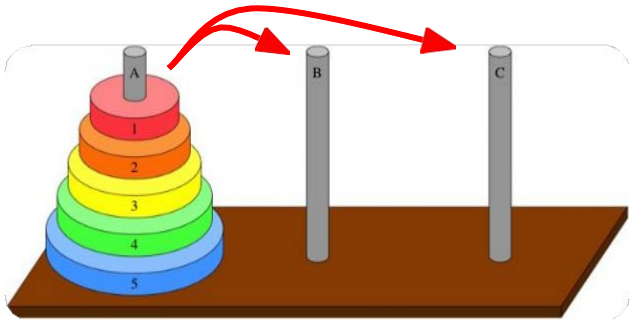
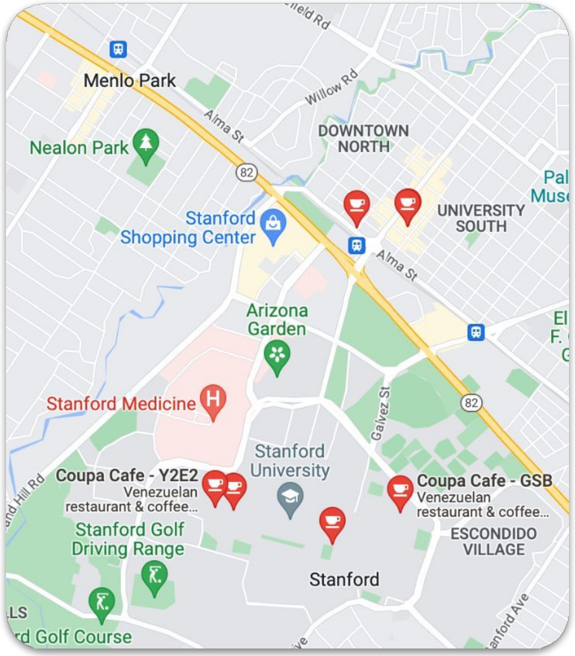


Image Credit: Khan Academy

Goals

- The goals decide what the end state of your search is.
- Possible to have more than one valid goal state.
- Ex) Trying to visit every Coupa Cafe on campus.
  - Visited every Coupa Cafe on campus
- Ex) Trying to solve towers of Hanoi
  - All blocks in ascending order by size on the final pole



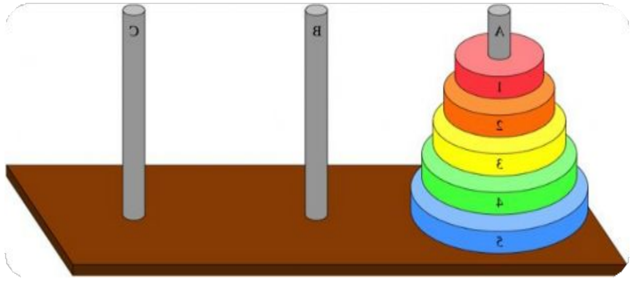
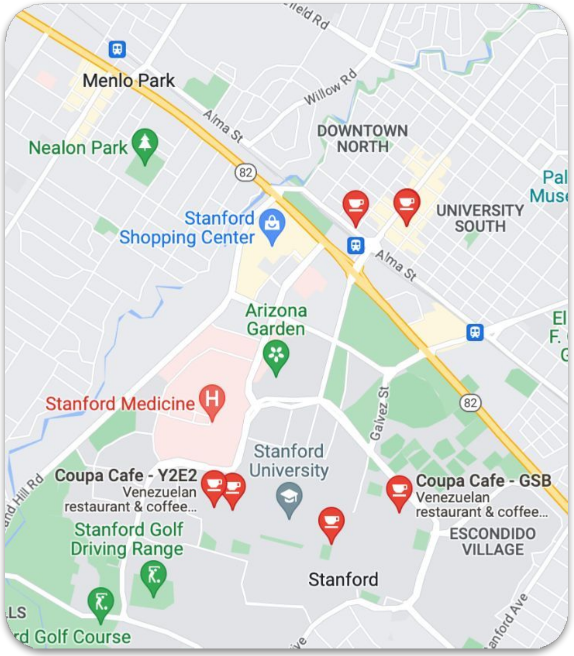


Image Credit: Khan Academy

Costs

- The costs assign a “price” to each action you take.
- Some search algorithms break under negative costs
  - BFS
  - UCS
  - A*
- Controls what you are optimizing for in the search
- Ex) Trying to visit every Coupa Cafe on campus.
  - Option 1: How long it will take to do an action (ex. 10 minutes on the marguerite)
  - Option 2: How far is the distance (ex. walk 100 meters north)
  - Option 3: Monetary cost (ex. $3 clipper card fare)
- Ex) Trying to solve towers of Hanoi
  - Can assign uniform cost to each action



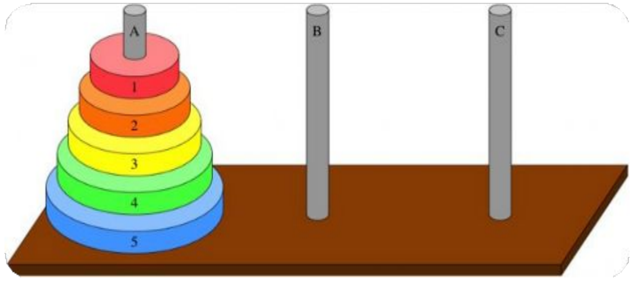
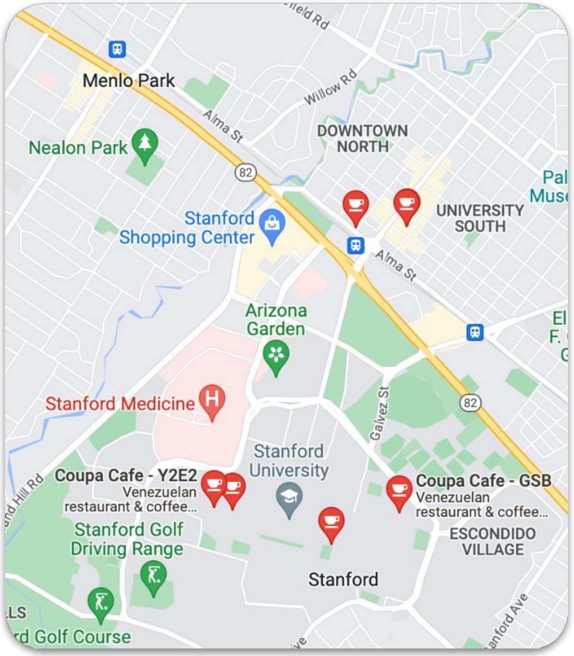


Image Credit: Khan Academy

Successors

- Defines the new state you are in from a current state after taking an action
- Successor(state, action) => new_state
- Ex) Trying to visit every Coupa Cafe on campus.
  - New longitude, New latitude, visitedY2E2?, visitedGSB?, visitedGreenLibrary?, ...
- Ex) Trying to solve towers of Hanoi
  - New order and positions of the blocks after the movement of a block



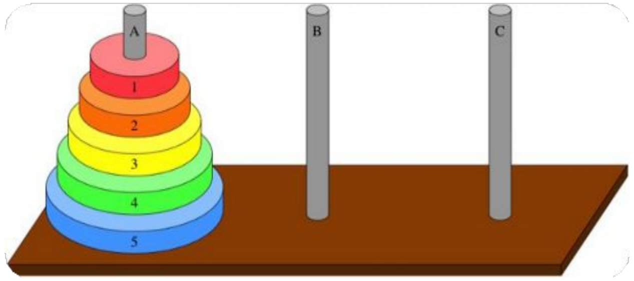
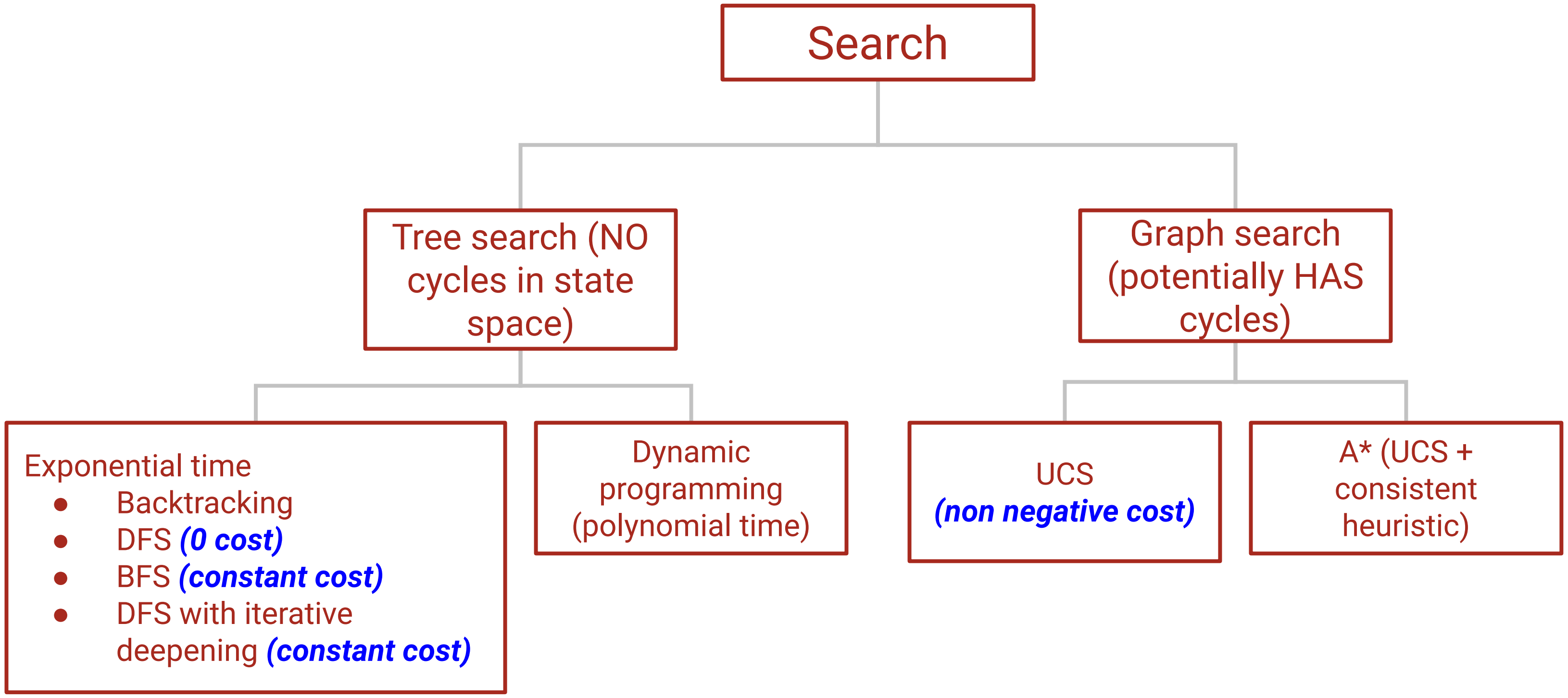


