NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Semester 1 AY2023/2024

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

December 5, 2023 Time Allowed: 2 Hours

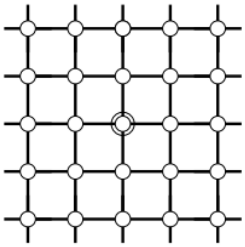
INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number only. Do not write your name.
2. This assessment contains FIVE (5) questions and comprises TWENTY-TWO (22) pages, including blank pages. All the questions are worth a total of 60 MARKS. It is set for a total duration of 120 MINUTES. You are to complete all 5 questions.
3. This is a CLOSED BOOK assessment. However, you may reference a SINGLE DOUBLE-SIDED A4 CHEAT SHEET.
4. You are allowed to use NUS APPROVED CALCULATORS.
5. 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.
6. You may not communicate with anyone other than the invigilators in any way.

STUDENT NUMBER:	**A**								

EXAMINER'S USE ONLY		
Question	Mark	Score
1	6	
2	12	
3	12	
4	14	
5	16	
TOTAL	60	

1a. Consider the unbounded regular 2-dimensional grid state space depicted on the right.



The initial state is the origin (which is marked in the diagram), and the goal state is at the coordinate (x, y). Assume that the branching factor for this search problem is 4, since at each state, a move can be made to the coordinate above, to the left, to the right, or below.

(i) [2 marks] How many distinct states will there be at depth k of the search tree (for $k > 0$)?

Option **A**: 4^k

Option **B**: $4k$

Option **C**: $4k^2$

Option **D**: None of the above.

Solution: B
/* The states at depth k form a square rotated at 45 degrees to the grid. Obviously, there are a linear number of states along the boundary of the square, so the answer is $4k$. */

(ii) [2 marks] How many nodes will a ***tree search*** implementation of ***Breadth-First Search*** (BFS) expand before terminating?

Option **A**: $((4^{x+y+1} - 1) / 3) - 1$

Option **B**: $4(x + y) - 1$

Option **C**: $2(x + y)(x + y + 1) - 1$

Option **D**: None of the above.

Solution: C
/* There are quadratically many states within the square for depth $x + y$, so the answer is $2(x + y)(x + y + 1) - 1$. */

1b. [2 marks] Let a ***randomised queue*** be defined as a queue whose pop operation removes and returns a ***random*** element from that queue. Suppose that we implement algorithm ***A***, which is ***Breadth-First Search*** (BFS) with ***late-goal testing*** and ***graph search version 2*** and implement algorithm ***B*** such that it is the same as algorithm ***A***, except that the ***frontier*** in algorithm ***B*** is a randomised queue. When this algorithm ***B*** is applied to search problems where algorithm ***A*** is both complete and optimal, which of the following would be true?

Option **A**: Algorithm ***B*** is complete.

Option **B**: Algorithm ***B*** is both complete and optimal.

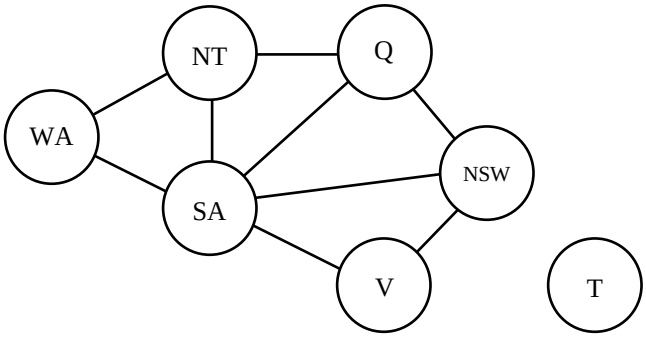
Option **C**: Algorithm ***B*** is neither complete nor optimal.

Option **D**: None of the above.

Solution: C
/* If a goal state exists, BFS is complete even with an infinite search space. However, the randomised version does not search systematically and may never find this goal state. */

2a. Recall the ***Australia Map Colouring*** problem mentioned during the lectures, where we must assign one of three possible colours (i.e., R: red, G: green, or B: blue) to each state in Australia (i.e., WA: *Western Australia*, NT: *Northern Territory*, SA: *Southern Australia*, Q: *Queensland*, NSW: *New South Wales*, V: *Victoria*, and T: *Tasmania*).

Further, recall that the associated constraint graph is defined as follows.



