NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MIDTERM EXAMINATION FOR
Semester 1 AY2022/2023

CS3243: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

October 5, 2022 Time Allowed: 90 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment contains FOUR (4) questions. All the questions are worth a total of 60 MARKS. It is set for a total duration of 90 MINUTES. You are to complete all 4 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 answers questions with Yes/No/No Comment answers.
5. You may not communicate with anyone other than invigilators in any way.

STUDENT NUMBER: _____

EXAMINER'S USE ONLY		
Question	Mark	Score
1	17	
2	22	
3	11	
4	10	
TOTAL	60	

1. There are n philosophers dining together at a circular table (where $n \geq 2$). Each philosopher has their own place at the table. Their philosophical problem in this instance is that the dish served is a kind of spaghetti that must be eaten with two forks.

Each of the n philosophers has their own plate of spaghetti. However, there is only one fork between each plate (notice therefore that there are also n forks).

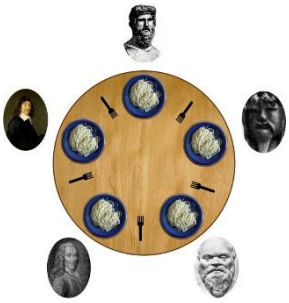


Image from Wikipedia

Each philosopher can only alternately think or eat. Moreover, a philosopher can only eat their spaghetti when they have both a left and right fork. Thus, two forks will only be available when their two nearest neighbours are thinking, not eating. After an individual philosopher finishes eating, they may put down both forks.

For better specificity, assume that ***each*** philosopher may apply any order to the following steps as the algorithm for that philosopher.

A_1: Pick up the left fork (if available).
A_2: Pick up the right fork (if available).
A_3: Eat (if holding both the left fork and right fork).
A_4: Put the left fork down (if held).
A_5: Put the right fork down (if held).
A_T: Think. (Note that this step is to be applied exactly $5n - 5$ times.)

Assume that different philosophers may use different sequences of steps. Further, assume that each action takes exactly 1 time unit. Also, notice that there are $5n$ - 5 instances of A_T in each algorithm – e.g., when $n = 3$, there are 3 philosophers, which implies that there are 10 A_T steps and 5 non-A_T steps (i.e., A_1, A_2, A_3, A_4, A_5) that may be specified in the algorithm for each philosopher. Note that there are no constraints over the sequence of steps (e.g., A_2 may occur before A_1, etc).

The objective is to define an algorithm for each of the n philosophers such that the **parallel** execution of the sequence of steps for all n philosophers results in none of the philosophers starving. Note that a philosopher will starve when that philosopher never gets an opportunity to eat.

Consider the following two algorithms for two philosophers.

Time Step	1	2	3	4	5	6	7	8	9	10
Philosopher 1	A_1	A_2	A_3	A_4	A_5	A_T	A_T	A_T	A_T	A_T
Philosopher 2	A_T	A_T	A_T	A_T	A_T	A_1	A_2	A_3	A_4	A_5

The parallel execution of the algorithms for both philosophers allows both to eat. However, with certain sequences, neither gets to eat.

Finally, assume that each philosopher's algorithm runs in a loop. This means then once we complete one iteration through the algorithm, it will loop and resume once more from the beginning. This looping occurs for all n philosophers. Again, note that the algorithms are run in parallel (i.e., all algorithms are run simultaneously).

(i) [6 marks] Define a search problem formulation for the given context that, when utilised, will result in a search tree that includes ***all possible algorithm permutations*** for all n philosophers. Your formulation should be efficient.