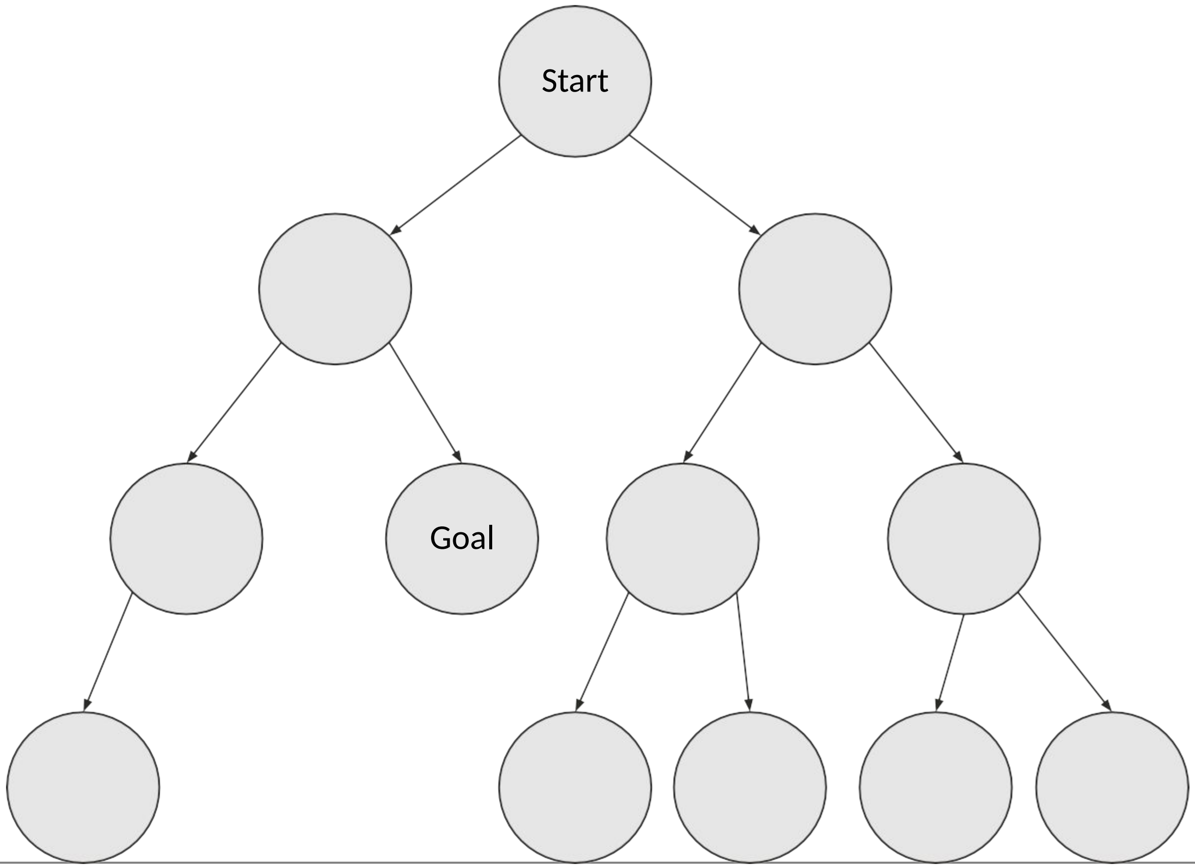
Image Credit: Khan Academy

How can we solve search problems?

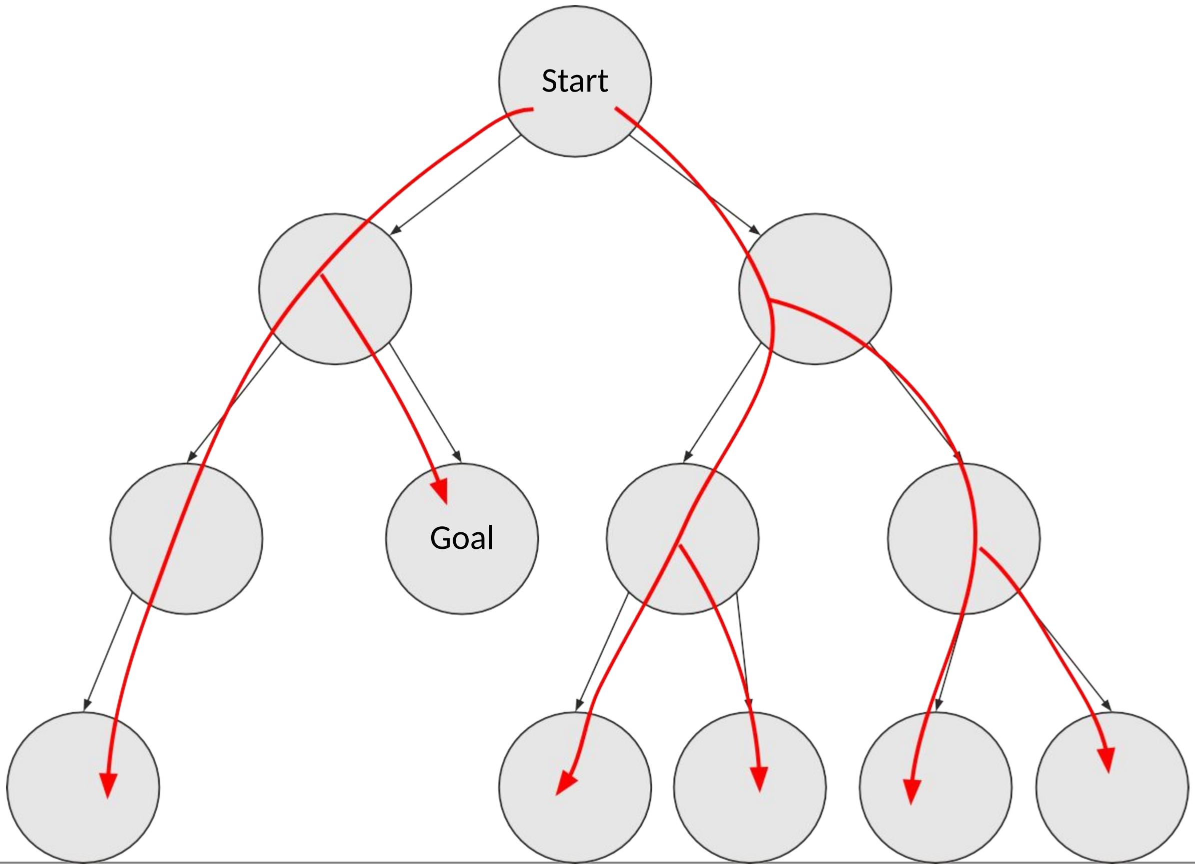

Search
Tree search (NO cycles in state space)
Graph search (potentially HAS cycles)
Exponential time
● Backtracking
● DFS *(0 cost)*
● BFS *(constant cost)*
● DFS with iterative deepening *(constant cost)*
Dynamic programming (polynomial time)
UCS *(non negative cost)*
A* (UCS + consistent heuristic)

Checkout https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-states-models for visualizations!