Suppose that the current state of a ***backtracking search*** that is applied to solve the above problem corresponds to the following domains.

WA	NT	Q	SA	NSW	V	T
R	G, B	R, G, B	G, B	R, G, B	R, G, B	R, G, B

Further, assume that WA has just been assigned R, and that constraint propagation has been completed.

(i) [1 mark] Determine the output of the ***Minimum-Remaining-Values*** (MRV) heuristic when it is applied to the state described above.

Solution: NT or SA

/* MRV over unassigned variables:
- **NT, SA: 2**
- Q, NSQ, V, T: 3

***/**

(ii) [1 mark] Determine the output of the ***Degree*** heuristic when it is applied to the state described above.

Solution: SA

/* Degree over unassigned variables:
- T: 0
- NT, V: 2
- Q, NSW: 3
- **SA: 4**

***/**

(iii) [1 mark] Determine the output of the ***Least-Constraining-Value*** (LCV) heuristic when it is applied to the state described above, assuming that we wish to assign a value to SA.

Solution: G or B

/* LCV over SA with domain (G, B) sharing constraints with (NT, Q, NSW, V):
LCV(SA = G) = | NT = (B); Q = (R, B); NSW(R, B); V = (R, B) | = 7
LCV(SA = B) = | NT = (G); Q = (R, G); NSW(R, G); V = (R, G) | = 7
*/

2b. Given the ***Australia Map Colouring*** problem described in Question **2a** above, suppose that we wish to attempt to solve the problem via local search. To do so, we will adopt a complete state formulation by initially colouring all states. An action in this case would be to change the colour assignment of just one state.

(i) [1 mark] What could be a good local search heuristic to apply to this problem? Note that your suggested heuristic may not be h(n) = 0 for all n, the (abstract) optimal heuristic, h*, or a linear combination/simple function thereof.

Solution:

One possible heuristic is the number of edges in the constraint graph that have variables with conflicting colours (i.e., at least one similar colour in their domains).

(ii) [1 mark] Assuming that we have the following state:

WA = NT = B; SA = Q = V = R; NSW = T = G

What is the value of the heuristic you defined in Question **2b(i)** for this state?

Solution:

Given the solution in **(i)** above, the given state has 3 conflicting pairs:

- WA and NT
- Q and SA
- SA and V

In this case, we have h = 3.

2c. [7 marks] In a particular single round-robin scheduling problem, there are n teams and $n - 1$ time slots in a tournament, where $n \geq 2$, and n is always even. The objective is to find a round-robin scheduling such that the following constraints are satisfied – i.e., by solving it as a ***constraint satisfaction problem*** (CSP).

- ***Constraint 1***: No team can play itself.
- ***Constraint 2***: For each time slot, each team must play exactly once.
- ***Constraint 3***: In the entire tournament, all teams must play against every other team exactly once.

For example, suppose that we have $n = 4$ teams. Denote these teams as X_0, X_1, X_2, and X_3. In this case, we will have 3 time slots, which we will denote as T_0, T_1, and T_2. A possible consistent schedule would then be as follows.

- T_0: { (X_0, X_1), (X_2, X_3) }
- T_1: { (X_0, X_2), (X_1, X_3) }
- T_2: { (X_0, X_3), (X_1, X_2) }

Given that each (X_i, X_j) above corresponds to a match between X_i and X_j, we observe the following.

- No team plays against itself (i.e., ***Constraint 1*** is satisfied).
- For each time slot, all teams play exactly once (i.e., ***Constraint 2*** is satisfied).
- All teams play against every other team exactly once (i.e., ***Constraint 3*** is satisfied).

To define the variables in this CSP, we first define a single matrix, $\boldsymbol{A}$, with n rows and $n - 1$ columns. The rows correspond to the teams, while the columns correspond to the time slots. The elements of this matrix thus correspond to the opponents.

For example, $\boldsymbol{A}[0, 2] = 3$ means that team X_0 will compete against team X_3 in time slot T_2.

More generally, notice that $\boldsymbol{A}[i, j] = k$ corresponds to the allocation of a match between X_i and X_k in time slot T_j.

The variables in this scheduling problem thus correspond to the elements of the matrix, while the domains correspond to positive integers from 0 to n (exclusive). The matrix on the right describes the schedule given in the example above.

1	2	3
0	3	2
3	0	1
2	1	0

Use the above variables and domains to define the constraints for this CSP.