Solution:

- **State**
  - Each state is defined via a 2-dimensional array, $\boldsymbol{A}[i][j]$. The index of the outer array, i, corresponds to one of the n philosophers (e.g., index 0 to Philosopher 1, index 1 to Philosopher 2, and so on). The index of the inner array, j, corresponds to the subscript for the algorithm steps A_1, A_2, A_3, A_4, and A_5 only. The value stored at $\boldsymbol{A}[i][j]$ corresponds to the position in Philosopher i's algorithm where A_j will be executed (i.e., the algorithm step index for A_j).
  - Notice here that a sparse representation is used since among the $5n$ steps for each philosopher, only exactly 5 steps are not the A_T step (i.e., we need only store the 5 non-A_T steps are in each philosopher's algorithm).
  - All values assigned to the array at $\boldsymbol{A}[i]$ must be distinct and in the interval $[1, 5n]$.
  - For the **initial state**, all values in $\boldsymbol{A}$ are unassigned.
- **Action**
  - Assignment of one of the remaining algorithm step indices in the range $[1, 5n]$ to one of the possible algorithm steps, A_1, A_2, A_3, A_4, A_5, for a specific philosopher.
  - This requires a reference to those previously assigned algorithm step indices already assigned to ensure an efficient way to avoid duplicate assignments.
  - Actions only refer to a particular philosopher and to a particular position of one of the steps, A_1, A_2, A_3, A_4, and A_5, for the philosopher in question. This thus requires a fixed order over the n philosophers and a fixed order over the possible algorithm steps, A_1, A_2, A_3, A_4, A_5 (like the variable order assignments in a CSP).
- **Goal Test**
  - Ensure that A_3 is successfully executed for all n philosophers, which requires the specific algorithms to be executed in parallel, but only at leaf nodes (i.e., at the maximum depth).
  - Note the link to deadlock detection. However, students are not required to state this.
- **Transition Model**
  - Add the new step assignment (corresponding to the chosen action – i.e., the line the algorithm assigned to one of A_1, A_2, A_3, A_4, or A_5) to $\boldsymbol{A}[i][j]$.
- **Action Costs**
  - All actions have the same cost. Any constant non-negative value for all actions would be sufficient.

(ii) [3 marks] Given the search tree specified (assuming n is a positive integer), 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, m, as the maximum among these values.

Solution:

We have $5n$ possible locations (in the algorithm) for A_1, $5n - 1$ for A_2, $5n - 2$ for A_3, $5n - 3$ for A_4, and $5n - 4$ for A_5 for each philosopher. These are possible values for b. Since only 5 assignments must be made for each philosopher, $m = 5n$.

You may assign the algorithms in parallel as well. However, you need to consider all possible algorithms for one philosopher with one option for another philosopher. This would result in $b = (5n)^n$ and $m = 5$. The search space is the same. Probably better to avoid the parallelism here.

Further, notice that defining a search tree to assign steps to all $5n$ algorithm positions would yield a search tree with $b \in [1, 6]$ and $m = 5n^2$, which is a less efficient representation that the one given above.

(iii) [2 marks] Write an expression in terms of n for the size of this search tree and comment on how realistic it would be to perform such a search.

Solution:

Sum of geometric series = $a + ar + ar^2 + \ldots + ar^{p-1} = a(r^p - 1) / (r - 1)$

Thus, the size of the search tree is $(r^p - 1) / (r - 1)$, where $r = 5n$ and $p = 5n + 1$.

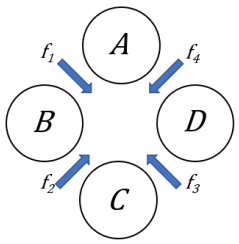
With $n = 10$, we have a tree of size $(50^{51} - 1) / 49$ which is unmanageably large. It is therefore unreasonable to solve this problem using uninformed search.

(iv) [4 marks] Recall that once the algorithm for each philosopher is specified, these algorithms are executed, in parallel, in a loop. Given n philosophers, state the minimum number of steps, k, that must be executed before all the philosophers would get a chance to eat at least once. Determine this based on the assumption that the algorithms specified for the n philosophers are optimal – i.e., that these algorithms will allow all philosophers to eat within k steps. Prove (with a detailed proof sketch) the validity of your answer upper bounded to the nearest 5 instructions/steps.

Solution:

Assume the base case, where $n = 2$. Here, one philosopher must eat first, then the other. 10 steps are necessary (here we assume the steps to put down the forks are also considered).

When $n = 3$, we again notice that all 3 philosophers must eat in sequence. This is because no matter which 1 philosopher eats first, the other 2 philosophers still share a fork, and must thus still be sequenced. We require 15 steps here.



