NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM EXAMINATION FOR
Special Term (Part 1) AY2022/2023

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

May 30, 2023 Time Allowed: 60 Minutes

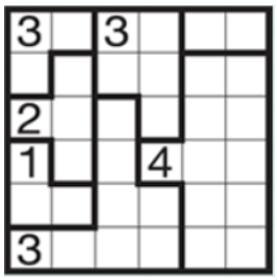
INSTRUCTIONS TO CANDIDATES

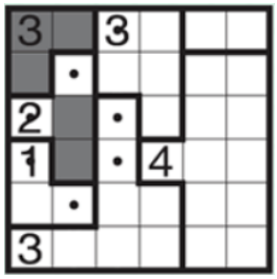
1. This assessment contains THREE (3) questions. All the questions are worth a total of 30 MARKS. It is set for a total duration of 60 MINUTES. You are to complete all 3 questions.
2. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
3. You are allowed to use NUS APPROVED CALCULATORS.
4. If something is unclear, solve the question under reasonable assumptions. State your assumptions clearly in the answer. If you must seek 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	

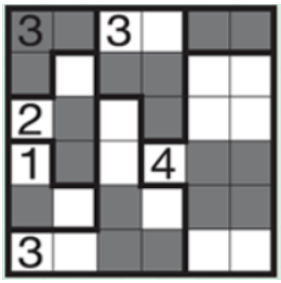
1. A certain puzzle game is played on a square n by n grid (i.e., a grid with n rows and n columns). The grid is separated into k contiguous regions. Each region, $\boldsymbol{R}_i$ (where $i \in [1, k]$), has an associated positive integer value, v_i. An example of this puzzle where $n = 6$ and $k = 7$ is depicted on the right. Notice that each of the k regions is bounded by a thick line, with the value for the region stated in the region. However, notice that some regions (e.g., the region in the top-right of the grid) have no associated value – such regions have variable values ranging from 0 to the size of the region (i.e., the number of cells in the region).



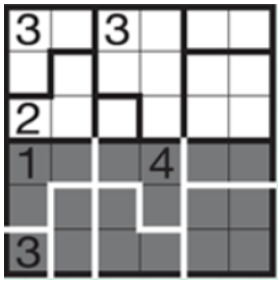


To solve this puzzle, one must fill contiguous cells in the region such that for region $\boldsymbol{R}_i$, the number of contiguous cells filled is v_i. When filling cells in this manner, there is a further constraint: filled cells cannot be contiguous across the regions. Continuing from the example puzzle above, suppose that we fill the first two regions in the top-left of the grid. The figure on the left shows a possible partial solution. The shaded cells denote filled cells. Notice that these filled cells only exist within the specified regions and are contiguous. The cells containing dots denote the cells that cannot be filled, since filling them would cause the solution to become inconsistent with the constraint that requires filled cells to not be contiguous across different regions.

To solve the puzzle, we must first fill all the regions (though you may choose to leave regions with no value empty). The puzzle is solved if the following constraint is satisfied. When all filled cells "fall" to the bottom of the grid, retaining their shape and not coexisting in cells with other filled cells, the bottom half of the grid must be completely filled. A solution to the example puzzle shown above is thus as follows.



← After all regions are filled but before the filled cells have fallen



← After cells have fallen, notice that the fallen cells are perfectly tessellated

Assume that the puzzle input is a list of 2-tuples, $\boldsymbol{P} = \{(\boldsymbol{R}_1, v_1), (\boldsymbol{R}_2, v_2), \ldots, (\boldsymbol{R}_k, v_k)\}$, where each $\boldsymbol{R}_i = \{(r_{i,1}, c_{i,1}), (r_{i,2}, c_{i,2}), \ldots\}$, and where each $(r_{i,j}, c_{i,j})$ refers to the row and column coordinates respectively for the j-th cell in the i-th region $\boldsymbol{R}_i$. Note that the sequence of row-column coordinates in each $\boldsymbol{R}_i$ is sorted primarily in terms of descending order of row coordinate value, and secondarily in terms of ascending order of column coordinate value – i.e., cells on the top row will be ordered before cells on lower rows, and within each row, cells are ordered from left to right; however, note that there is no specific order over the regions. Finally, note the following properties for $\boldsymbol{P}$.