You are allowed to use logical operators – i.e., the for-all operator, ∀; the there-exists operator, ∃, etc.

You are also allowed to use the `AllDiff` constraint, with the following syntax: `AllDiff`(***S***), where ***S*** is the set of variables whose values must be different. However, do note that you ***may not*** use the `ExactlyOnce` function.

For simplicity of definitions, you may use the following sets if necessary:

- $\boldsymbol{X}$ = { 0, 1, 2, …, $n - 1$ }
- $\boldsymbol{T}$ = { 0, 1, 2, …, $n - 2$ }

Solution:

- No team can play itself:
  $\forall x \in \boldsymbol{X}, \forall t \in \boldsymbol{T}: \boldsymbol{A}[x, t] \neq x$

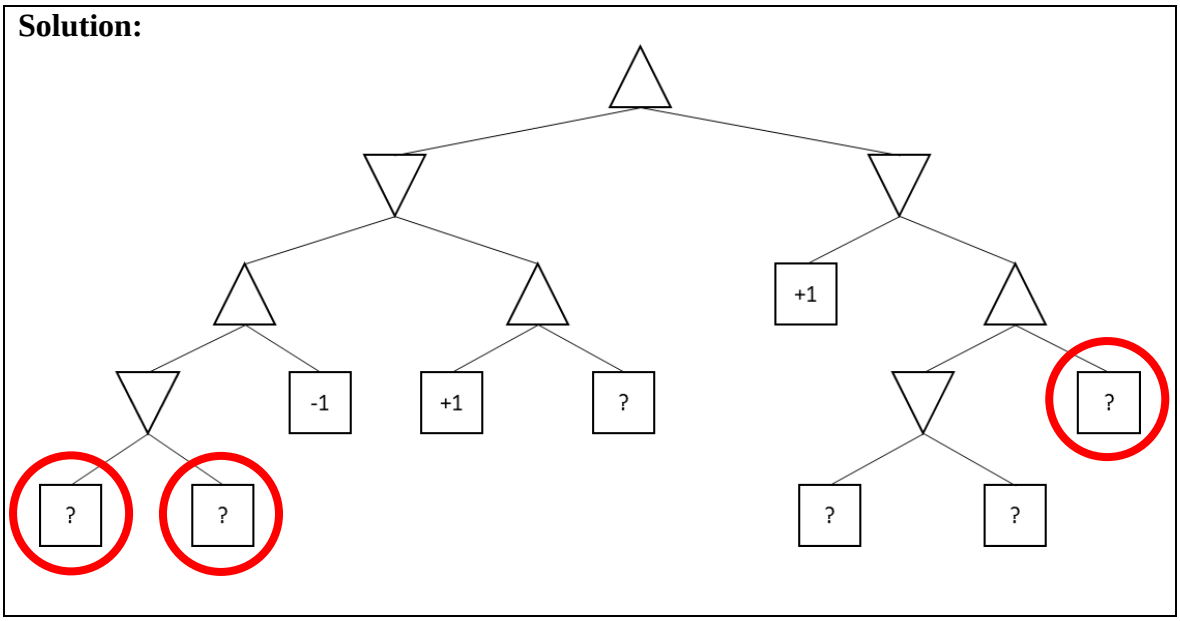
- For each time slot, all teams play exactly once:
  $\forall x \in \boldsymbol{X}, \forall t \in \boldsymbol{T}: \boldsymbol{A}[\boldsymbol{A}[x, t], \mathrm{t}] = x$

- All teams play against every other team exactly once:
  $\forall x \in \boldsymbol{X}: \texttt{AllDiff}(\{\boldsymbol{A}[x, t] \mid \forall t \in \boldsymbol{T}\})$
  // i.e., for each team x, opponents across all time slots are different.

3a. The ***Minimax*** algorithm will select the move that guarantees the best achievable utility, assuming a utility-optimising opponent. However, if two or more moves lead to the same utility, the algorithm will tend to choose the first best move it encounters deterministically.

(i) [2 marks] Given the game tree below, ***circle three or fewer leaf nodes that are marked with a question mark*** (i.e., the ? symbol) so as to assign a utility value of +1. The objective to ensure that ***both actions that may be taken by MAX at the root will result in a payoff of +1***. You are not given any additional information about the values of the leaf nodes that are not selected.