However, when $n = 4$, this time, when one philosopher eats the other there is another philosopher on the other side of the table that does not need to wait since both forks are available. For example, with the diagram on the left, A may use f_1 and f_4, while C uses f_2 and f_3. Or B may use f_1 and f_2, while D uses f_3 and f_4. We require 10 steps

When n = 5, say we have A, B, C, D, E. Notice here that B and D may eat, but afterward, only C and A OR E may eat, but NOT both A AND E.

We may thus inductively show that, given $n \geq 2$:

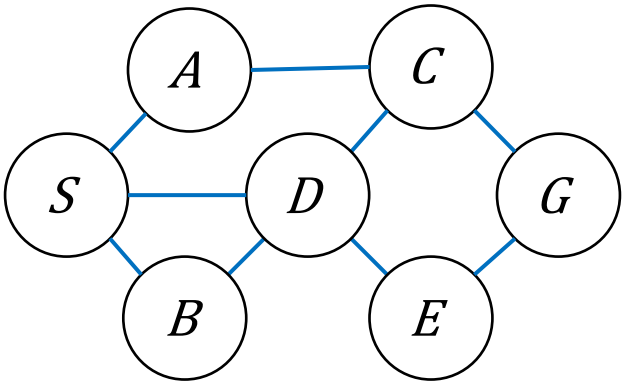
- when n is even, we require minimally 10 steps, and
- when n is odd, we require minimally 15 steps.

(v) [2 mark] Which of breadth-first and depth-first search is more appropriate here? Why?

Solution:

Depth-first search (DFS) is more appropriate. Given the known and fixed maximum depth, DFS is complete. It has a lower space complexity than BFS.

2a. [5 marks] Consider the undirected graph depicted in the Figure below. Let ***S*** be the initial state and ***G*** be the goal state.



Trace the sequence of nodes popped from the frontier when employing depth-limited search based on a tree-search implementation. Show the contents of the frontier for each iteration.

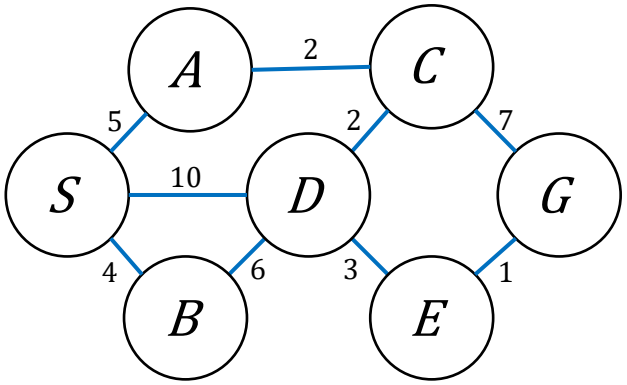
Assume that the maximum depth is 2, where the depth of the start node is 0. Further, assume that ties, when pushing to the frontier, are resolved based on **reverse alphabetical order** (e.g., ***B*** is pushed before ***A***).

Solution:

Frontier = Stack	Popped from Frontier
{(S,0)}	-
{(A,1), (B,1), (D,1)}	S
{(C,2), (S,2), (B,1), (D,1)}	S-A
{(S,2), (B,1), (D,1)}	S-A-C
{(B,1), (D,1)}	S-A-C-S
{(D,2), (S,2), (D,1)}	S-A-C-S-B
{(S,2), (D,1)}	S-A-C-S-B-D
{(D,1)}	S-A-C-S-B-D-S
{(B,2), (C,2), (E,2), (S,2)}	S-A-C-S-B-D-S-D
{(C,2), (E,2), (S,2)}	S-A-C-S-B-D-S-D-B
{(E,2), (S,2)}	S-A-C-S-B-D-S-D-B-C
{(S,2)}	S-A-C-S-B-D-S-D-B-C-E
{}	S-A-C-S-B-D-S-D-B-C-E-S

Answer: S-A-C-S-B-D-S-D-B-C-E-S

2b. Consider the undirected 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. The heuristic values of each state are given the table next to the figure.



n	h(*n*)
S	12
A	5
B	7
C	6
D	3
E	1
G	0

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, trace the given algorithms in terms of the sequence of nodes popped from the frontier. Show the contents of the frontier and reached hash table for each iteration.