- For the bottom row, the leftmost cell is (1, 1), while the rightmost is (1, n).
- For the top row, the leftmost cell is (n, 1), while the rightmost is (n, n).
- Each $v_i \leq |\boldsymbol{R}_i|$, $\sum_{i=1}^{k} v_i, = n^2/2$, and for cases where $\boldsymbol{R}_i$ has no value (i.e., is variable from 0 to $|\boldsymbol{R}_i|$), then $v_i = \textit{null}$.

Thus, for the example puzzle above, we have $\boldsymbol{P}$ = {($\boldsymbol{R}_1$, 3), ($\boldsymbol{R}_2$, 3), ($\boldsymbol{R}_3$, null), ($\boldsymbol{R}_4$, 2), ($\boldsymbol{R}_5$, 4), ($\boldsymbol{R}_6$, 3), ($\boldsymbol{R}_7$, 1)}, where: $\boldsymbol{R}_1$ = {(6, 1), (6, 2), (5, 1)}; $\boldsymbol{R}_2$ = {(6, 3), (6, 4), (5, 3), (5, 4), (4, 4)}; $\boldsymbol{R}_3$ = {(6, 5), (6, 6)}; $\boldsymbol{R}_4$ = {(5, 2), (4, 1), (4, 2), (3, 2)}; $\boldsymbol{R}_5$ = {(5, 5), (5, 6), (4, 5), (4, 6), (3, 4), (3, 5), (3, 6), (2, 5), (2, 6), (1, 5), (1, 6)}; $\boldsymbol{R}_6$ = {(4, 3), (3, 3), (2, 3), (2, 4), (1, 1), (1, 2), (1, 3), (1, 4)}; and $\boldsymbol{R}_7$ = {(3, 1), (2, 1), (2, 2)}.

Your task is to model the puzzle as a search problem so that we may apply uninformed, informed, or local search algorithms to solve the problem. In doing this, you are to assume the following partial search problem formulation.

- **State**: each state corresponds to partially assigned solutions, $\boldsymbol{S}$ = {$\boldsymbol{F}_1$, $\boldsymbol{F}_2$, …, $\boldsymbol{F}_k$}, where each $\boldsymbol{F}_i \subseteq \boldsymbol{R}_i$, $|\boldsymbol{F}_i| \leq v_i$, and such that all coordinates in $\boldsymbol{F}_i$ are contiguous.
- **Initial State**: with the initial state, each $\boldsymbol{F}_i \in \boldsymbol{S}$ is empty, i.e., $\forall \boldsymbol{F}_i \in \boldsymbol{S}$, $|\boldsymbol{F}_i| = \emptyset$.
- **Action Costs**: all action costs correspond to a positive constant – e.g., 1.
- **Goal Check**: this function, *isGoal*($\boldsymbol{S}$), returns *True* if $\forall \boldsymbol{F}_i \in \boldsymbol{S}$, $|\boldsymbol{F}_i| = v_i$, and after all the filled cells have fallen, the bottom half of the grid is completely filled. Note that this function automatically returns *False* if $\exists \boldsymbol{F}_i \in \boldsymbol{S}$ where $|\boldsymbol{F}_i| \neq v_i$.

(i) [7 marks] Complete the search problem formulation by defining **Actions** and **Transition Model**. Your formulation must be efficient (i.e., significantly better than brute-force search).