Solution:



Given the game tree depicted in Question **3a(i)** above, assume that both actions by MAX will result in the same payoff of +1. Notice that the Minimax algorithm, with pseudocode given below, will always choose the action that leads to the node on the left (i.e., backtracking will pick the first action with the highest utility). We wish to modify the algorithm such that if two nodes lead to the same utility, then we must choose between them randomly.

```
def minimax(state, player):
  if is_terminal(state):
    return evaluate(state)
  if isMax(player):
    best_value = -1 * INFINITY
    for child in get_children(state):
      value = minimax(child, MIN)
      if value > best_value:
        best_value = value
    return best_value
  elif isMin(player):
    best_value = INFINITY
    for child in get_children(state):
      value = minimax(child, MAX)
      if value < best_value:
        best_value = value
    return best_value
```

```
def find_best_move_for_max(state):
  best_move = None
  best_value = -1 * INFINITY

  for child in get_children(state):
    value = minimax(child, False)
    if best_move is None or \
       value > best_value:
      best_value = value
      best_move = child
  return best_move
```

(ii) [2 marks] One way to randomly select between child nodes with the same utility is to randomly shuffle all the child nodes with the maximum payoff before choosing one of them. However, if the ***Alpha-Beta Pruning*** algorithm is instead utilised, a drawback arises with this approach. What is the ***drawback*** of randomly shuffling the child nodes if Alpha-Beta Pruning is utilised?

Solution:

Less pruning will occur as move-ordering heuristics cannot be applied.

(iii) [3 marks] Modify the `find_best_move_for_max` function so that it ***chooses randomly from all the best moves***, instead of choosing the first best move it encounters. Assume that move ordering cannot be changed (i.e., randomly shuffling the children is not allowed). You may annotate the given pseudocode (which is duplicated in the solution box for your convenience) or describe the changes by stating any additional lines, removals, or replacements that you deem necessary to necessitate the required functionality.

Solution:

```
1  def minimax(state, player):
2    if is_terminal(state):
3      return evaluate(state)
4    if isMax(player):
5      best_value = -1 * INFINITY
6      for child in get_children(state):
7        value = minimax(child, MIN)
8        if value > best_value:
9          best_value = value
10     return best_value
11   elif isMin(player):
12     best_value = INFINITY
13     for child in get_children(state):
14       value = minimax(child, MAX)
15       if value < best_value:
16         best_value = value
17     return best_value
```

```
1  def find_best_move_for_max(state):
2    best_moves = []
3    best_value = -1 * INFINITY
4
5    for child in get_children(state):
6      value = minimax(child, False)
7      if value > best_value:
8        best_value = value
9        best_moves = [child]
10     elif value == best_value:
11       best_moves.append(child)
12   return random_choice(best_moves)
```

(iv) [5 marks] Choosing randomly from among all actions that give the best utility is not the best choice. Instead, if actions have the same utility, we wish to prioritise the action that will lead to the ***terminal state in the fewest number of moves***. This applies to both the MAX and MIN players, i.e., each player primarily aims to maximise or minimise their respective utility, but if two moves lead to equal utilities, then the secondary objective is to reach the terminal state in the fewest number of moves, ***even if that state is a losing state*** (recall that we are merely optimising utilities, not ensuring non-losing states).

For example, at the root node, MAX may have two actions. One action leads to a utility of +1 after 5 moves, and the other leads to a utility of -1 after 3 moves. Normal Minimax rules apply and the action that leads to a utility of +1 after 5 moves should be chosen. However, if both actions lead to +1 utility, but one action achieves that utility after 3 moves while the other action requires 5 moves, then the action that achieves the +1 utility after 3 moves should be chosen.

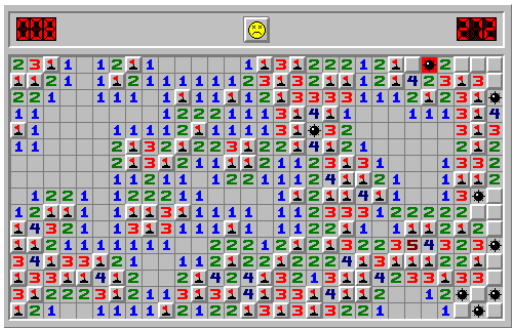
Describe the changes you would make to the pseudocode such that the move that results in a better utility is chosen first, and if two moves result in the same utility, the move that achieves that utility at a lower depth is chosen first. The changes you describe must be clear and specific.

