Artificial Intelligence

BS (CS)\_SUMMER\_2024

Lab\_06 Manual

# Learning Objectives:

1. Informed Search Overview
2. Greedy Search
3. A\* Search
4. Coding Example for Greedy and A\*

# Informed Search

Informed search algorithms, also known as heuristic search algorithms, use problem-specific knowledge to find solutions more efficiently than uninformed search algorithms. These algorithms make use of a heuristic function to estimate the cost of reaching the goal from a given state.

**Heuristics function:** Heuristic is a function which is used in Informed Search, and it finds the most promising path. It provides an **estimate** of the cost from a given node to the goal node. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by **h(n)**, and it calculates the **cost of an optimal path between the pair of states**. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

**h(n) <= h\*(n)**

**Here h(n) is heuristic cost, and h\*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.**

Two popular informed search algorithms are Greedy Search and A\* (A-star).

## Greedy Search:

Greedy Search is an informed search algorithm that always chooses the path that appears to be the best at the current moment. It evaluates each state based solely on the heuristic function, without considering the cost of reaching that state. While Greedy Search can find solutions quickly, it may not always find the optimal solution.

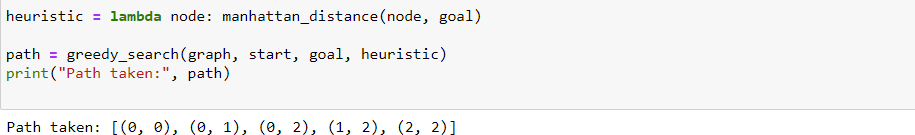
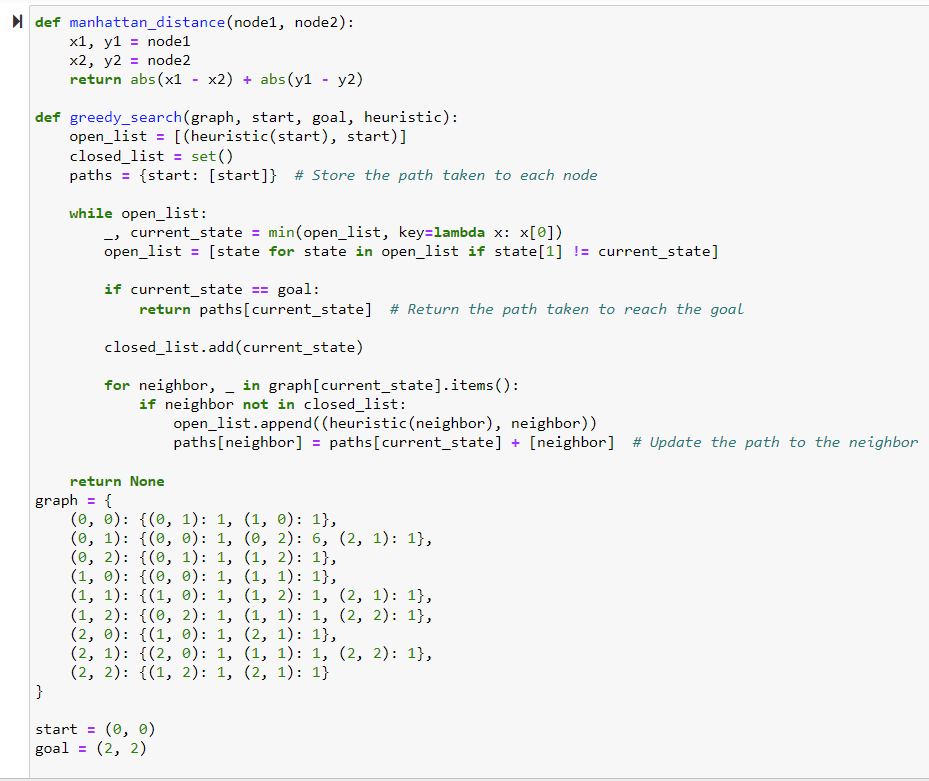
### STEP BY STEP GUIDE:

1. Initialize the *open* list with the *initial* state and its *heuristic* value.
2. Initialize the closed list as empty.
3. While the *open* list is *not empty*:
   * Remove the state with the lowest heuristic value from the *open* list.
   * If the *state* is the *goal* state, *return* the path to the *goal*.
   * Otherwise, *expand* the *state* and add its neighbors to the *open* list with their *heuristic*

values.

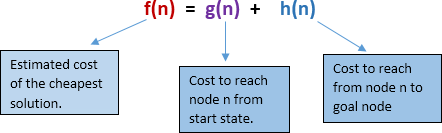
* + Add the current *state* to the *closed* list.