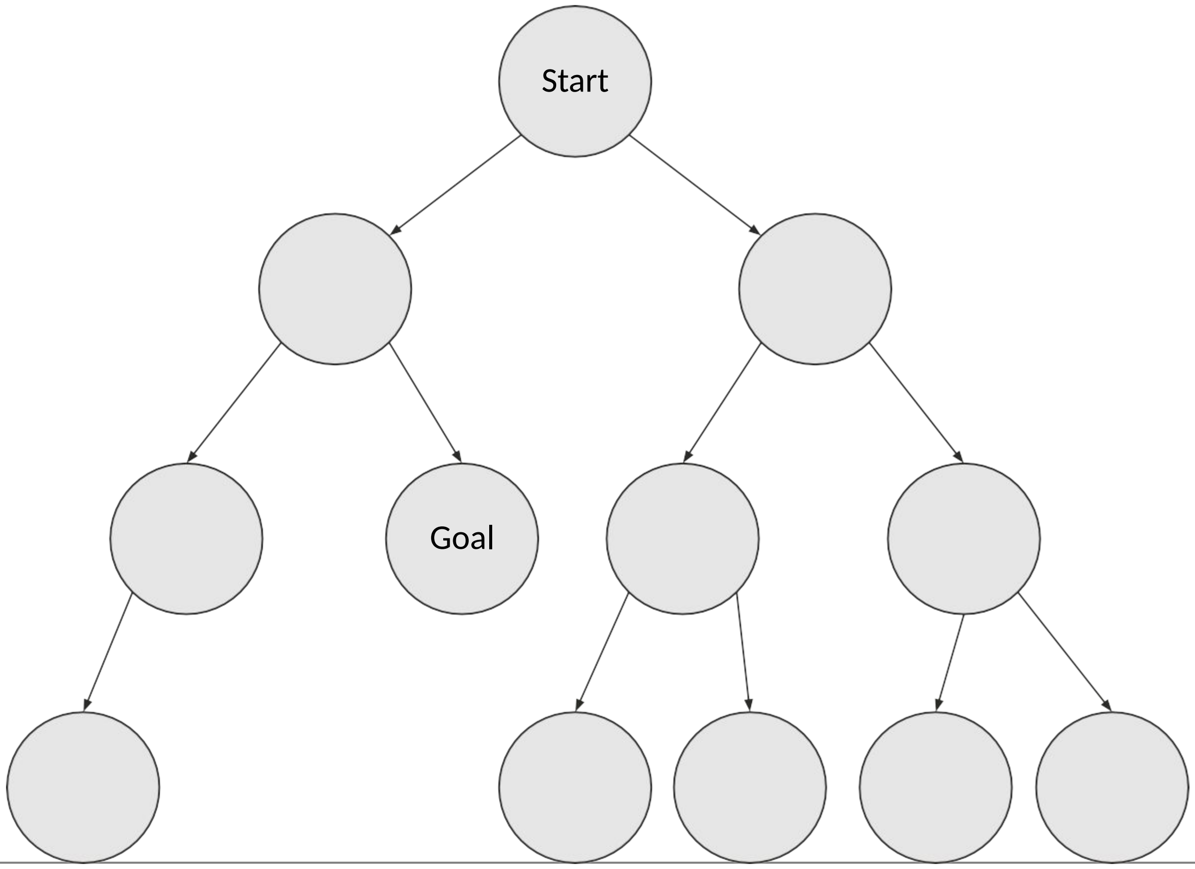
Backtracking Search



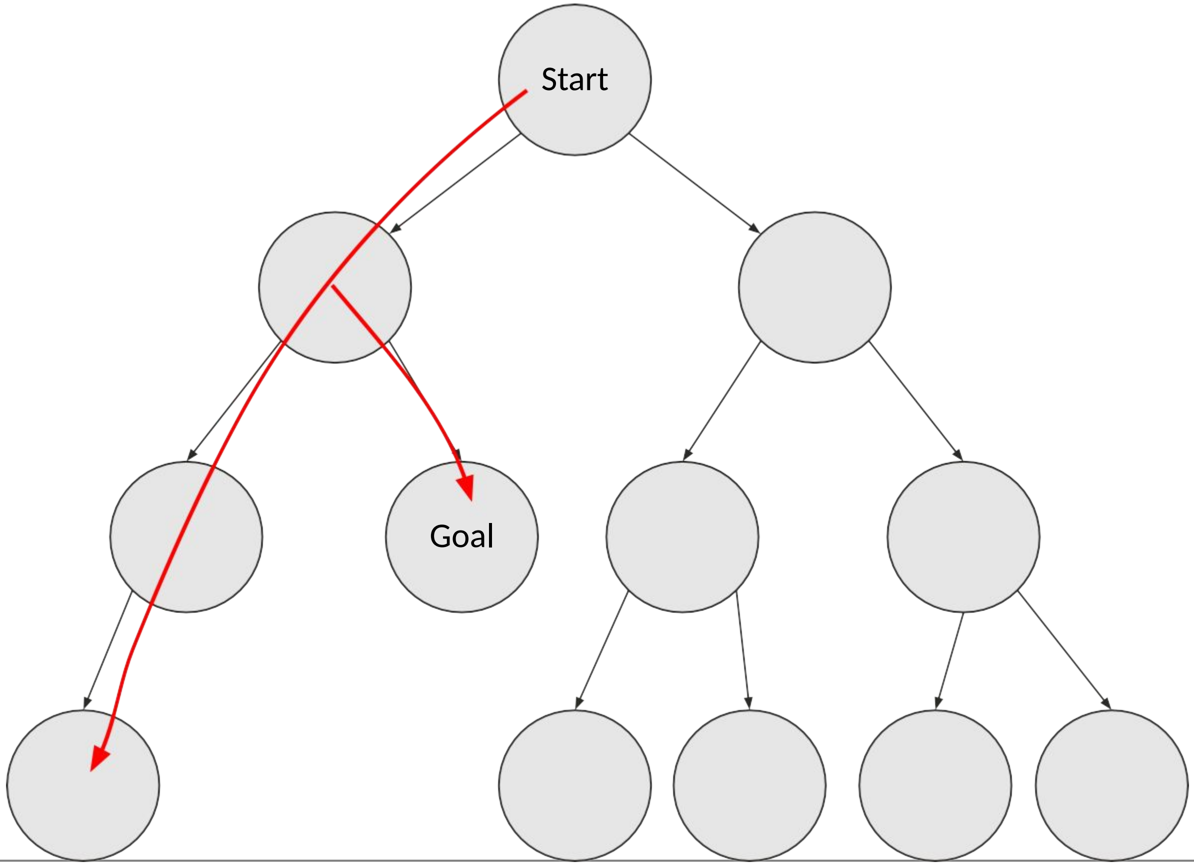
Backtracking Search



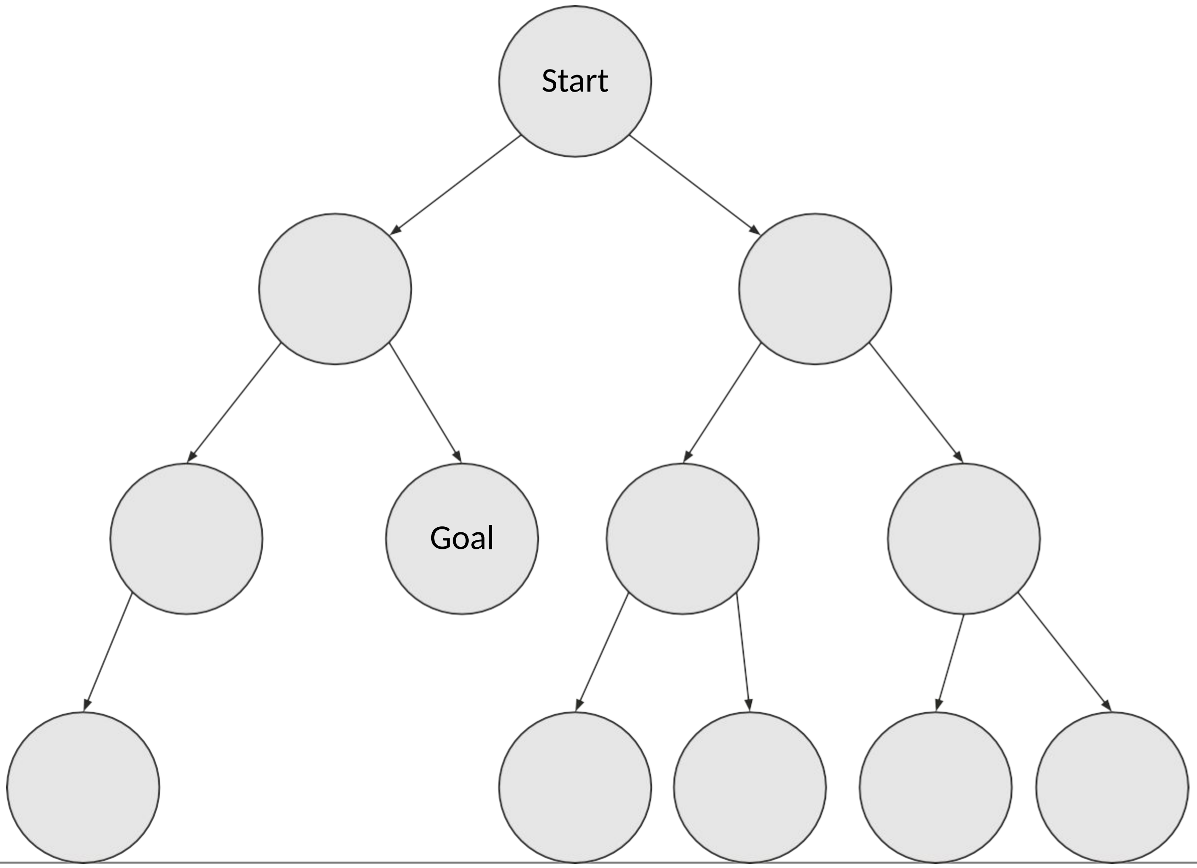
Depth First Search



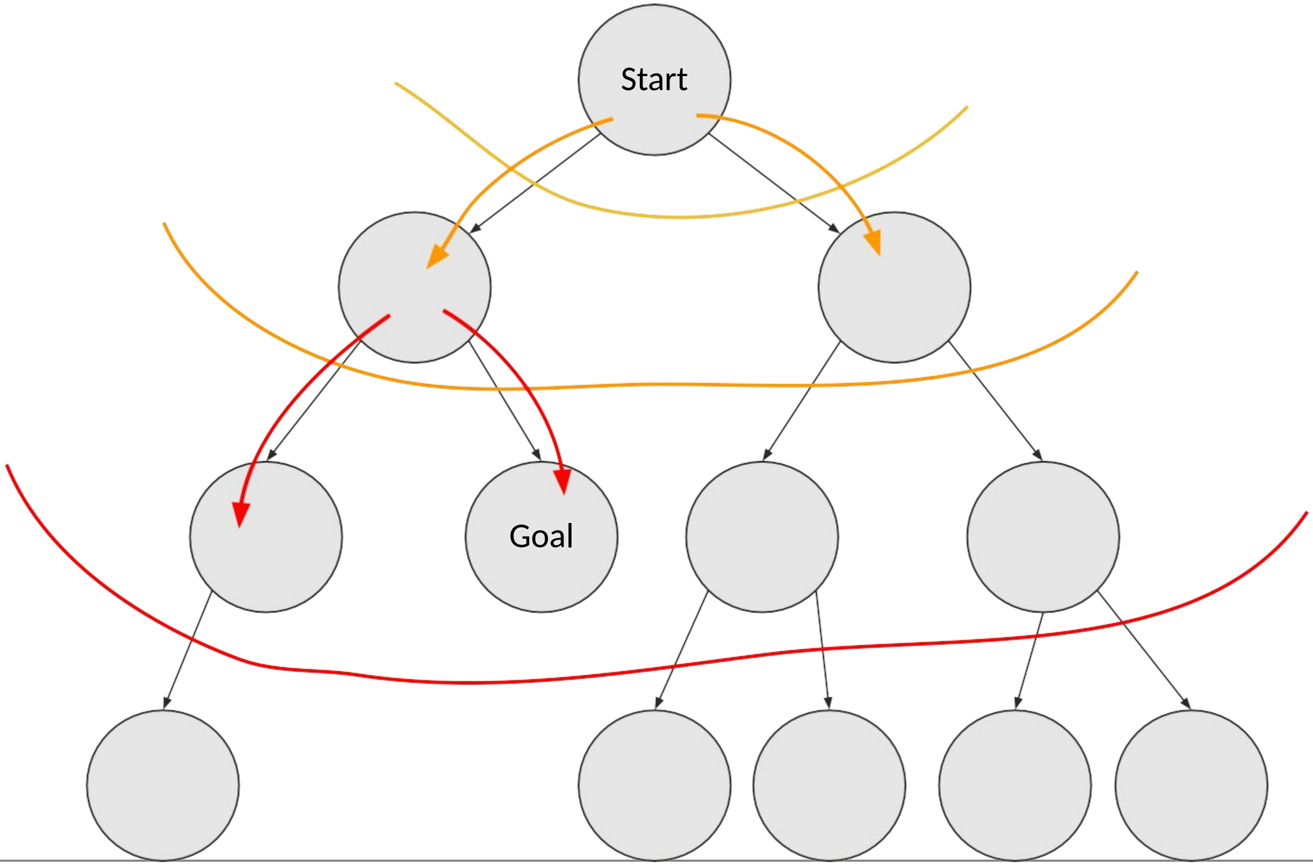
Depth First Search



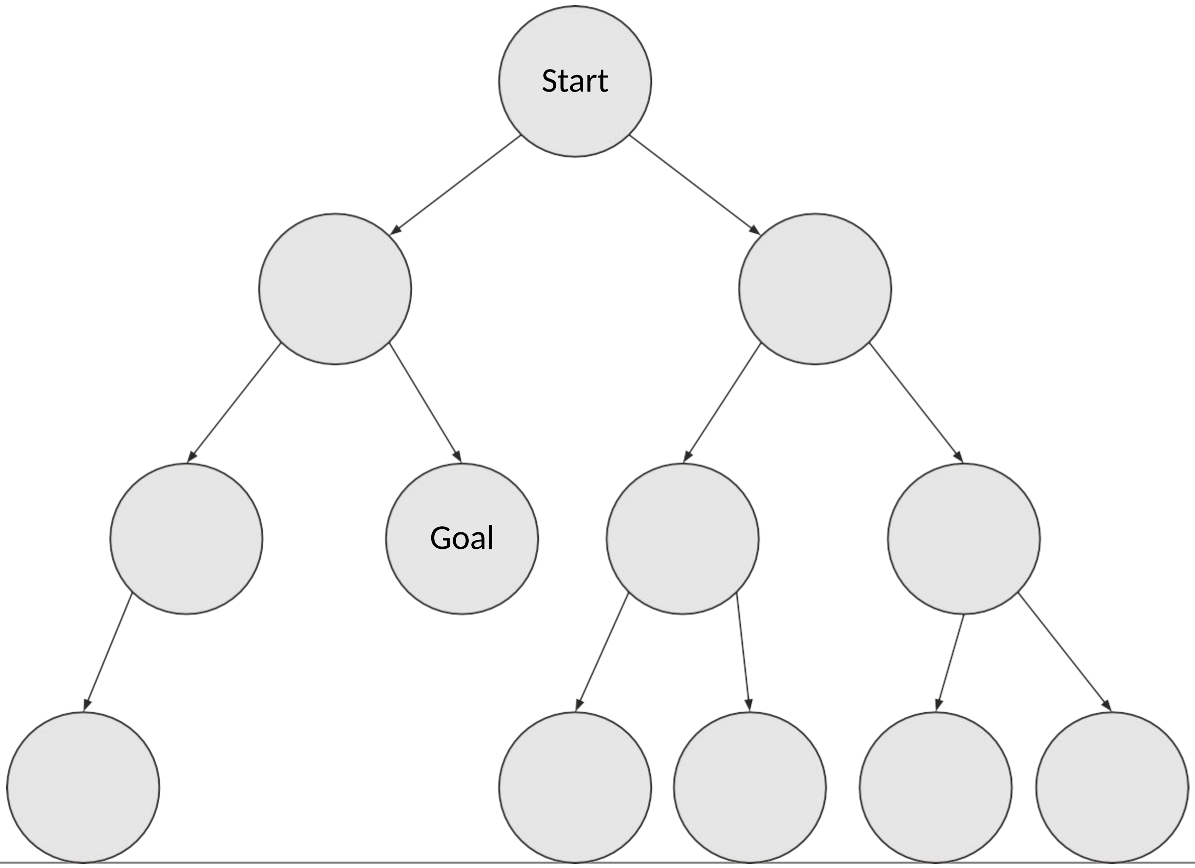
Breadth First Search



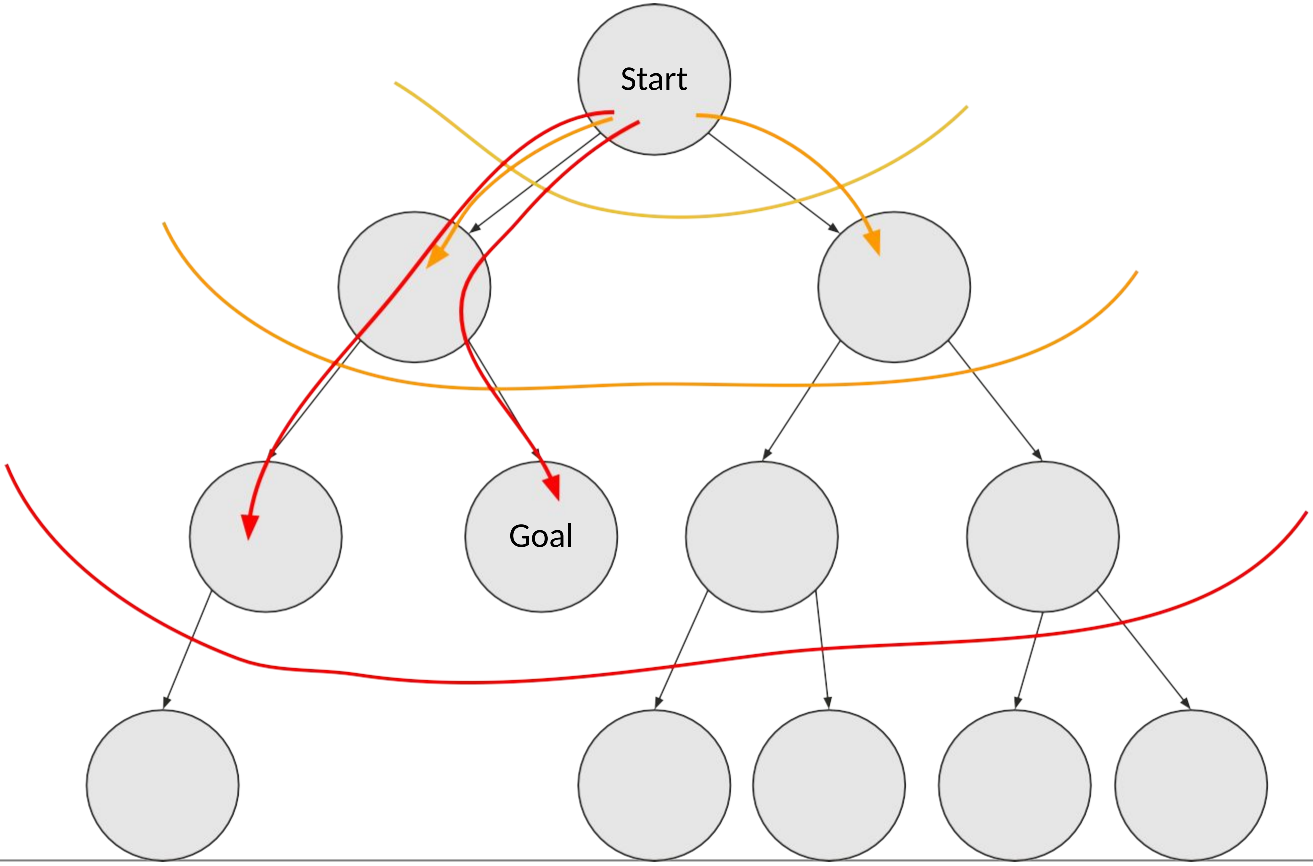
Breadth First Search



Iterative Deepening Depth First Search

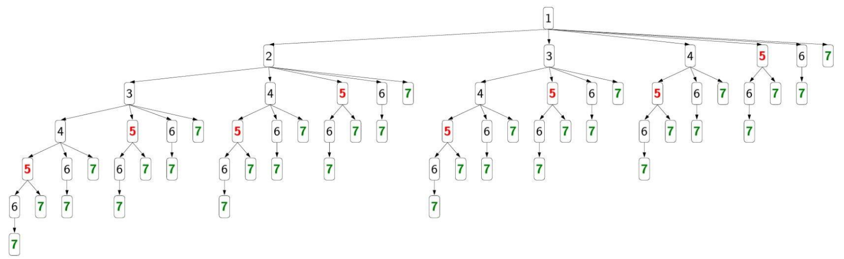


Iterative Deepening Depth First Search

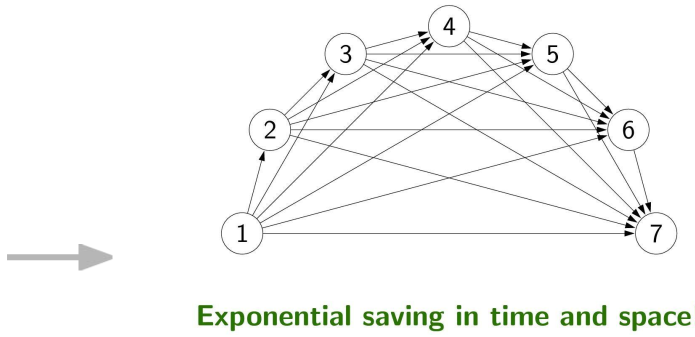


Dynamic Programming

- An algorithm that is akin to **backtracking search with memoization** and potentially exponential savings!