(i) [4 marks] Breadth-first Search implemented using a graph-search implementation.

Solution:

Should assume graph search version 1 and early-goal testing.

Frontier = Queue	Reached	Popped from Frontier
{S}	S	S
{A, B, D}	S, A, B, D	S-A
{B, D, C}	S, A, B, D, C	S-A-B
{D, C}	S, A, B, D, C	S-A-B-D
{C, E}	S, A, B, D, C, E	S-A-B-D-C
{E, G}	S, A, B, D, C, E, G	(Optional for clarity)

Answer: S-A-B-D-C

(ii) [5 marks] A* search implemented using a graph-search implementation (version 1).

Solution:

Should assume graph search version 1 and late-goal testing.

Frontier = Priority Queue	Reached	Popped from Frontier
{(S,12(-))}	S	S
{(A,10(S)), (B,11(S)), (D,13(S))}	S, A, B, D	S-A
{(B,11(S)), (C,13(S-A)), (D,13(S))}	S, A, B, D, C	S-A-B
{(C,13(S-A)), (D,13(S))}	S, A, B, D, C	S-A-B-C
{(D,13(S)), (G,14(S-A-C))}	S, A, B, D, C, G	S-A-B-C-D
{(E,14(S-D)), (G,14(S-A-C))}	S, A, B, D, C, G, E	S-A-B-C-D-E
{(G,14(S-A-C))}	S, A, B, D, C, G, E	S-A-B-C-D-E-G

Answer: S-A-B-C-D-E-G

(iii) [2 mark] What is the optimal path from node ***S*** to node ***G***? Why didn't **(ii)** give this?

Solution:

Answer: S-A-C-D-E-G

Since ***graph search version 1*** was implemented, an optimal solution is ***not*** guaranteed. This is because graph search version will only consider the first path to each node, even if it is not the optimal one.

(iv) [2 mark] Is the heuristic given consistent? Justify your answer.

Solution:

No. The heuristic given is ***inconsistent***.

For a heuristic to be consistent, given any node n, and a successor, n', we require:
$h(n) \leq d(n, n') + h(n')$, where $d(n, n')$ is the actual cost from n to n'

However, we may observe the counter example h(S) > d(S, B) + h(B) since h(S) = 12, d(S, B) = 4, and h(B) = 7.

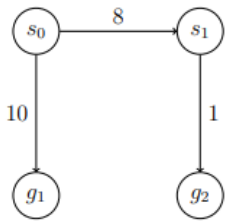
(v) [4 mark] Suppose we use a variant of the A* search algorithm that instead utilises $f(n) = w \times g(n) + (1 - w) \times h(n)$, where $0 < w < 1$ (instead of $f(n) = g(n) + h(n)$). For any value of w, an optimal path to a goal will be output whenever h is a consistent heuristic (assuming that this algorithm is implemented using graph search version 1).

Determine if the statement above is True or False and provide a rationale.

Solution:

Answer: False.

Consider the example on the right.



Let the initial state be s_0, and the goal states be g_1 and g_2.

Further, assume $h(s_0) = 9$, $h(s_1) = 1$, and $h(g_1) = h(g_2) = 0$.

Notice that h is both admissible and consistent.

However, note that with this case, when we pop s_0 and push s_1 and g_1, we have:

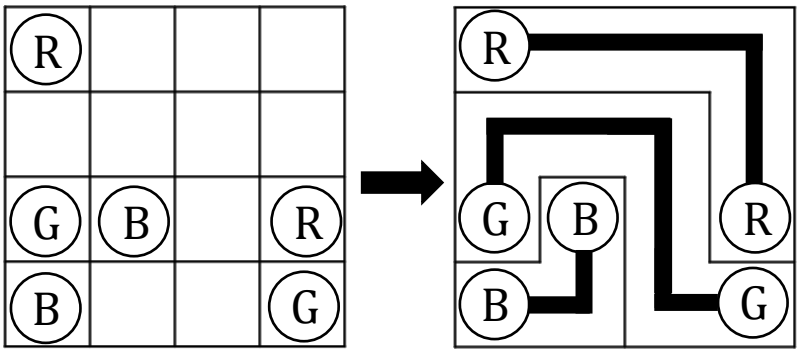
- $f(g_1) = 10w + 0(1 - w) = 10w$
- $f(s_1) = 8w + 1(1 - w) = 7w + 1$

For any value $10w < 7w + 1$, or rather, $w < 1/3$, A* would explore g_1 before s_1 and thus output the non-optimal path.