Solution:

```
1 def minimax(state, player, d):
2   if is_terminal(state):
3     return (evaluate(state), d)
4   if isMax(player):
5     best_value = -1 * INFINITY
6     best_d = INFINITY
7     for child in get_children(state):
8       value, d = minimax(child, MIN, d+1)
9       if value > best_value:
10        best_value, best_d = value, d
11      elif value == best_value and \
              d < best_d:
12        best_value, best_d = value, d
13    return (best_value, best_d)
14  elif isMin(player):
15    best_value = INFINITY
16    best_d = INFINITY
17    for child in get_children(state):
18      value, d = minimax(child, MAX, d+1)
19      if value < best_value:
20        best_value, best_d = value, d
21      elif value == best_value and \
              d < best_d:
22        best_value, best_d = value, d
22    return best_value, best_d
```

```
1 def find_best_move_for_max(state):
2   best_move = None
3   best_value = -1 * INFINITY
4   best_d = INFINITY
5
6   for child in get_children(state):
7     value, d = minimax(child, False, 1)
8     if best_move is None or \
          value > best_value or \
          (value == best_value and \
          d < best_d):
9       best_value = value
10      best_d = d
11      best_move = child
12  return best_move
```

4a. In this question, you are tasked to create a logical agent to solve a simplified version of the Minesweeper game (depicted on the right), which is a logic puzzle video game that is generally played on personal computers.



The game features an n by m grid of covered cells, with k hidden ***mines*** scattered throughout the grid. The game rules are as follows.

- When a player uncovers a cell at (r, c), where r is the row index and c is the column index, if that cell ***does not*** contain a mine, it is a ***safe*** cell, and will reveal how many mines are in the 8 cells adjacent to it (i.e., on uncovering a safe cell at (r, c), the number of mines in the cells at coordinates (r+1, c), (r+1, c+1), (r, c+1), (r–1, c+1), (r–1, c), (r–1, c–1), (r, c–1), and (r+1, c–1) will be revealed).
- When a player uncovers a cell that is not a safe cell (i.e., it contains a mine), it will detonate the mine and cause the player to lose the game.
- The goal of the game is to uncover every safe cell on the grid (note that there will always be $nm - k$ safe cells given k mines). Once all the safe cells are uncovered (without detonating a mine), the player wins the game.

Consider the following example states on a 3 by 3 grid, where there is 1 mine.

State s_1

A	B	C
D	1	E
F	G	H

In this example state the player has uncovered the middle cell (only), which has revealed the value 1, meaning that there is exactly one (1) mine in the adjacent covered cells marked A, B, C, D, E, F, G, and H.

State s_3

1	**B**	C
1	1	E
F	G	H

Suppose that the player has uncovered two additional cells without losing. The resultant state is as shown. Here, we deduce that the mine is – since there must be 1 mine adjacent to A, but both the centre cell and D are safe – at B!

For this simplified version of the game, we will also assume the following.

- Only cells that the player uncovers will become uncovered; no additional cells will become uncovered except the one cell chosen to be uncovered by the player. (Note that in the original version of the game, certain cells would also be automatically uncovered for the player. However, this is not the case in the simplified version.)
- When a cell is uncovered, there will only be maximally ***one*** neighbouring cell containing mines.
- In the simplified version of the game, we do not know the value k (i.e., do not include any rules about k).
- You may only use the Boolean variables $X_{i,j}$, which represent the cell at (i, j) – i.e., the cell with row index i and column index j. If a mine is present at (i, j), then we have $X_{i,j}$, but if (i, j) is safe, then we instead have $\neg X_{i,j}$.

(i) [6 marks] Consider the following state of the simplified Minesweeper game, where unmarked cells denote cells that are still covered, while cells with integer value denote the uncovered cells. (Also, note the indices for each cell given on the right for your convenience.)

1		
1	1	

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)

You are given an incomplete knowledge base (KB) for the logical agent applied to the above state. Complete the specification of the KB. Ensure that all rules are in conjunctive normal form (CNF).

