NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM EXAMINATION FOR
Semester 2 AY2022/2023

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

February 27, 2023 Time Allowed: 60 Minutes

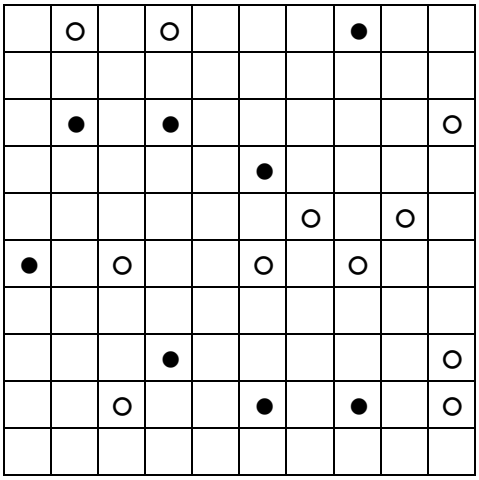
INSTRUCTIONS TO CANDIDATES

1. This assessment contains THREE (3) questions. All the questions are worth a total of 30 MARKS. It is set for a total duration of 60 MINUTES. You are to complete all 3 questions.
2. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
3. You are allowed to use NUS APPROVED CALCULATORS.
4. If something is unclear, solve the question under a reasonable assumption. State your assumption clearly in the answer. If you must seek a clarification, the invigilators will only answer questions with Yes/No/No Comment answers.
5. You may not communicate with anyone other than the invigilators in any way.

STUDENT NUMBER: _____

EXAMINER'S USE ONLY		
Question	Mark	Score
1	10	
2	10	
3	10	
TOTAL	30	

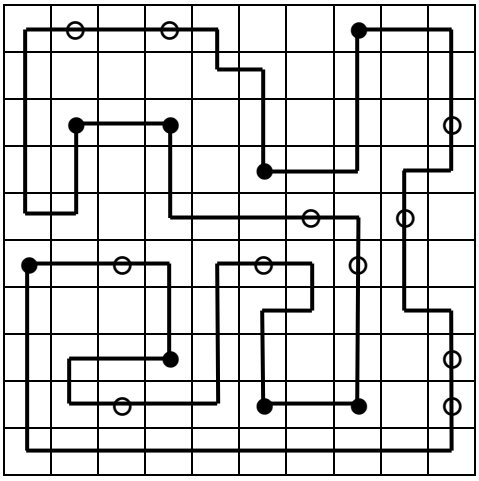
1. A Masyu puzzle is defined on a rectangular *r* by *c* grid (i.e., a grid with *r* rows and *c* columns). The grid is populated with *n* black circles (i.e., ●) and *m* white circles (i.e., ○). An example of one such Masyu puzzle is shown below.



The objective of the puzzle is to form a single looped path that links all the circles, such that this path follows certain rules. These rules are as follows.

- The path must pass through a subset of the cells in the grid such that it passes through the centre of those cells. This may be done as a straight line or as a 90° turn.
- The path may not cross itself, may not branch into two or more paths, and may not pass through the same cell twice.
- The path must pass through all cells with either a black circle or a white circle.
- When the path passes through a white circle, it must pass through it as a straight line. However, it must extend into a 90° turn in at least one of the cells adjacent to the white circle that it has extended into.
- When the path passes through a black circle, it must pass through it as a 90° turn. However, it must extend into straight lines in both cells adjacent to the black circle that it has extended into.
- The path need not pass though all cells.
- There is always exactly one solution for any given puzzle.

Based on the above rules, the solution to the Masyu puzzle given above is shown below.



(i) [6 marks] Define a search formulation for this path search problem. Your formulation should be efficient.

Solution: (Note that the solution below is purposely verbose for clarity. A summarised version that abstracted certain details would also be acceptable.)

State representation

- A list of 3-tuples, *path* = [(x_1, y_1, c_1), (x_2, y_2, c_2), …, (x_k, y_k, c_k)], where
  - each (x_i, y_i) corresponds to a position in the 2-dimensional grid, and
  - $c_i \in$ {North, South, East, West}, which indicates an exit direction for the path.
    - Note that the last 3-tuple added corresponds to the current end of the path.
    - Note that each parent state tells us how the path entered the current state
- A counter, *count*, which stores the number of B/W cells currently included in *path*.
  - Note that we denote a B/W cell as a cell containing either a white or black circle.