1. If the *open* list is *empty* and the goal state has not been found, *return failure*.



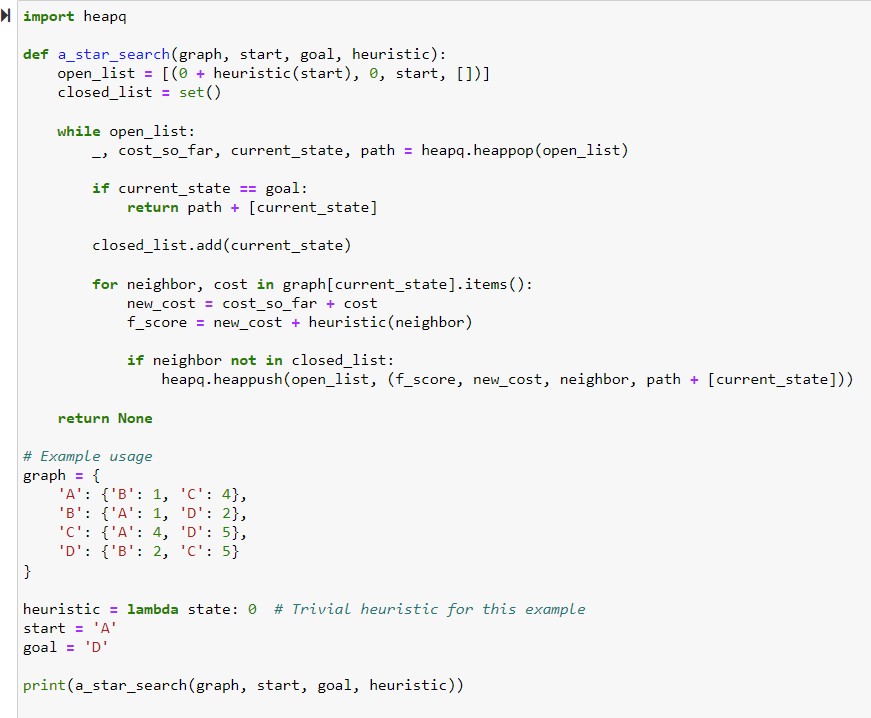
## A\* Search:

A\* is a widely used informed search algorithm that efficiently finds the lowest cost path between two nodes in a graph, but it incorporates heuristic information for the guided search. A\* combines the cost to reach a node from the start node (g(n)/g\_score) and a heuristic estimate of the cost to reach the goal from that node (h(n)/h\_score). It selects nodes to expand based on the sum of g\_score and h\_score (which is f(n)/ f\_score), choosing the node that minimizes this sum.

A\* uses a heuristic function to estimate the cost from each node to the goal. This heuristic guides the search towards the goal, making A\* more efficient than UCS in many cases.

### STEP BY STEP GUIDE:

1. Initialize the open list with the initial state and its f-score to be the heuristic estimate from the initial state to the goal.
2. Initialize the closed list as empty.
3. While the open list is not empty:
   * Remove the state with the lowest f-score from the open list.
   * If the state is the goal state, return the path to the goal.
   * Otherwise, expand the state and add its neighbors to the open list with updated f scores.
   * Add the current state to the closed list. If the open list is empty and the goal state has not been found, return failure.
4. If the open set is empty and the goal has not been reached, return failure.



## Coding Example for Greedy and A\* Searches:

### Finding Distance Between States

import math import heapq

# City map with distances between neighboring cities city\_map = {

'New York': {'Philadelphia': 80, 'Boston': 215}, 'Philadelphia': {'New York': 80, 'Washington D.C.': 135}, 'Boston': {'New York': 215, 'Providence': 50},

'Washington D.C.': {'Philadelphia': 135, 'Richmond': 95},

'Providence': {'Boston': 50}

}

# Euclidean distance between cities (approximation) def euclidean\_distance(city1, city2):

city\_coords = {

'New York': (40.7128, -74.0060),

'Philadelphia': (39.9526, -75.1652),

'Boston': (42.3601, -71.0589),

'Washington D.C.': (38.9072, -77.0369),

'Providence': (41.824, -71.4128)

}

lat1, lon1 = city\_coords[city1] lat2, lon2 = city\_coords[city2]

return math.sqrt((lat2 - lat1)\*\*2 + (lon2 - lon1)\*\*2)

#Greedy Search

def greedy\_search(start\_city, goal\_city, heuristic):

open\_list = [(heuristic(start\_city, goal\_city), start\_city)] closed\_list = set()

path = {start\_city: [start\_city]} # Store the path taken to each city

while open\_list:

\_, current\_city = min(open\_list, key=lambda x: x[0])

open\_list = [state for state in open\_list if state[1] != current\_city]

if current\_city == goal\_city:

return path[current\_city] # Return the path taken to reach the goal closed\_list.add(current\_city)

for neighbor, \_ in city\_map[current\_city].items():

if neighbor not in closed\_list: open\_list.append((heuristic(neighbor, goal\_city), neighbor))

path[neighbor] = path[current\_city] + [neighbor] # Update the path to the neighbor return None

#A\* Search

def a\_star\_search(start\_city, goal\_city, heuristic):

open\_list = [(0, start\_city)] closed\_list = set() cost\_so\_far = {start\_city: 0}

path = {start\_city: [start\_city]} # Store the path taken to each city

while open\_list:

\_, current\_city = heapq.heappop(open\_list)

if current\_city == goal\_city:

return path[current\_city] # Return the path taken to reach the goal closed\_list.add(current\_city)

for neighbor, distance in city\_map[current\_city].items():

new\_cost = cost\_so\_far[current\_city] + distance

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]: cost\_so\_far[neighbor] = new\_cost

f\_score = new\_cost + heuristic(neighbor, goal\_city) heapq.heappush(open\_list, (f\_score, neighbor))

path[neighbor] = path[current\_city] + [neighbor] # Update the path to the neighbor return None

#Performing Searches start\_city = 'New York' goal\_city = 'Washington D.C.' heuristic = euclidean\_distance

greedy\_path = greedy\_search(start\_city, goal\_city, heuristic) a\_star\_path = a\_star\_search(start\_city, goal\_city, heuristic)

print("Greedy Search Path:", greedy\_path) print("A\* Search Path:", a\_star\_path)