Solution:

Actions:

- This function, *actions*($\boldsymbol{S}$), must determine the specific v_i coordinates to add to each $\boldsymbol{F}_i$.
- This is a ***goal search*** problem, so a path is not important. Consequently, the size of the search tree may be reduced by <u>not</u> permuting over assignment order (similar to CSPs). Let the ordering be $\boldsymbol{F}_1$ on level 1, $\boldsymbol{F}_2$ on level 2, and so on. This also means that *actions*($\boldsymbol{S}$) must be able to access the current depth.
- For each $\boldsymbol{F}_i$, there are two general cases:
  - Case 1: $v_i \neq null$
    find all possible $\boldsymbol{F}_i \subseteq \boldsymbol{R}_i$ where $|\boldsymbol{F}_i| = v_i$ and the coordinates are contiguous.
  - Case 2: $v_i = null$
    find all possible $\boldsymbol{F}_i \subseteq \boldsymbol{R}_i$ where $|\boldsymbol{F}_i| \leq |\boldsymbol{R}_i|$ and the coordinates are contiguous.
- Decompose *actions*($\boldsymbol{S}$) such that it will utilise a *candidates*($\boldsymbol{R}_i$, v_i) function that finds all legal combinations for $\boldsymbol{F}_i$ under Case 1 above. Consequently, when Case 2 is observed, *candidates*($\boldsymbol{R}_i$, v_i) must be called iteratively, replacing v_i with the range [0, $|\boldsymbol{R}_i|$].
- The function *candidates*($\boldsymbol{R}_i$, v_i) can be implemented as a nested search.
  - First, determine the adjacency matrix over the coordinates in $\boldsymbol{R}_i$. This can be done by linking any coordinate with another if the row difference or the column difference, exclusively, is exactly 1. This requires $|\boldsymbol{R}_i|(|\boldsymbol{R}_i|+1)/2$ comparisons.
  - Next, perform a Depth-limited Search (DLS) with each coordinate in $\boldsymbol{R}_i$ as the root (i.e., $|\boldsymbol{R}_i|$ DLS executions). The depth limit is v_i. The paths at all leaf nodes at depth $v_i - 1$ (since the root is at depth 0) correspond to candidates for $\boldsymbol{F}_i$. Note that each such execution of DLS must return all paths at depth $v_i - 1$.
  - Note that this function is only called exactly k times (as preprocessing, since we can simply reference all possible legal permutations of $\boldsymbol{F}_i$ after generation).

Transition Model:

- Simply update the current $\boldsymbol{F}_i$ (where $\boldsymbol{F}_i = \emptyset$; recall the CSP-like ordering) with a valid set of coordinates (i.e., one of the sets output by *candidates*($\boldsymbol{R}_i$, v_i)).

(ii) [3 marks] Define the runtime complexity of the ***actions*** function you specified in Part **(i)** and define the size of the resultant search tree.

Solution:

Let N be the size of the grid (i.e., $N = n^2$).

The runtime complexity of the actions function requires the following.

- Generation of the adjacency matrix: $|\boldsymbol{R}_i|(|\boldsymbol{R}_i|+1)/2$ comparisons or $O(N^2)$.
- Generation of all candidates for $\boldsymbol{F}_i$ via DLS:
  - Let M be the size of the region in question (i.e., $M = v_i$).
  - Each $\boldsymbol{F}_i$ requires M DLS executions.
  - Each DLS execution has complexity $O(4^{M-1})$ since the depth limit is $M - 1$.
  - Runtime complexity is thus: $O(M \cdot 4^{M-1})$ or $O(N \cdot 4^{N-1})$.

Thus, the ***actions*** function has a runtime complexity of $O(N^2) + O(N \cdot 4^{N-1})$, or just $O(N \cdot 4^{N-1})$.