- The **states** in DP contain a summary of **past actions sufficient to choose future actions** optimally.



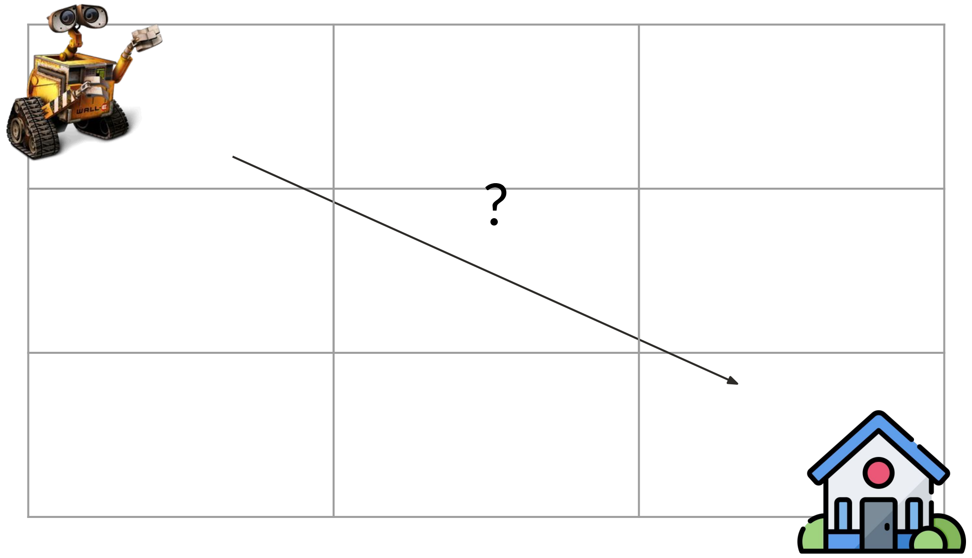
Backtracking (tree)



Dynamic Programming (graph)

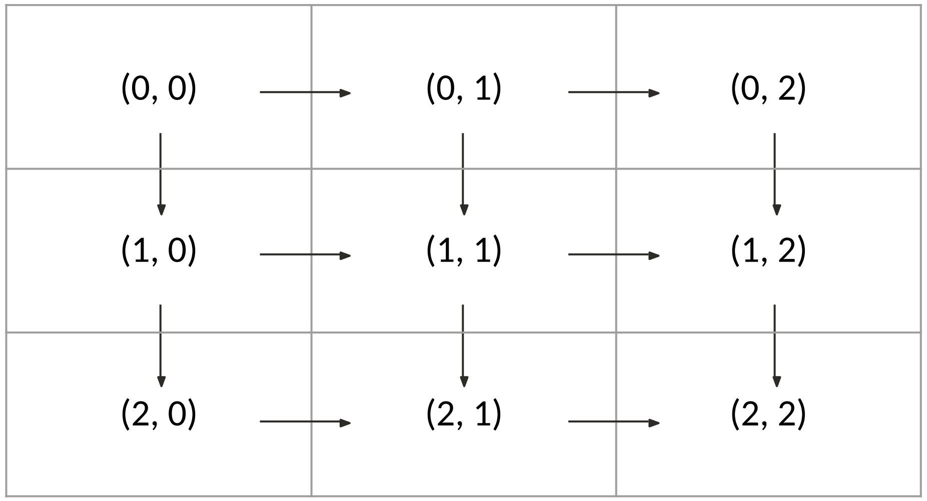
Dynamic Programming Example

- **Grid Dimensions**: The grid dimensions are m (rows) and n (columns)
- **Movement Constraints**: Wall-E can only move either down or to the right at any given point. It cannot move diagonally or backwards.
- **Problem**: find the number of unique paths the Wall-E can take to get home.



Dynamic Programming Example

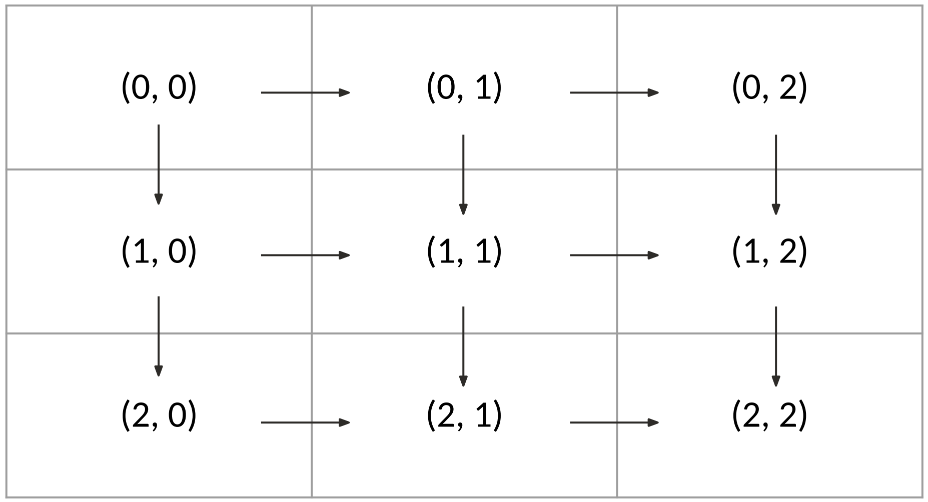
- **Nodes:** each cell is a node on the graph
- **Edges**: each edge is a possible path for the robot.
- **Ideas and Intuition:** ?