Initial state

- This state is a sentinel state; it is slightly different from a regular state in the ***Actions*** and ***Transition*** (i.e., transition model) functions will act differently for this state as compared to all other states that are not the initial state.
- Randomly select one B/W cell, (x*, y*).
  - Note that we pick a B/W cell as the initial point of the path since we know that the path ***must*** pass through each B/W cell, but may not pass through other cells.
- Instead of storing *path* and *counter*, it instead stores a list of 2-tuples representing the exits for the path as it passes through this B/W cell, *starts* = [(*enter_dir*, *exit_dir*), …].
  - If the initial state corresponds to a cell containing a white circle, we have:
    - (North, South) and (East, West).
  - If the initial state corresponds to a cell containing a black circle, we have:
    - (West, North), (North, East), (East, South) or (South, West).
- The position of the B/W cell at the initial state, *start_pos* = (x*, y*), is also stored.

Actions

- If the given state, ***s***, is the initial state, the set of actions corresponds to all the possible *exit_dir* values in ***s***.*starts* that are legal.
- Else, if ***s*** is not the initial state, then the set of actions is given by the 3 possible directions that can be taken when we move to the cell that lies in direction ***s***.*path*[-1][2] from the cell (***s***.*path*[-1][0], ***s***.*path*[-1][1]) that are legal.
- Note that there are only 3 possible directions to take when we enter a cell because we do not wish to backtrack and return to the parent cell.
- Note that possible directions are constrained by grid boundaries, and by the rules governing the path flow around B/W cells as defined in the rules – i.e., we define an invariant over the search tree that ensures all states are consistent with the rules.

Transition model

- The intermediate state, ***s***' is defined as follows.
  - Let (x', y') denote the coordinates of the new cell traversed to from ***s***, which is defined by the cell coordinates of ***s***, (***s***.*path*[-1][0], ***s***. *path*[-1][1]), and the direction the path took from ***s***, ***s***.*path*[-1][2].
    - Note that in the case of the initial state, you must instead use *start_pos*.
  - Append (x', y', ***a***) to ***s***'.*path*, where ***a*** is the action that was input with ***s***.
  - If (x', y') is a B/W cell, increment ***s***'.*count* by 1.

Goal test

- If the current state shares the location as the initial state, and direction taken matches ***s***.*path*[0][2], then it is a goal state.

Action costs

- Constant (e.g., 1)

(ii) [2 marks] Given the search tree specified, and assuming that r, c, n, and m are all positive integers, list the possible values for the branching factor and define the branching factor, b, as the maximum among these values. Similarly, specify the list of possible values for the depth and define the depth, d, as the maximum among these values.

Solution:

The path may have to traverse every cell of the grid, and at least $n+m$ cells, since it must pass through all the cells with black and white circles. Therefore, we have:

$$d \in [n+m, rc]; \text{ and } \boldsymbol{max}(d) = rc$$

The branching factor is governed by the number of possible actions at each node. The initial node either has 2 or 4 possible actions, given the possible paths in cells with white and black cells respectively. However, after the initial node, we have the following possible actions.

- Cells with a white circle: 1 possible action (since the path into the cell is already fixed, and we must either continue the path horizontally or vertically based on that input)
- Cells with a black circle: 2 possible actions (since we can only turn in 2 directions given the input)
- Cells without a circle: 3 possible actions (based on the 3 directions excluding the input)

Consequently, we have:

$$b \in [1, 3]; \text{ and } \boldsymbol{max}(b) = 3$$

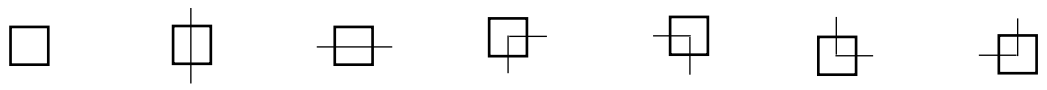
Note that the above ignores terminal nodes.

(iii) [2 marks] Write an expression in terms of r, c, n, and m for the size of this search tree and explain how your search formulation is efficient.