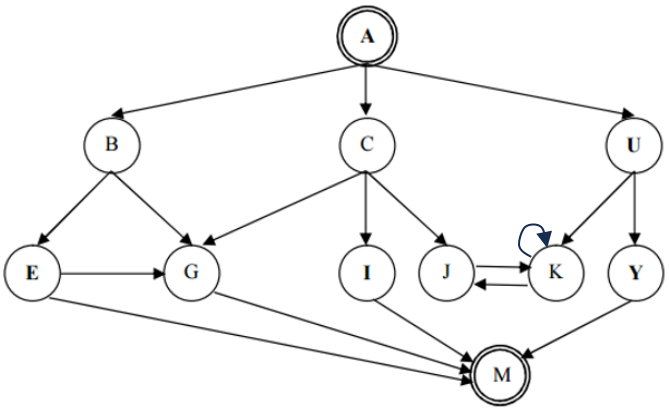
Note that this has lower complexity than brute-force search, which has complexity $O(k \cdot N!)$. This is because you must find the $|\boldsymbol{R}_i|$ choose v_i subsets.

The size of the search tree is bounded by $O((N \cdot 4^N)^k)$.

- At each level there are $O(N \cdot 4^N)$ possible options for $\boldsymbol{F}_i$; this is the branching factor.
- There are k levels in this tree.

Note that the bounds above can be defined more tightly by considering that $\sum|\boldsymbol{R}_i| = N$.

2a. Consider the directed graph depicted in the Figure below. Let **A** be the initial state and **M** the goal state.



h(A) = 7
h(B) = 5
h(C) = 3
h(U) = 4
h(E) = 2
h(G) = 3
h(I) = 6
h(J) = 2
h(K) = 1
h(Y) = 2
h(M)= 0

(i) [2 marks] Trace the exploration order for ***hill-climbing with sideways moves***. For the evaluation function, use f(n) = –h(n).

Solution:

A – C – J – K – K – … // infinite loop due to sideways move

(ii) [2 marks] Trace the exploration order using ***local beam search*** with k = 2. For the evaluation function, use f(n) = –h(n). Note the following additional information.

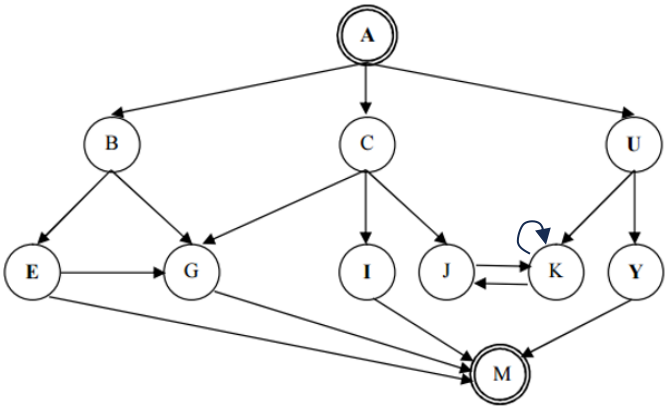
- No sideways moves.
- Candidates for the next beam include only states whose f-values are larger than the maximum f-value over the states in the previous iteration's beam.
- For tie-breaking, use **alphabetical order** (e.g., **A** will appear before **B** on the beam).
- Terminate the search when the beam contains a goal state or is empty.
- Even though k = 2, the initial beam only contains A.

Solution:

(A) – (C, U) – (K, J)

// Pop A(-7); Successors: B(-5), C(-3), U(-4); all > A(-7); insert C(-3), U(-4)
// Pop C(-3), U(-4); Successors: G(-3), I(-6), J(-2), K(-1), Y(-2); J(-2), K(-1), Y(-2) > C(-3);
// insert K(-1), J(-2); J(-2) over Y(-2) due to tie-breaker
// Pop K(-1), J(-2); Successors: K(-1), J(-2); none > K(-1) so terminate search.