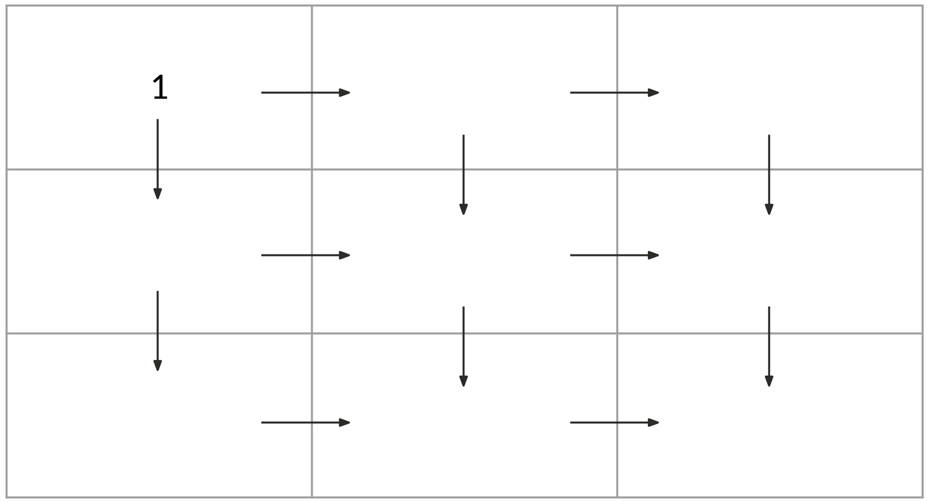
$$\begin{tikzcd}
(0, 0) \arrow[r] \arrow[d] & (0, 1) \arrow[r] \arrow[d] & (0, 2) \arrow[d] \\
(1, 0) \arrow[r] \arrow[d] & (1, 1) \arrow[r] \arrow[d] & (1, 2) \arrow[d] \\
(2, 0) \arrow[r] & (2, 1) \arrow[r] & (2, 2)
\end{tikzcd}$$

Dynamic Programming Example

- **Ideas and Intuition:** use a 2D array to store the number of unique paths to each cell. A cell (i,j) can be reached either from (i−1,j) or (i,j−1), and thus the number of unique paths to (i,j) is the sum of the number of unique paths to these two cells.



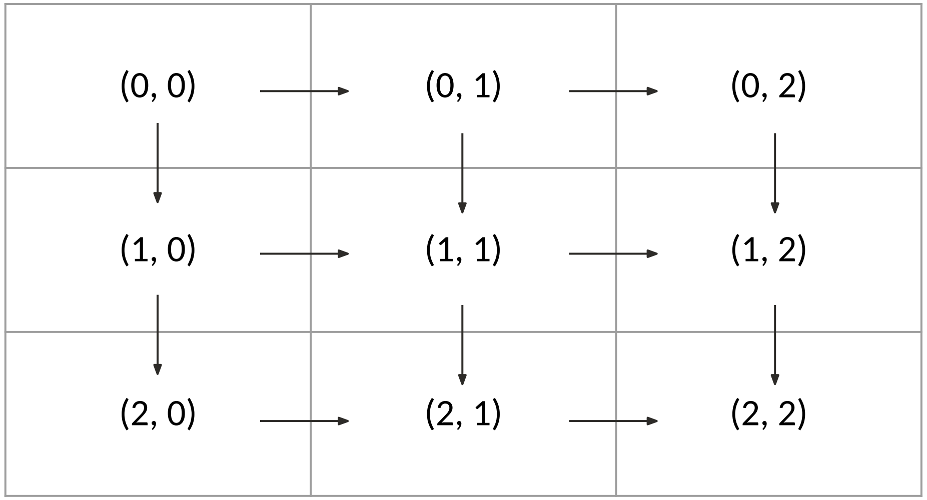
Original Grid



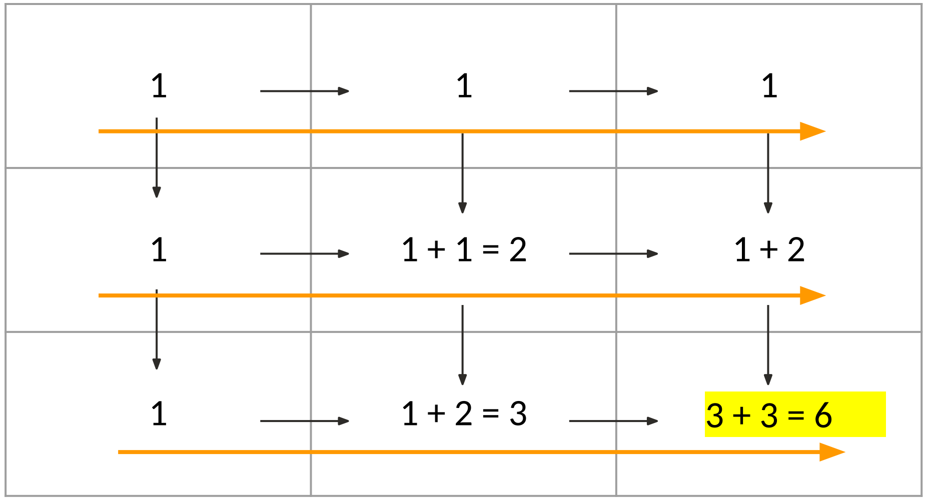
Dynamic Programming 2D array

Dynamic Programming Example

- **Ideas and Intuition:** use a 2D array to store the number of unique paths to each cell. A cell (i,j) can be reached either from (i−1,j) or (i,j−1), and thus the number of unique paths to (i,j) is the sum of the number of unique paths to these two cells.



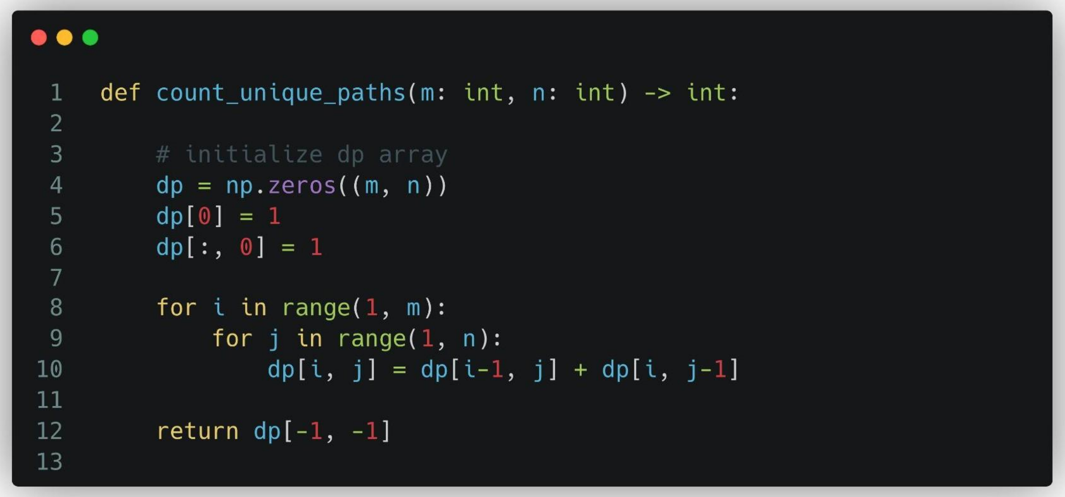
Original Grid



Dynamic Programming 2D array

Dynamic Programming Example

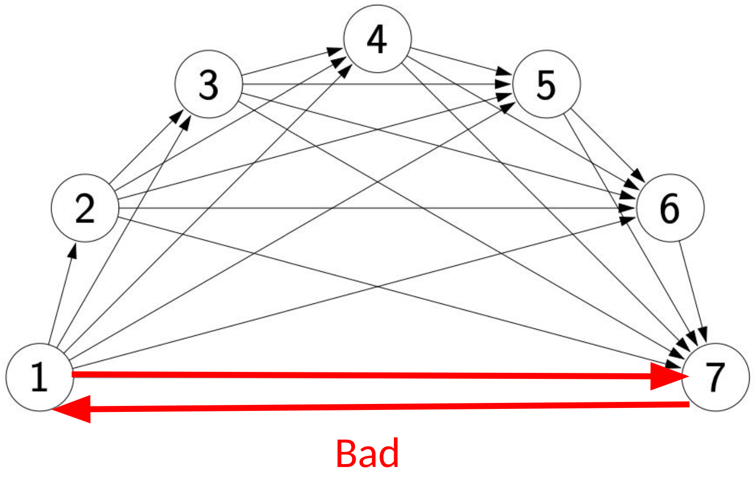
- **Ideas and Intuition:** use a 2D array to store the number of unique paths to each cell. A cell (i,j) can be reached either from (i−1,j) or (i,j−1), and thus the number of unique paths to (i,j) is the sum of the number of unique paths to these two cells.

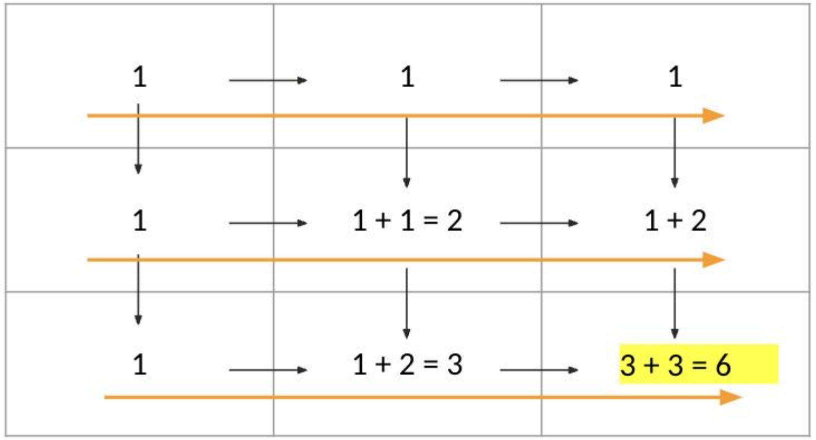


```python
def count_unique_paths(m: int, n: int) -> int:

    # initialize dp array
    dp = np.zeros((m, n))
    dp[0] = 1
    dp[:, 0] = 1

    for i in range(1, m):
        for j in range(1, n):
            dp[i, j] = dp[i-1, j] + dp[i, j-1]

    return dp[-1, -1]