Solution:

R1: $\neg X_{0,0}$
R2: $\neg X_{1,0}$
R3: $\neg X_{1,1}$
R4: $\neg X_{0,0} \vee \neg X_{0,1}$
R5: $\neg X_{0,0} \vee \neg X_{0,2}$
R6: $\neg X_{0,0} \vee \neg X_{1,0}$
R7: $\neg X_{0,0} \vee \neg X_{1,2}$

At most 1 mine in neighbouring cell of $(1, 1)$:
R8: $\neg X_{0,1} \vee \neg X_{0,2}$
R9: $\neg X_{0,1} \vee \neg X_{1,0}$
R10: $\neg X_{0,1} \vee \neg X_{1,2}$
R11: $\neg X_{0,2} \vee \neg X_{1,0}$
R12: $\neg X_{0,2} \vee \neg X_{1,2}$
R13: $\neg X_{1,0} \vee \neg X_{1,2}$

At least 1 mine in neighbouring cell of (1, 1):
R14: $X_{0,0} \vee X_{0,1} \vee X_{0,2} \vee X_{1,0} \vee X_{1,2}$

At most 1 mine in neighbouring cell of $(0, 0)$:
R15: $\neg X_{0,1} \vee \neg X_{1,1}$
R16: $\neg X_{1,0} \vee \neg X_{1,1}$

Note that $\neg X_{0,1} \vee \neg X_{1,0}$ is duplicated from R9 so it does not need to be considered.

At least 1 mine in neighbouring cell of (0, 0):
R17: $X_{0,1} \vee X_{1,0} \vee X_{1,1}$

At most 1 mine in neighbouring cell of $(1, 0)$:
R18: $\neg X_{0,0} \vee \neg X_{1,1}$

Note that $\neg X_{0,0} \vee \neg X_{0,1}$ is duplicated from R4 $\neg X_{0,1} \vee \neg X_{1,1}$ is duplicated from R16 so it does not need to be considered.

At least 1 mine in neighbouring cell of (1, 0):
R19: $X_{0,0} \vee X_{0,1} \vee X_{1,1}$

(ii) [3 marks] Using resolution, infer that cell (0, 1) contains a mine.

Note that no error carried forward (ECF) from Question **4a(i)** *will be considered.*

Solution:

Inference for cell (0, 1) containing a mine: $\neg\alpha \equiv \neg X_{0,1}$
Let R20: $\neg X_{0,1}$

Resolution Steps:
Resolve R19: $(X_{0,0} \vee X_{0,1} \vee X_{1,1})$ with R20: $(\neg X_{0,1})$ → R21: $X_{0,0} \vee X_{1,1}$
Resolve R01: $(\neg X_{0,0})$ with R21: $(X_{0,0} \vee X_{1,1})$ with → R22: $X_{1,1}$
Resolve R03: $(\neg X_{1,1})$ with R22: $(X_{1,1})$ → R23: $\emptyset$

Therefore, since $KB \wedge \neg\alpha$ is unsatisfiable, and as such we infer that $(1, 0)$ contains a mine.

(iii) [3 marks] Assuming an agent function that utilises a KB and inference engine (IF) similar to those used in Questions **4a(i)** and **4a(ii),** describe how the agent function can formulate queries to send to the IF, and explain how these queries are linked to actions that may be taken in the simplified Minesweeper game.

Solution:

One possible way to perform inferences and, consequently, take actions would be to perform inference on cells where there is some information about their safety (i.e., cells that have neighbouring cells that are uncovered).

To do this, we will maintain 3 lists:

1. A list of cells that are uncovered (i.e., the cells that do not contain mines and have been uncovered), ***U***.
2. A list of cells that have been identified to contain mines (i.e., based on inferences) and thus, have not been uncovered, ***M***.
3. A list of active cells, ***A***, where a cell, *c*, is considered active if (i) at least one of its neighbouring cells is uncovered, (ii) it does not contain a mine (i.e., not in ***M***), and (iii) is itself not uncovered (i.e., not in ***U***).

For each cell *c* in ***A***, we perform two inferences:

i. *c* contains a mine.
ii. *c* does not contain a mine.

Consequently, if we can infer that *c* contains a mine, then we mark it as a mine by removing it from ***A*** and adding it to ***M***. Otherwise, if we can infer that *c* does not contain a mine, then we can uncover *c* and add it to ***U***. If c is uncovered, then any cell neighbouring c that has not already in ***A***, ***M***, or ***U***, is then added to ***A***.

When all cells are in ***M*** and ***U***. Then the game has been cleared.

If the game has not been cleared, but ***A*** is empty or we have already unsuccessfully attempted to make inferences on all cells in ***A***, then we must make guesses about which cells are safe.

/* When making guesses, we can use probabilities to make safer guesses. */

4b. [2 marks] Given the knowledge base, KB: X ⇔ Y, determine which, from among the following queries, cannot be entailed from KB.