Solution:

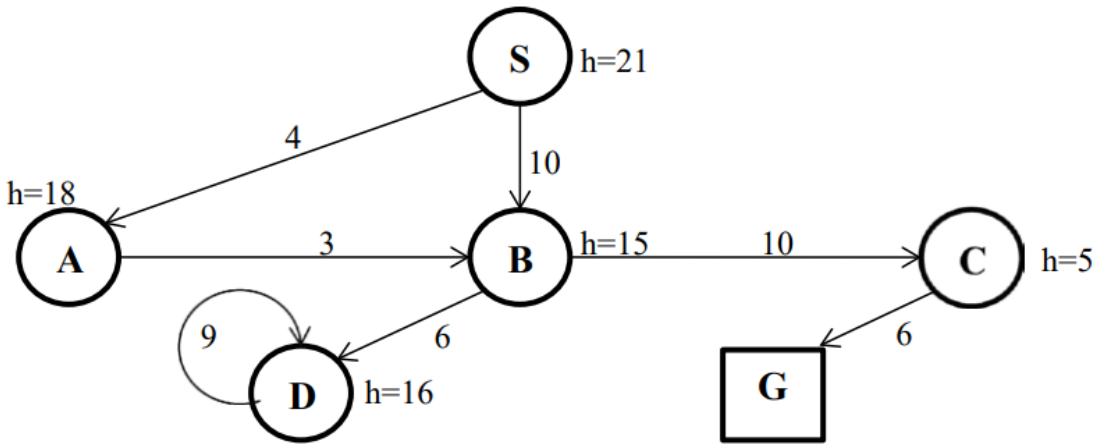
The size of the tree is thus bounded by $O(3^{rc})$.

This is efficient as compared to the brute force search version that is bounded by $O(7^{rc})$. Note that the base 7 is determine by the possible paths that can exist in a cell.



Further, the goal test for this version requires a traversal of the path, ensuring that all rules of the game are followed and that the path has no branching and is a cycle. This requires a minimal $O(rc)$ additional cost. However, in the version specified, the goal test is an $O(1)$ operation.

2a. Consider the directed graph depicted in the Figure below. Let **S** be the initial state and **G** the goal state. The cost of each action is as indicated in the figure. Further, assume that you are given a heuristic, h, whose values for each state are listed next to each state.



(i) [1 mark] Determine if h is admissible. Justify your answer.

Solution:

n	Shortest Path from n to G	h*(n)	h(n)
S	S-A-B-C-G	23	21
A	A-B-C-G	19	18
B	B-C-G	16	15
C	C-G	6	5
D	Not Applicable	∞	16
G	G	0	0

As $\forall n, h(n) < h^*(n)$, h **is admissible**.

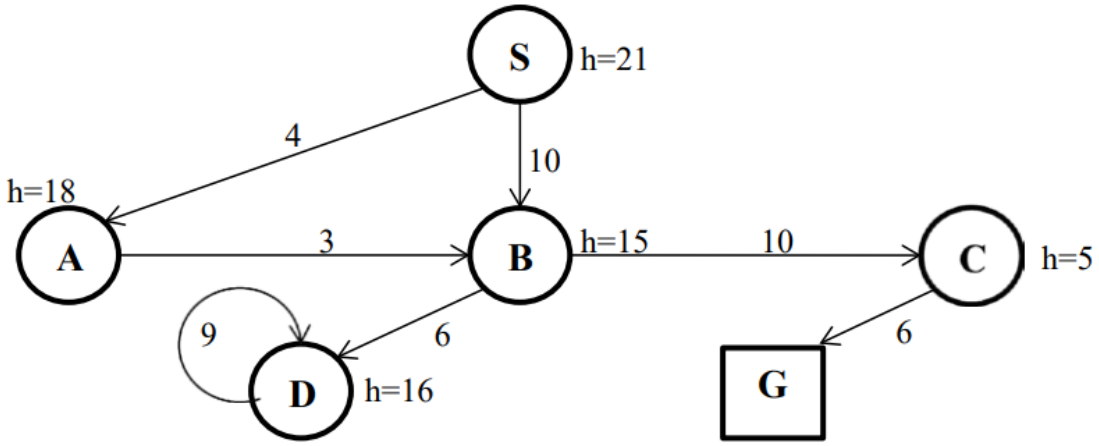
(ii) [1 mark] Determine if h is consistent. Justify your answer.

Solution:

n	n'	h(n') + d(n, n')	h(n)
S	**A**	18 + 4 = 22	21
S	**B**	15 + 10 = 25	21
A	**B**	15 + 3 = 18	18
B	**C**	5 + 10 = 15	15
B	**D**	16 + 6 = 22	15
C	**G**	0 + 6 = 6	5
D	**D**	16 + 9 = 25	16

As $\forall n, n'$ (where n' is a successor to n), $h(n) \leq h(n') + d(n, n')$, h **is consistent**.