```

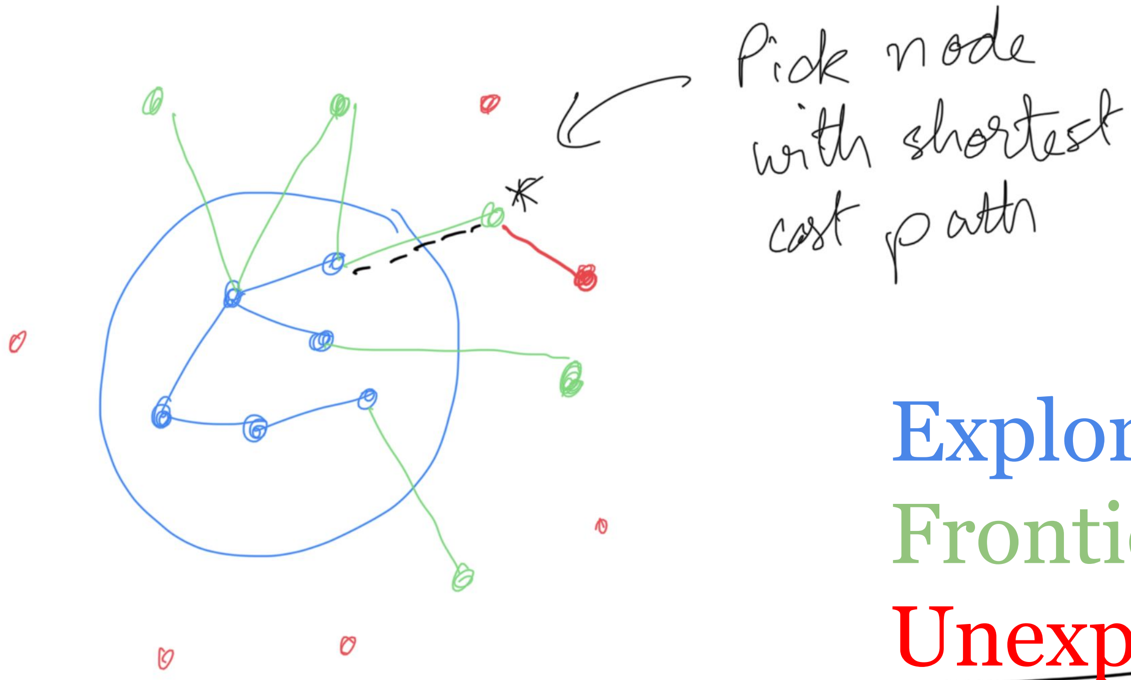
DP does not work if there are cycles

- DP requires the search tree to be a Directed Acyclic Graph (DAG)
- This is because we need an ordering to fill out the entries in the memo; otherwise, we would not know where to start!



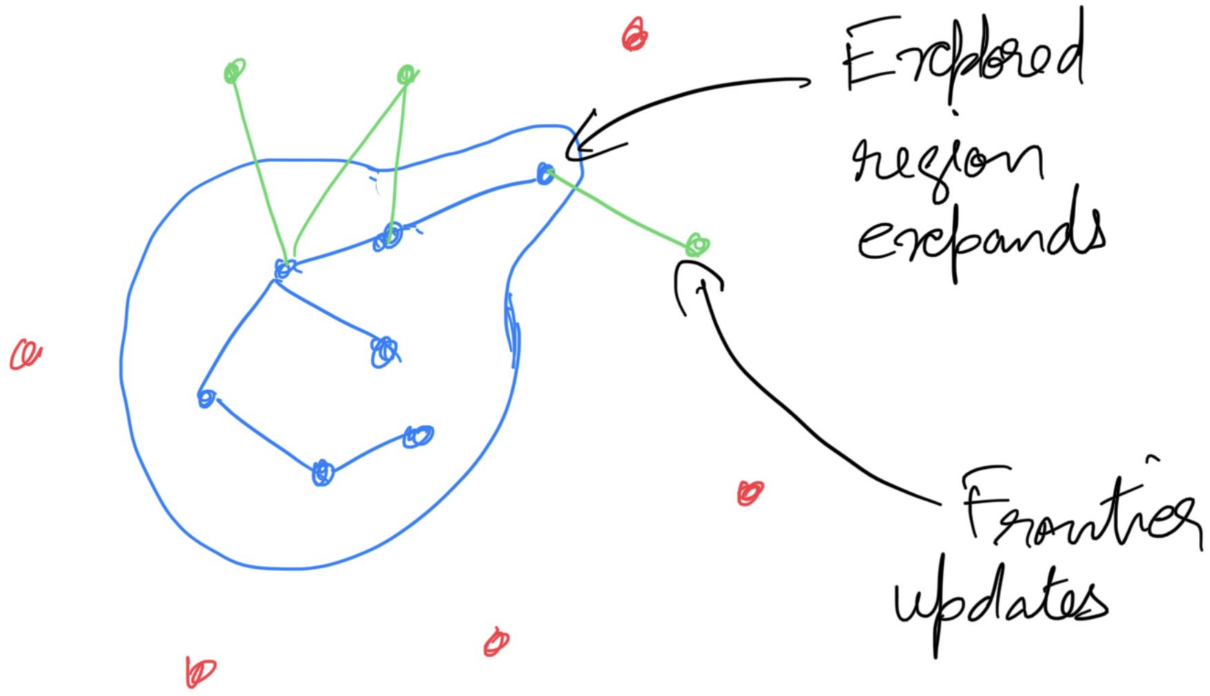


Uniform Cost Search

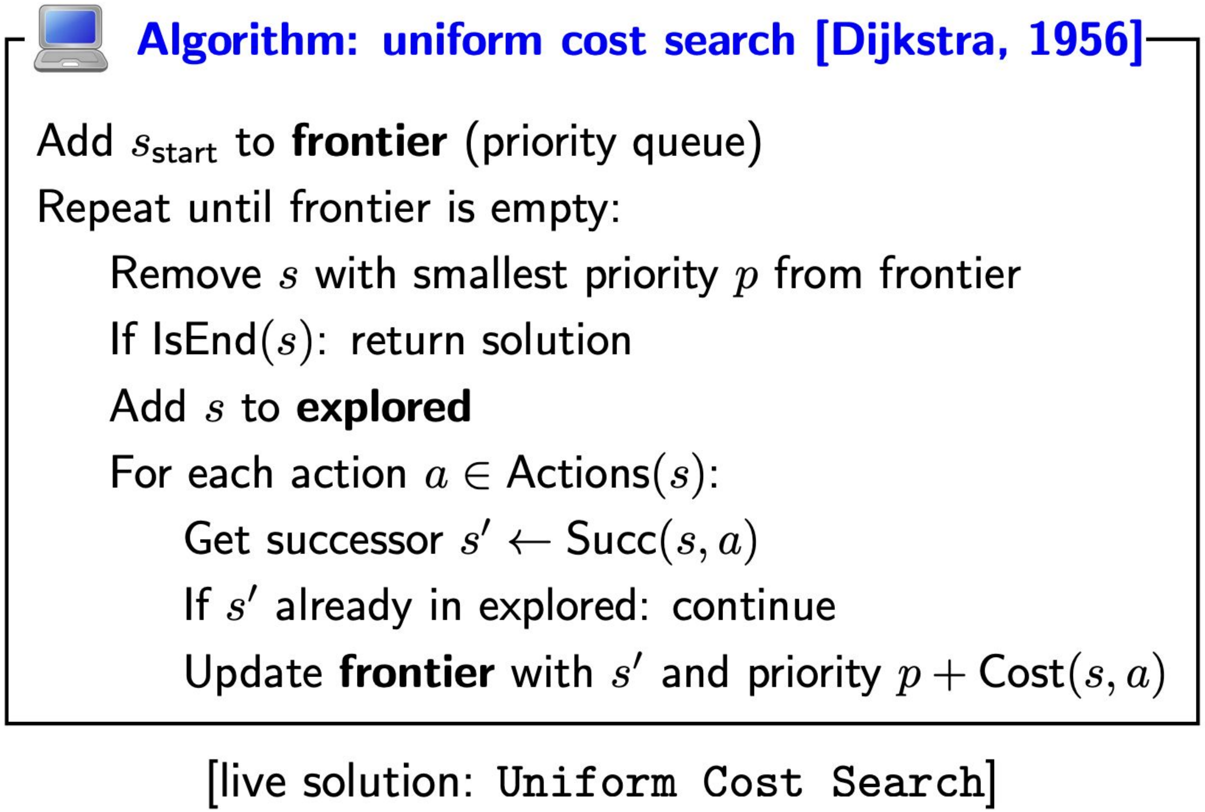


Explored
Frontier
Unexplored

Uniform Cost Search



UCS - Pseudocode *(from lecture slides)*



Algorithm: uniform cost search [Dijkstra, 1956]

```
Add $s_{\text{start}}$ to **frontier** (priority queue)
Repeat until frontier is empty:
    Remove $s$ with smallest priority $p$ from frontier
    If IsEnd$(s)$: return solution
    Add $s$ to **explored**
    For each action $a \in$ Actions$(s)$:
        Get successor $s' \leftarrow$ Succ$(s, a)$
        If $s'$ already in explored: continue
        Update **frontier** with $s'$ and priority $p +$ Cost$(s, a)$
```

[live solution: `Uniform Cost Search`]

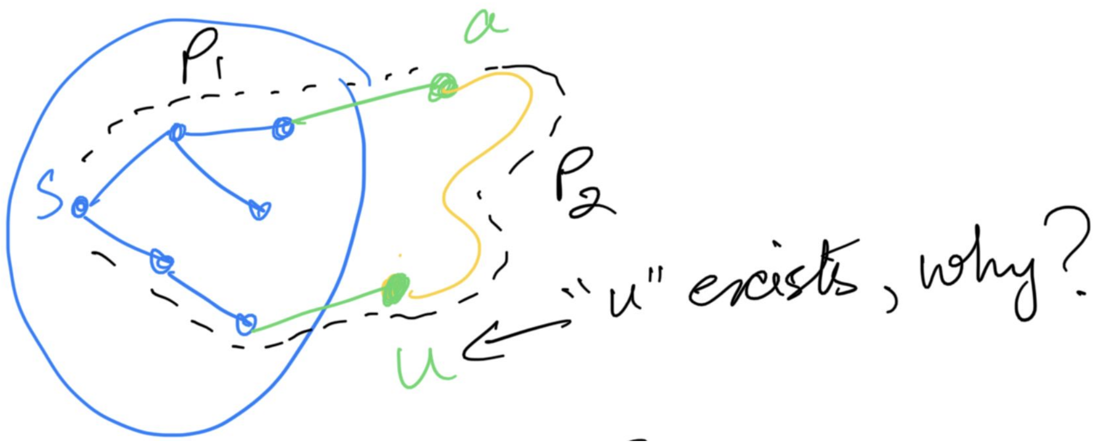
UCS - Proof of Correctness

Claim: When node "a" gets added to EXPLORED region, the path is the shortest path.

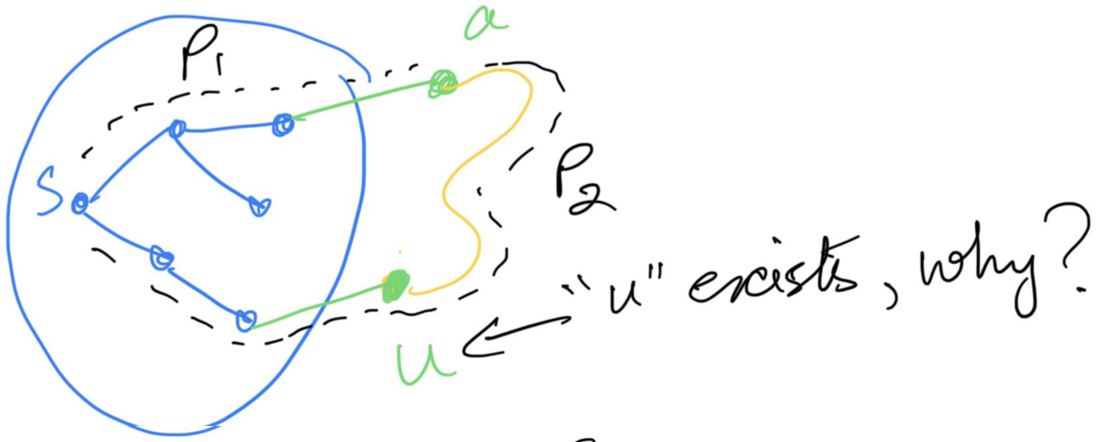
UCS - Proof of Correctness

<u>Proof</u>: Lets say not!