2b. With reference to the directed search graph depicted in **(2a)**, which is repeated below for your convenience, assume that all action costs are 1. Further, assume that ties when pushing to the frontier or resolving priority are broken based on **reverse alphabetical order** (e.g., **B** pushed/prioritised before **A**). If both are tied, resolve based on order of occurrence.



h(A) = 7
h(B) = 5
h(C) = 3
h(U) = 4
h(E) = 2
h(G) = 3
h(I) = 6
h(J) = 2
h(K) = 1
h(Y) = 2
h(M) = 0

(i) [1.5 marks] Determine if the given heuristic, h, is ***admissible***. Justify your answer.

Solution:

No. h*(A) = 3 but h(A) = 7.

// There are multiple paths from A to M with path cost 3 – e.g., A – B – E – M.
// Also, note that there are other examples that may be used as a counterexample:
// e.g., h(B) = 5 > h*(B) = 2; OR h(C) = 3 > h*(C) = 2, etc.

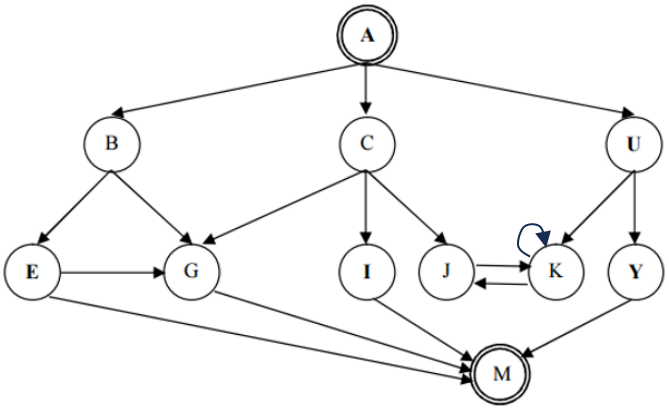
(ii) [1.5 marks] Determine if the given heuristic, h, is ***consistent***. Justify your answer.

Solution:

No. Since consistency would imply admissibility, but h is inadmissible.
// X (consistency) ⇒ Y (admissibility) must hold (i.e., has been proven);
// However, given ¬Y, then ¬X since X ⇒ Y must be true.

// OR: by counterexample: e.g., cost(A, B) = 1, h(B) = 5, but h(A) = 7 > h(B) + cost(A, B).

(iii) [3 marks] Trace the order of nodes popped from the frontier when ***A* Search*** implemented using ***graph search version 3*** is executed. Further, determine if the path found is optimal. For your convenience, the relevant graph is repeated below. Further, recall that all action costs are 1.



h(A) = 7
h(B) = 5
h(C) = 3
h(U) = 4
h(E) = 2
h(G) = 3
h(I) = 6
h(J) = 2
h(K) = 1
h(Y) = 2
h(M)= 0

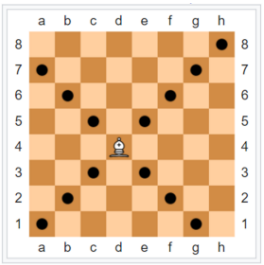
Solution:

Order of nodes popped: **A – C – J – K – U – Y – M**.

// [A(0,7,-)]; **pop A(0,7,-)**
// [C(1,3,A), U(1,4,A), B(1,5,A)]; **pop C(1,3,A)**
// [J(2,2,A-C), U(1,4,A), G(2,3,A-C), B(1,5,A), I(2,6,A-C)]; **pop J(2,2,A-C)**
// [K(3,1,A-C-J), U(1,4,A), G(2,3,A-C), B(1,5,A), I(2,6,A-C)]; **pop K(3,1,A-C-J)**
// [U(1,4,A), G(2,3,A-C), B(1,5,A), I(2,6,A-C)]; **pop U(1,4,A)**
    // J(4,2,A-C-J-K) not added above as J already on reached
// [Y(2,2,A-U), G(2,3,A-C), B(1,5,A), I(2,6,A-C)]; **pop Y(2,2,A-U)**
    // K(2,1,A-U) not added above as K already on reached
// [M(3,0,A-U-Y), G(2,3,A-C), B(1,5,A), I(2,6,A-C)]; **pop M(3,0,A-U-Y)**
    // **Goal Found** (return)

Path found: **A – U – Y – M**, which **is an optimal path** (path cost = 3).

