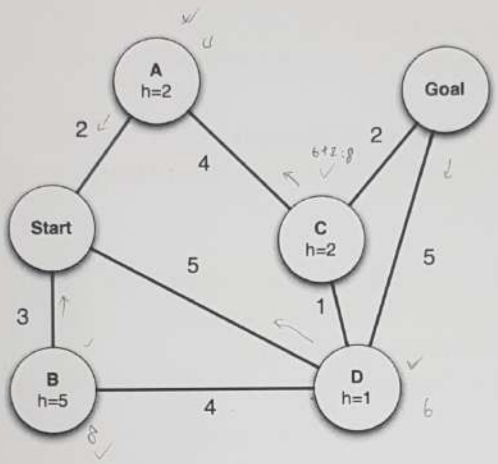
B09902053
何碩宸

Due: 03/31/2022 23:59

Foundations of Artificial Intelligence: Homework 1

Instructor: Shang-Tse Chen & Hsuan-Tien Lin

Problem 1 (10 points)



Write down the order of state expansion and the final path returned by each of the graph search (as oppose to tree search) algorithms below. You can assume ties are resolved alphabetically.

a) Depth-first search.
expansion: Start → A → C → D → B (back to D) → Goal
final path: Start → A → C → D → Goal

b) Breadth-first search.
expansion: Expand Start (discover A.B.D) → Expand A (discover C) → Expand B → Expand D (discover Goal)
final path: Start → D → Goal

c) Uniform cost search.
expansion: Expand Start (discover A.B.D) → Expand A (discover C) → Expand B → Expand D (discover Goal) → Expand C (update Goal) → Goal
final path: Start → A → C → Goal

d) Greedy search with the heuristic h shown on the graph.
expansion: Start → D → Goal
final path: Start → D → Goal

e) A^* search with the same heuristic.
expansion: Expand Start (discover A.B.D) → Expand A (discover C) → Expand D (discover Goal) → Expand B → Expand C (update Goal) → Goal
final path: Start → A → C → Goal

Problem 2 (10 points)

Suppose that the heuristic overestimates the shortest path from any state to the goal by a factor of at most ϵ, where $\epsilon > 1$. Prove that the cost of the path found by A* tree search is at most ϵ times the cost of the optimal path.

Suppose T_{true} is the cost annouced by A^*, S_{true} is the optimal cost, T_{eval} and S_{eval} are their evaluation costs.

We have
$$\begin{cases} T_{eval} = T_{true} & \text{(termination step, the heuristic} = 0) \\ T_{eval} \le S_{eval} & \text{(T terminates first)} \\ S_{eval} \le \varepsilon S_{true} & \text{(heuristic is } \varepsilon\text{-bounded)} \end{cases} \Rightarrow T_{true} = T_{eval} \le S_{eval} \le \varepsilon S_{true}$$

Hence, the cost found by A^* is ε-bounded.

Problem 3

(10 points)

```
function A* GRAPH SEARCH(problem)
    fringe ← an empty priority queue
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    closed ← an empty set
    ADD INITIAL-STATE[problem] to closed
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then
            return node
        end if
        for successor in GETSUCCESSORS(problem, STATE[node]) do
            if successor not in closed then
                ADD successor to closed
                fringe ← INSERT(MAKE-SUCCESSOR-NODE(successor, node), fringe)
            end if
        end for
    end loop
end function
```

Chances are a successor could have a better solution with a later expanded predecessor. But the implementation above simply ignore the successor if it is closed, which may lead to non-optimal solution.

The implementation of the A^* graph search algorithm above is incorrect. Briefly explain the bug in this implementation and justify your answer.

Take Problem 1 as an example, the above implementation will return Start → D → Goal as the answer, which is not an optimal solution.

Problem 4

(10 points)

You are scheduling for 6 classes taught by 3 instructors. Off course, each instructor can only teach one class at a time.

The classes are:

- Class 1 - Intro to Programming: 8:00-9:00am
- Class 2 - Intro to Artificial Intelligence: 8:30-9:30am
- Class 3 - Natural Language Processing: 9:00-10:00am
- Class 4 - Computer Vision: 9:00-10:00am
- Class 5 - Machine Learning: 10:30-11:30am

The instructors are:

- Instructor A - Can teach Classes 1, 2, and 5.
- Instructor B - Can teach Classes 3, 4, and 5.
- Instructor C - Can teach Classes 1, 3, and 4.

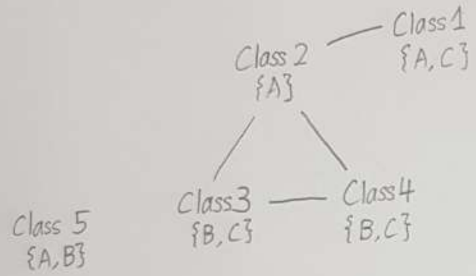
(1) Formulate this problem as a CSP. Describe the variables, domains and constraints.

Variables: Class 1 ~ Class 5

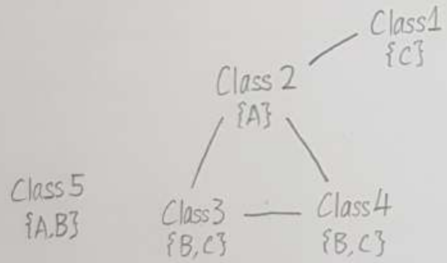
Domains: Class 1: { Instructor A, Instructor C }
Class 2: { Instructor A }
Class 3: { Instructor B, Instructor C }
Class 4: { Instructor B, Instructor C }
Class 5: { Instructor A, Instructor B }

Constraints: Class 1 ≠ Class 2
Class 2 ≠ Class 3
Class 2 ≠ Class 4
Class 3 ≠ Class 4

(2) Draw the constraint graph associated with your CSP.



(3) Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints).



(4) Give one solution to this CSP.

Class 1: Instructor C
Class 2: Instructor A
Class 3: Instructor B
Class 4: Instructor C
Class 5: Instructor A

(5) Your CSP should look nearly tree-structured. Briefly explain (one sentence or less) why we might prefer to solve tree-structures CSPs.

For normal CSP, we may solve it in $O(d^n)$ time
For tree-structured CSP, we may solve it in $O(nd^2)$ time