So there is another shorter path to "a" from "s".



UCS - Proof of Correctness



$$c^{P_1}(s,a) > c^{P_2}(s,a)$$

$$c^{P_2}(s,u) < c^{P_2}(s,a)$$
$$< c^{P_1}(s,a)$$

UCS - Proof of Correctness

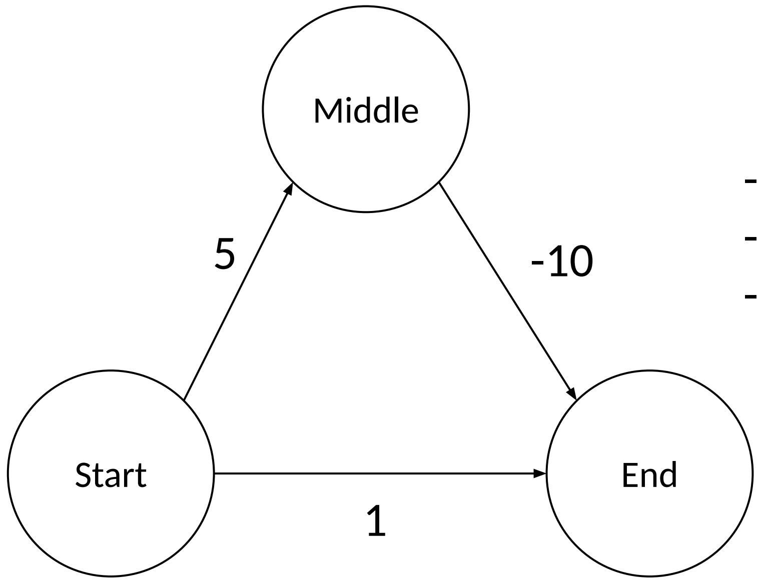
$$c^{P_1}(s,a) > c^{P_2}(s,a)$$

$$c^{P_2}(s,u) < c^{P_2}(s,a) < c^{P_1}(s,a)$$

So, "u" should be added to EXPLORED Region before adding "a"

Contradiction!

UCS does not work if there are negative edges



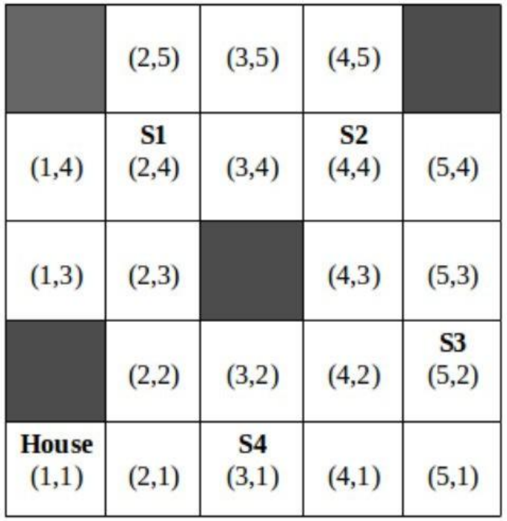
- Optimal: start -> middle -> end with a cost of -5
- UCS finds: start -> end with a cost of 1
- Always try to come up with your own examples
    - The simpler the example, the better!

Problem (1a)

- Describe A:
- s_{start} =
- Actions((x,y,A)) = {N,S,E,W}
- Succ((x,y,A),a) =
- Cost((x,y,A),a) =
- IsGoal((x,y,A)) =

1) **Problem 1**

Sabina has just moved to a new town, which is represented as a grid of locations (see below). She needs to visit various shops $S_1, \dots, S_k$. From a location on the grid, Sabina can move to the location that is immediately north, south, east, or west, but certain locations have been blocked off and she cannot enter them. It takes one unit of time to move between adjacent locations. Here is an example layout of the town:

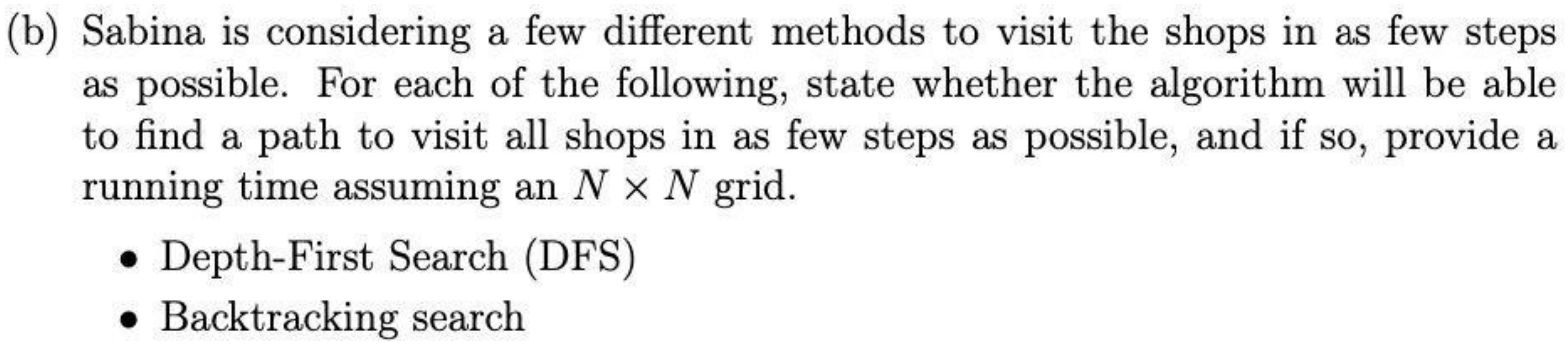


▨	(2,5)	(3,5)	(4,5)	▨
(1,4)	**S1** (2,4)	(3,4)	**S2** (4,4)	(5,4)
(1,3)	(2,3)	▨	(4,3)	(5,3)
▨	(2,2)	(3,2)	(4,2)	**S3** (5,2)
House (1,1)	(2,1)	**S4** (3,1)	(4,1)	(5,1)

Sabina lives at (1, 1), and no location contains more than one building (Sabina's house or a shop).

(a) Sabina wants to start at her house, visit the shops $S_1, \dots, S_k$ **in any order**, and then return to her house as quickly as possible. We will construct a search problem to find the fastest route for Sabina. Each state is modeled as a tuple $s = (x, y, A)$, where (x, y) is Sabina's current position, and A is some auxiliary information that you need to choose. If an action is invalid from a given state, set its cost to infinity. Let V be the set of valid (non-blocked) locations; use this to define your search problem. You may assume that the locations of the k shops are known. You must choose a minimal representation of A and solve this problem for general k. Be precise!

Problem (1b)



(b) Sabina is considering a few different methods to visit the shops in as few steps as possible. For each of the following, state whether the algorithm will be able to find a path to visit all shops in as few steps as possible, and if so, provide a running time assuming an $N \times N$ grid.

- Depth-First Search (DFS)
- Backtracking search

Looking Ahead: Heuristics

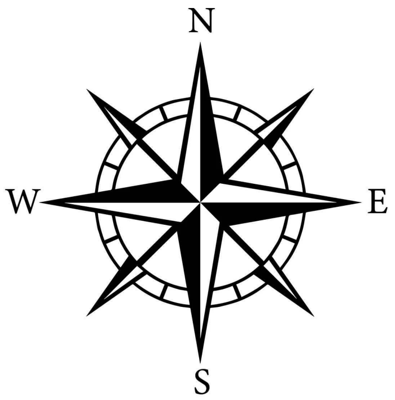
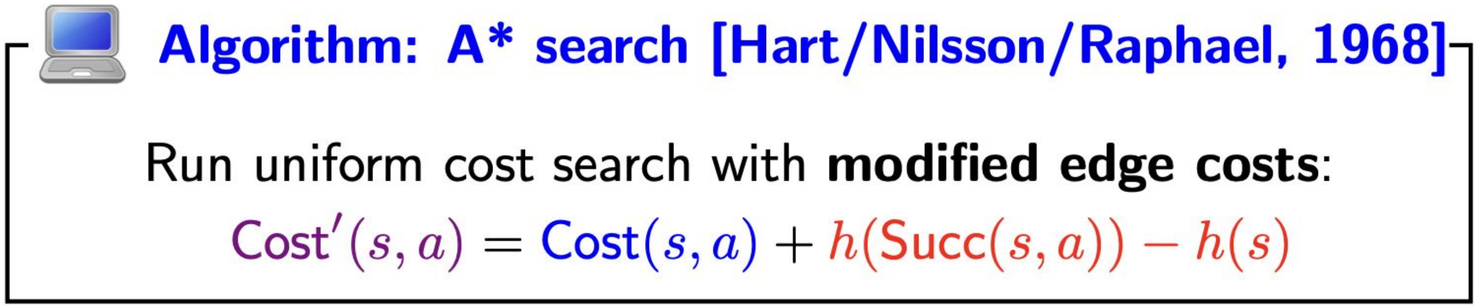


Image Credit: Cleveland.com

A*: UCS with heuristics

- A* is an expansion of UCS, but we use an estimate of “future cost”
- This should give us a better estimate of **total cost to END = past cost + future cost**
    - Leads to more efficient search!



Algorithm: A* search [Hart/Nilsson/Raphael, 1968]

Run uniform cost search with **modified edge costs**:

$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

Requirements of a good heuristic for A*:

- h(Succ(s, a)) - h(s) should a measurement of whether we are getting closer to END
- We will run UCS on the new costs, so the new costs have to be non-negative

Consistent Heuristic

- We will run UCS on the new costs, so the new costs have to be non-negative

Definition: consistency

A heuristic h is **consistent** if

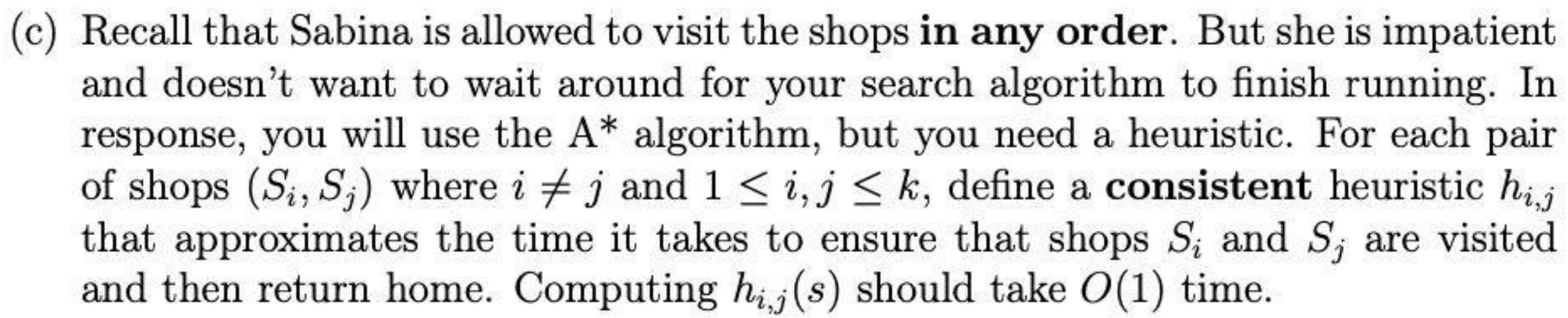
- $\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s) \geq 0$
- $h(s_{\text{end}}) = 0.$