3. Consider the Project 1 problem description – i.e., given a Chess board, some obstacles and opposing pieces – find a path from a starting position to a goal position. However, here, instead of the agent corresponding to a Knight piece, the agent will correspond to a Bishop piece. The movement range of the Bishop piece is depicted on the right. As with Project 1, assume that board corresponds to a rectangular grid. For this question, you may further assume that there is only one goal position.



Let a heuristic, h_1, be defined as follows for any given state n.

- $h_1(n) = 0$ if $n = goal$, where n and $goal$ are coordinates on the Chess board
- $h_1(n) = \infty$ if $(n \neq goal) \wedge (\text{cellColour}(n) \neq \text{cellColour}(goal))$
- $h_1(n) = 0$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 1)$
- $h_1(n) = 1$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0)$

(i) [5 marks] Prove that h_1 is ***admissible***.

Solution:

Going case by case, we have the following.

- If $n = goal$, $h_1(n) = 0 = h^*(n) = h^*(goal)$; i.e., $h_1(n) \leq h^*(n)$.
- If $(n \neq goal) \wedge (\text{cellColour}(n) \neq \text{cellColour}(goal))$, we know that the Bishop can never reach this cell since it can only reach cells of the same colour. Thus, we have: $h_1(n) = \infty = h^*(n)$; i.e., $h_1(n) \leq h^*(n)$.
- For the case where $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 1)$, use the following observations:
  - $(\text{cellColour}(n) = \text{cellColour}(goal)) \Rightarrow (\text{ManhattanDist}(n, goal) \% 2 = 0)$
  - $(\text{cellColour}(n) \neq \text{cellColour}(goal)) \Rightarrow (\text{ManhattanDist}(n, goal) \% 2 = 1)$
  - This may be inferred by observation and proven inductively.
  - Consequently, no possible (n, goal) pairing can satisfy this condition – i.e., we have $h_1(n) \leq h^*(n)$ since it is vacuously true.
- For the case where $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0)$, we know that at least one move is necessary to reach the goal since $n \neq goal$, and that any such goal position is reachable since it shares the same coloured cell. Thus, we have $h_1(n) = 1 \leq h^*(n)$; i.e., $h_1(n) \leq h^*(n)$.

Thus, we may conclude that h_1 **is admissible**.

(ii) [5 marks] Construct an ***admissible*** heuristic, h_2, that ***dominates*** h_1.

Note that your heuristic must not be: (A) h_1, (B) $\forall n$, $h(n) = 0$; (C) the (abstract) optimal heuristic, h*; or (D) a linear combination or simple function over (A), (B) and (C).

You must show that h_2 is admissible, and that it dominates h_1.

Solution:

Modify the following case under h_1.

- $h_1(n) = 1$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0)$

Replace it with the following.

- $h_2(n) = 1$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0) \wedge |\, n.\text{row} - goal.\text{row} \,| = |\, n.\text{column} - goal.\text{column} \,|$

  // Notice that for the above case, both n and the goal are on the same diagonal.

- $h_2(n) = 2$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0) \wedge |\, n.\text{row} - goal.\text{row} \,| \neq |\, n.\text{column} - goal.\text{column} \,|$

  // Consequently, these cases correspond to *n* and the *goal* being on different diagonals.

We may easily observe that when *n* and the *goal* are not on the same diagonal (i.e., we cannot link them via the same diagonal on the Chess board), then more than 1 move is necessary. As such, this version of h_2 is admissible.

Further, since the set of cases where h_2 returns a value of 2 is a subset of the cases where h_1 returns a value of 1, we can directly show that h_2 dominates h_1.

END OF PAPER

BLANK PAGE

BLANK PAGE