Option **A**: ¬X ∨ Y

Option **B**: Y ⇒ X

Option **C**: X ∧ Y

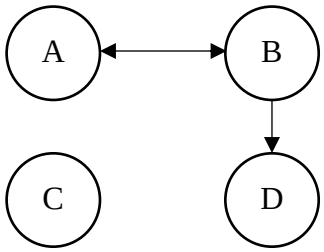
Option **D**: ¬X ∨ X

You may select any number of the above options, or else, simply specify (Option) **E** ***if none of the above are valid.***

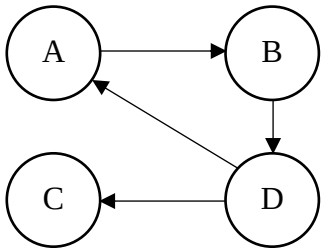
Solution: C

5a. [1 mark] From among all the following options, specify all valid Bayesian Networks.

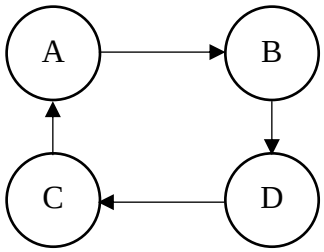
Option **A**



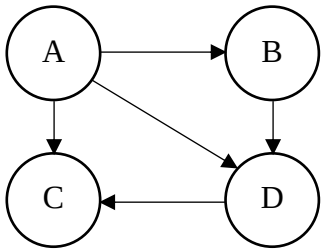
Option **B**



Option **C**



Option **D**



You may select any number of the above options, or else, simply specify (Option) **E** ***if none of the above are valid.***

Solution: D

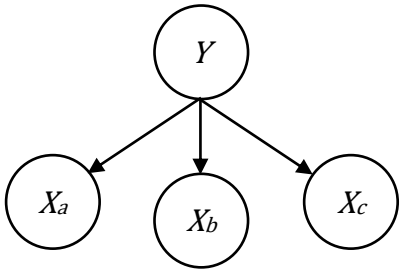
// Recall that Bayesian Networks correspond to DAGs (all options with cycles are invalid).

5b. Consider the following table, which contains performance data for 8 interns at a certain company. The data for the i-th intern is given in column i. Each intern is represented by 3 Boolean variables, X_a, X_b, and X_c, as well as one class label Y.

- If $X_a = 1$, then the intern did well in Task 1, else $X_a = 0$.
- If $X_b = 1$, then the intern did well in Task 2, else $X_b = 0$.
- If $X_c = 1$, then the intern did well in Task 3, else $X_c = 0$.
- The class label $Y = 1$ if the intern was offered a full-time position, else, $Y = 0$.

i	1	2	3	4	5	6	7	8
X_a	1	0	0	0	1	1	0	1
X_b	0	0	1	1	0	1	1	0
X_c	1	0	1	1	0	1	0	0
Y	0	0	0	0	1	1	1	1

Assume that a ***Naive Bayes*** model is to be utilised. More specifically, assume that we are to adopt the following Bayesian Network.



Note that no error carried forward (ECF) will be considered between Questions **5b(i)** *and* **5b(ii)**.

(i) [2 marks] Complete the following probability tables – i.e., fill in the last column in each of the tables given below.

Solution:

Y	P(Y)
0	0.5
1	0.5

Y	X_a	P(X_a|Y)
0	0	0.75
0	1	0.25
1	0	0.25
1	1	0.75

Y	X_b	P(X_b|Y)
0	0	0.5
0	1	0.5
1	0	0.5
1	1	0.5

Y	X_c	P(X_c|Y)
0	0	0.25
0	1	0.75
1	0	0.75
1	1	0.25

(ii) [4 marks] Assume that we observe the performance of a new intern, where $X_a = 1$, and $X_c = 0$. However, note that this intern was excused from Task 2, i.e., there is no value for X_b. Is this intern more likely to be offered a full-time position (i.e., have $Y = 1$)?

Option **A**: The intern is more likely to be offered a full-time position (i.e., have $Y = 1$).
Option **B**: The intern is more likely to not be offered a full-time position (i.e., have $Y = 0$).
Option **C**: Both $Y = 1$ and $Y = 0$ are equally likely.
Option **D**: There is not enough information to determine which outcome is more likely.