- If the new costs are non-negative, UCS would return the correct result

Proposition: correctness

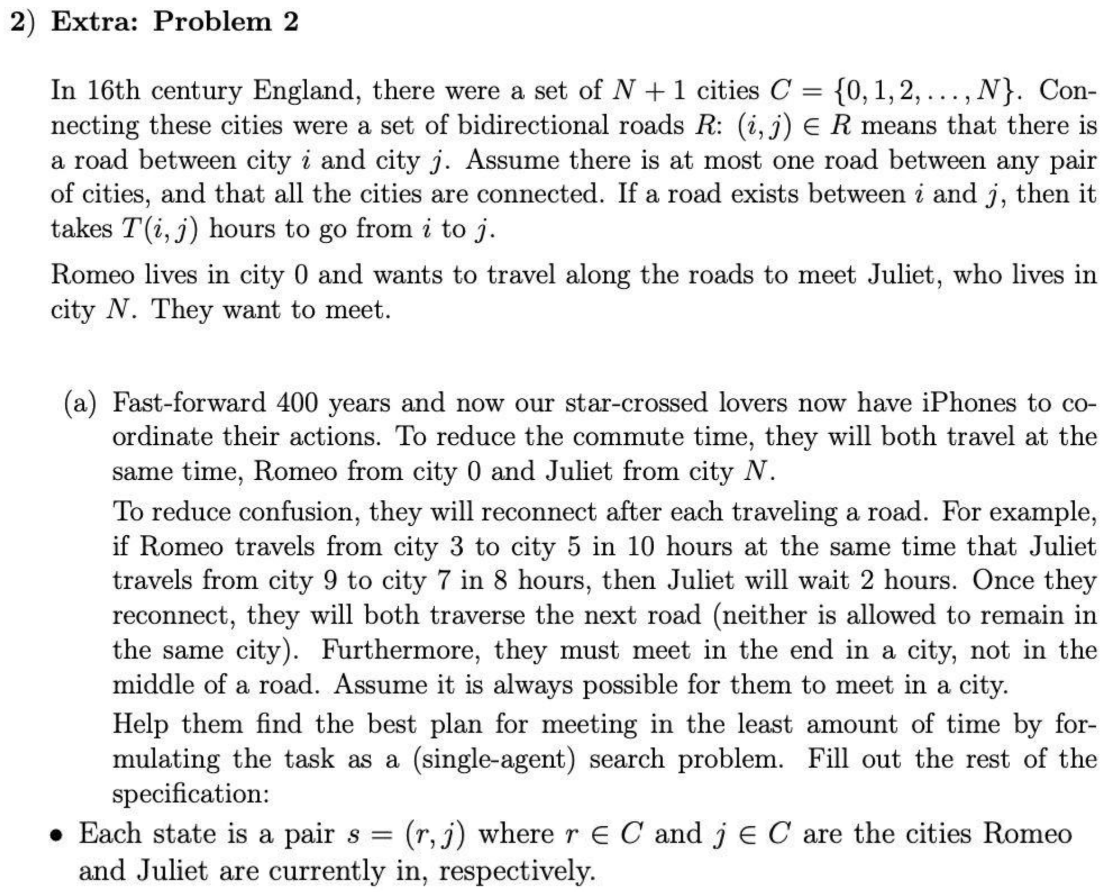
If h is consistent, A* returns the minimum cost path.

Problem (1c)



(c) Recall that Sabina is allowed to visit the shops **in any order**. But she is impatient and doesn't want to wait around for your search algorithm to finish running. In response, you will use the A* algorithm, but you need a heuristic. For each pair of shops (S_i, S_j) where $i \neq j$ and $1 \leq i, j \leq k$, define a **consistent** heuristic $h_{i,j}$ that approximates the time it takes to ensure that shops S_i and S_j are visited and then return home. Computing $h_{i,j}(s)$ should take $O(1)$ time.

Problem (2a)



2) Extra: Problem 2

In 16th century England, there were a set of $N+1$ cities $C = \{0, 1, 2, \ldots, N\}$. Connecting these cities were a set of bidirectional roads R: $(i, j) \in R$ means that there is a road between city i and city j. Assume there is at most one road between any pair of cities, and that all the cities are connected. If a road exists between i and j, then it takes $T(i, j)$ hours to go from i to j.

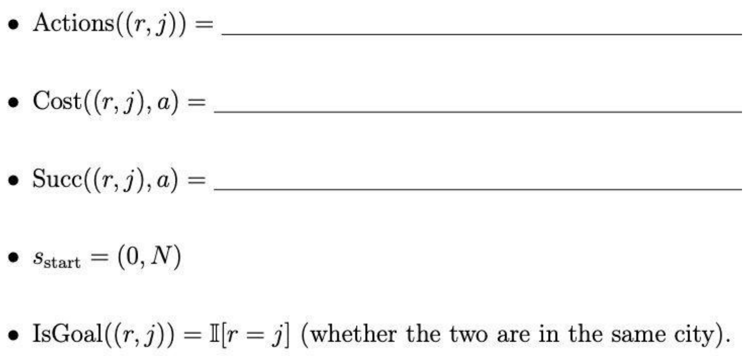
Romeo lives in city 0 and wants to travel along the roads to meet Juliet, who lives in city N. They want to meet.

(a) Fast-forward 400 years and now our star-crossed lovers now have iPhones to co-ordinate their actions. To reduce the commute time, they will both travel at the same time, Romeo from city 0 and Juliet from city N.

To reduce confusion, they will reconnect after each traveling a road. For example, if Romeo travels from city 3 to city 5 in 10 hours at the same time that Juliet travels from city 9 to city 7 in 8 hours, then Juliet will wait 2 hours. Once they reconnect, they will both traverse the next road (neither is allowed to remain in the same city). Furthermore, they must meet in the end in a city, not in the middle of a road. Assume it is always possible for them to meet in a city.

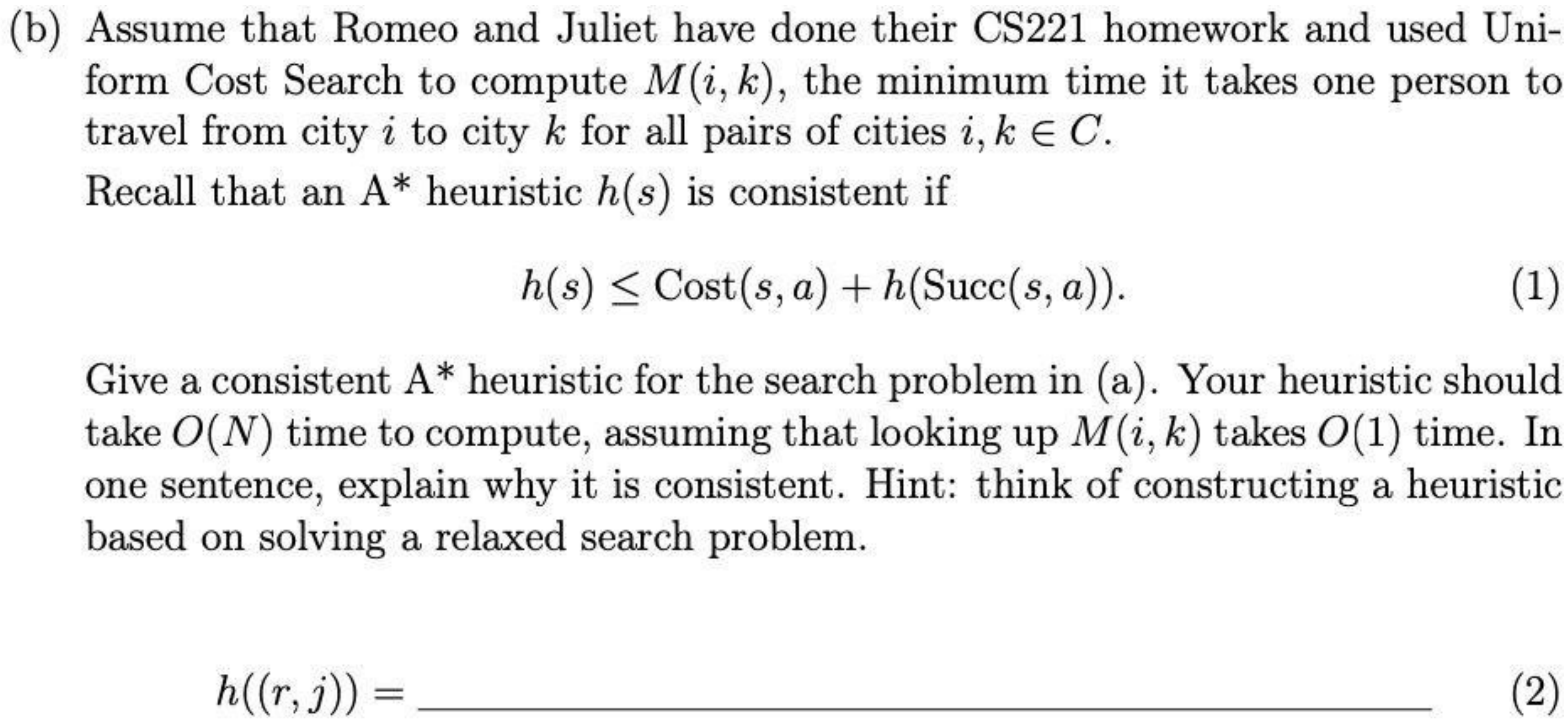
Help them find the best plan for meeting in the least amount of time by formulating the task as a (single-agent) search problem. Fill out the rest of the specification:

- Each state is a pair $s = (r, j)$ where $r \in C$ and $j \in C$ are the cities Romeo and Juliet are currently in, respectively.



- $\text{Actions}((r, j)) =$ ______________________
- $\text{Cost}((r, j), a) =$ ______________________
- $\text{Succ}((r, j), a) =$ ______________________
- $s_{\text{start}} = (0, N)$
- $\text{IsGoal}((r, j)) = \mathbb{I}[r = j]$ (whether the two are in the same city).

Problem (2b)



(b) Assume that Romeo and Juliet have done their CS221 homework and used Uniform Cost Search to compute $M(i,k)$, the minimum time it takes one person to travel from city i to city k for all pairs of cities $i,k \in C$.

Recall that an A* heuristic $h(s)$ is consistent if

$$h(s) \leq \text{Cost}(s,a) + h(\text{Succ}(s,a)). \tag{1}$$

Give a consistent A* heuristic for the search problem in (a). Your heuristic should take $O(N)$ time to compute, assuming that looking up $M(i,k)$ takes $O(1)$ time. In one sentence, explain why it is consistent. Hint: think of constructing a heuristic based on solving a relaxed search problem.

$$h((r,j)) = \underline{\hspace{10cm}} \tag{2}$$

Thank You

Stanford University

Come to our office hours!

Jeremy Kim

HW OH: Fridays	11:00am-12:30pm Huang
HW OH: Sundays	2:00pm-3:30pm Online

Joey O'Brien

HW OH: Tuesdays	9:00am to 10:30am Huang Basement + Zoom
General OH: Thursday	12:50pm to 2:20pm Huang Basement