2b. With reference to the directed search graph depicted in **(2a)**, which is repeated below for your convenience, assume that ties when pushing to the frontier or resolving priority are broken based on **alphabetical order** (e.g., **A** pushed/prioritised before **B**). If both are tied, resolve based on order of occurrence.



For each of the parts below, determine the **sequence of nodes popped from the frontier** based on the specified search algorithm. In cases where a heuristic is necessary, utilise h as shown in the figure.

Note that each node should be specified by a path (e.g., S-A-B). For example, given a search tree with only one path, i.e., A to B to C, we push A (i.e., node A), then pop A, then push B (via A – i.e., push node A-B), then pop B, then push C (via A-B – i.e., push node A-B-C). ***Do NOT specify a trace table. There are no marks awarded due to Error Carried Forward.***

(i) [1 mark] Depth-First Search (DFS) implemented using a tree search.

Solution:

S, **S-B**, **S-B-D**, **S-B-D-D**, … // **D** repeats

// Note all nodes are sub-paths of the same path
// Non-node answer: **S-B-D-D-**…

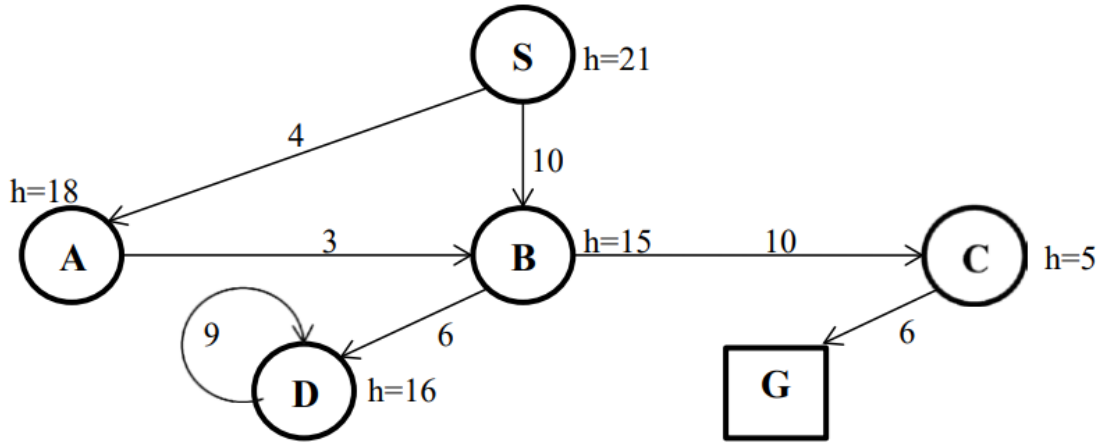
(ii) [2 marks] Uniform-Cost Search (UCS) implemented using graph search (Version 1).

Solution:

S, **S-A**, **S-B**, **S-B-D**, **S-B-C**, **S-B-C-G**

// Note all nodes are sub-paths of the same path
// Non-node answer: **S-A-B-D-C-G**

The directed search graph depicted in **(2a)** is again repeated below for your convenience.



(iii) [1 mark] Greedy Best-First Search implemented using tree search.

Solution:

S, **S-B**, **S-B-C**, **S-B-C-G**

// Note all nodes are sub-paths of the same path
// Non-node answer: **S-B-C-G**

(iv) [2 marks] A* Search implemented using graph search (Version 3).

Solution:

S, **S-A**, **S-A-B**, **S-A-B-C**, **S-A-B-C-G**

// Note all nodes are sub-paths of the same path
// Non-node answer: **S-A-B-C-G**

2c. [2 Marks] Assume that Uniform-Cost Search (UCS) and Depth-First Search (DFS) are applied on a graph where all nodes at depth *d* are goal nodes. Explain the advantage of using either approach.

Solution:

When we know we will reach a goal at depth *d*, DFS is **efficient** since it only incurs O(*bd*) time and space complexity (and O(*d*) if the backtracking variant is used), where *b* is the branching factor.

If we require an **optimal path**, then UCS should instead be used.

3a. Assume that you are given a maze-navigation puzzle that is implemented on a 2-dimentional grid (similar to the grid in the puzzle from Project 1). The objective is to move the agent from a start position to a goal position. Here, the agent can move in 8 directions (***northwest***, ***west***, ***southwest***, ***south***, ***southeast***, ***east***, ***northeast***, and ***north*** – this is similar to the possible moves that the King piece in Chess can make, which is again similar to the agent in the puzzle from Project 1) unless obstructed by an obstacle or the edge of the grid. Each state is thus defined by the position of the agent within the 2-dimenstional grid.