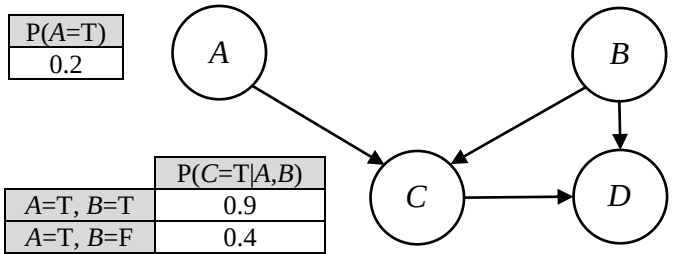
Solution: A

/*
$P(Y=1 \mid X_a=1, X_c=0)$
$= P(X_a=1, X_c=0 \mid Y=1)\, P(Y=1) / P(X_a=1, X_c=0)$
$= \alpha\, P(Y=1)\, P(X_a=1 \mid Y=1)\, P(X_c=0 \mid Y=1)$
$= \alpha * 0.5 * 0.75 * 0.75$
$= (9/32)\alpha = 0.28125\alpha$

$P(Y=0 \mid X_a=1, X_c=0)$
$= P(X_a=1, X_c=0 \mid Y=0)\, P(Y=0) / P(X_a=1, X_c=0)$
$= \alpha\, P(Y=0)\, P(X_a=1 \mid Y=0)\, P(X_c=0 \mid Y=0)$
$= \alpha * 0.5 * 0.25 * 0.25$
$= (1/32)\alpha = 0.03125\alpha$

As $P(Y=1 \mid X_a=1, X_c=0) > P(Y=0 \mid X_a=1, X_c=0)$, we pick Option A.
*/

5c. Consider the following Bayesian Network.



P(*A*=T)
0.2

P(*B*=T)
0.7

	P(*C*=T|*A*,*B*)
A=T, *B*=T	0.9
A=T, *B*=F	0.4
A=F, *B*=T	0.5
A=F, *B*=F	0.1

	P(*D*=T|*B*,*C*)
B=T, C=T	0.1
B=T, C=F	0.3
B=F, C=T	0.6
B=F, C=F	0.8

Note that the questions in parts **(i)** *to* **(iv)** *below are based on the above context* (Question **5c**). *No error carried forward (ECF) will be considered.*

(i) [1 mark] State the formula for the joint probability distribution induced by the above Bayesian Network – i.e., define the expression for P(*A*, *B*, *C*, *D*).

Solution:

P(*D* | *C*, *B*) P(*C* | *B*, *A*) P(*B*) P(*A*)

(ii) [2 marks] Calculate the probability of P(*A* = F, *B* = F, *C* = F, *D* = F).

Solution:

P(*D* | *C*, *B*) P(*C* | *B*, *A*) P(*B*) P(*A*)
= P(*D*=F | *C*=F, *B*=F) * P(*C*=F | *B*=F, *A*=F) * P(*B*=F) * P(*A*=F)
= 0.2 * 0.9 * 0.3 * 0.8
= 0.0432

(iii) [4 marks] Calculate the probability of P(D = T).

Solution:

P(D=T) = $\sum$ P(D=T | B, C) P(B, C)

P(B, C) = $\sum$ P (C, B | A) P(A)
= $\sum$ P (C | B, A) P(B | A) P(A)
= P(B) $\sum$ P (C | B, A) P(A)

Thus, we have:

- P(B=F, C=T) = (1 – 0.7) [(0.9)(0.2) + (0.1)(0.8)]
- P(B=F, C=F) = (1 – 0.7) [(1 – 0.4)(0.2) + (1 – 0.1)(0.8)]
- P(B=T, C=T) = (0.7) [(0.9)(0.2) + (0.5)(0.8)]
- P(B=T, C=F) = (0.7) [(1 – 0.9)(0.2) + (1 – 0.5)(0.8)]

Which gives us:

B	C	P(B, C)
T	T	0.406
T	F	0.294
F	T	0.048
F	F	0.252

Finally, we have:
P(D=T) = (0.406)(0.1) + (0.294)(0.3) + (0.048)(0.6) + (0.252)(0.8) = 0.3592

(iv) [2 marks] Calculate the probability of P(B = T, C = T).

Solution:

As calculated in the working from Part (iii) above:
P(B = T, C = T) = 0.406

END OF PAPER

BLANK PAGE

BLANK PAGE

BLANK PAGE