You may also assume that there are no other pieces in the maze (i.e., the maze only contains the agent and obstacles). You may also assume (i) that both the start and goal positions are on the grid, (ii) that the goal position is always reachable from the start position, and (iii) that all states considered in this puzzle are reachable from the start position.

However, we will define new action costs for the current puzzle. The ***north***, ***south***, ***east***, and ***west*** moves each cost 1 unit, while the ***northwest***, ***northeast***, ***southwest*** and ***southeast*** moves each cost $\sqrt{2}$ units.

(i) [2 marks] Let the heuristic h_1 be defined by Manhattan distance. Prove/Disprove the admissibility of this heuristic.

Solution:

Manhattan distance is **inadmissible** as a heuristic.

Consider the **counterexample** below.

n	
	G

Given that the agent is at location **n**, and the goal is at **G**, $h^*(\mathbf{n}) = \sqrt{2}$.

However, $h(\mathbf{n}) = 2$ when h is based on Manhattan distance.

(ii) [4 marks] Let the heuristic h_2 be defined by Euclidean distance. Assume that h_2 is admissible. Design another admissible heuristic, h_3, which dominates h_2.

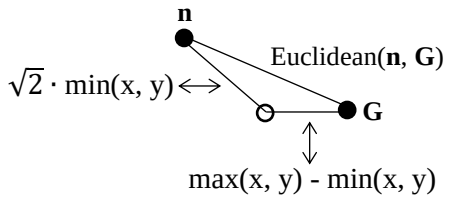
You must show that h_3 is admissible, and that it dominates h_2.

Solution:

Let $h_3(\mathbf{n}) = \sqrt{2} \cdot \min(x, y) + \max(x, y) - \min(x, y)$,
where $\mathbf{n}$ is located at $(\mathbf{n}_x, \mathbf{n}_y)$, and the nearest goal, $\mathbf{G}$, is located at $(\mathbf{G}_x, \mathbf{G}_y)$,
and where $x = |\mathbf{n}_x - \mathbf{G}_x|$, $y = |\mathbf{n}_y - \mathbf{G}_y|$.

// Note that this is based on weighted Chebyshev distance

h_3 **dominates** h_2 since they form a triangle inequality:



h_3 is **admissible**.

Since each diagonal move is more efficient than the 2 non-diagonal moves required to reach the same destination, the **shortest distance must include min(x, y) diagonal moves**. (This can again be shown using a triangle inequality similar to the above for any (x, y) tuple.)

Specifically, at least **min(x, y) diagonal moves** are required to move to the same row or the same column as **G**, and then at least **an additional max(x, y) - min(x, y) non-diagonal moves** are required to reach **G**.

As such if there are no obstacles, $h_3 = h^*$. Since this is a **relaxation of the problem constraints**, we know that the inclusion of obstacles will increase the value of h^*, therefore we would have $h_3 \leq h^*$ in cases involving obstacles.

Consequently, h_3 satisfies the admissibility constraint: $\forall \mathbf{n}$, $h_3(\mathbf{n}) \leq h^*(\mathbf{n})$.

3b. [4 marks] Suppose that Graph-Based Best-First Search is utilised with f(n) = 1/(g(n) + h(n)) if g(n) + h(n) > 0, and otherwise, f(n) = 0. Further, assume that the given search space is finite, and that a goal exists.

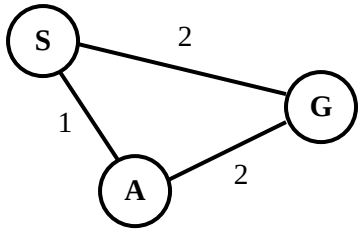
If h is an arbitrary heuristic that is consistent, prove that the search will give the longest path to the goal, or else, provide a counterexample. (Recall that g(n) refers to the actual path cost from the initial state to the ***node*** n.)

Solution:

False.

Consider the following **counterexample**.

Assume that ∀**n**, h(**n**) = 0, and that we have the following search tree with initial state **S**, and goal state **G**.



In this case, we have the following trace.

Iteration 1 Frontier: [(**S**, f(**S**) = 0 since g(**S**) + h(**S**) = 0)]
Iteration 2 Fronter: [(**S**-**G**, f(**S**-**G**) = 1 / (2+0) = 0.5), (**S**-**A**, f(**S**-**A**) = 1 / (1+0) = 1)]

Returns **S**-**G**, which is not the longest path.

END OF PAPER

BLANK PAGE

BLANK PAGE