Other acceptable answers are as follows.

- With h = 0 (which is admissible and consistent), we essentially have the UCS algorithm, which is not optimal.
- When w = 0.5 (only), we essentially have the A* algorithm, which is not optimal under graph search version 1.

3. Flow Free is a puzzle game played on a 2-dimensional grid. The game begins with *n* pairs of similarly coloured cells, called the ***source cells***. The goal is to fill all empty grid cells with colours such that ***pipes*** are formed between each pair of similarly coloured source cell. The diagrams below correspond to a particular instance of the puzzle, where the left corresponds to the ***initial state*** (with only source cells filled), while the right corresponds to a ***goal state***.



For the above diagram, let R denote red, G denote green, and B denote Blue.

The specific Rules for Flow Free are as follows.

- There are 2*n* source cells in each puzzle comprising *n* distinct colours. Each distinct colour will be represented by exactly 2 source cells.
- A valid action in the game is to fill an empty cell with a colour corresponding to one of the source cells.
- Orthogonally adjacent cells of the same colour are said to be ***connected***. As suggested earlier, diagonal connections are not to be considered.
- A ***pipe*** is formed when two **source cells** of the same colour are connected.
- Each cell may only contain one colour. Consequently, any two pipes **cannot intersect**.
- The goal state **contains no empty cells** and **has a pipe between each pair of similarly coloured source cells**.

You should assume that each state corresponds to the grid and its current contents (i.e., source cell assignments and intermediate colour assignments). Assume that all puzzles are solvable.

(i) [2 marks] A heuristic for this game is: *n* - *m*, where *m* corresponds to the number of existing pipes. Prove that this heuristic is admissible. Additionally, provide the relaxed rule(s) that achieve this heuristic.

Solution:

The given heuristic corresponds to the number of colours that ***do not have pipes formed***. Since ***at least one action*** (i.e., at least filling one cell) is necessary to complete the pipe for each such colour, this heuristic must be admissible.

The relaxed rule must allow the following.

- One action is to form a pipe between any one pair of sources of the same colour that does not already have a pipe.

(ii) [5 marks] Let d_c denote the Manhattan distance between the source cells with the colour c. Let x_c denote the number of cells on the grid with colour c. Let C be the set of all colours present in the given puzzle. Another heuristic is defined as follows.

$$h(n) = \sum_{c \in C} max(d_c - x_c, 0),$$

where C, and each d_c, x_c for $c \in C$, is determined based on state n.

Prove/Disprove the admissibility of this heuristic.

Solution:

The sum in the heuristic is taken over all colours (i.e., each pair of source cells of the same colour). We will show that for each colour the value calculated is less than or equal to the true cost.

Notice that x_c corresponds to the moves already made - 2 (for the 2 source cells). Eventually, once a pipe is formed, the number of actions taken to form it is x_c - 2. Let us denote y_c* such that y_c* - 2 is the eventual cost of the pipe to be formed.

The shortest pipe that can be formed between two sources must be the Manhattan distance between the sources plus one since connections may only be formed via orthogonally adjacent cells.

Let us consider the following cases for each colour.

- Shortest pipe (i.e., $y_c^* = d_c + 1$): initially, only the source cells are present for the colour in question, and we have $x_c = 2$. Exactly $(d_c + 1) - x_c$ more cells must be filled to complete the pipe. As we progress to complete this pipe, x_c increases and $(d_c + 1) - x_c$ decreases (exactly by the same amount). When the pipe is formed, $d_c + 1 = x_c$, and $(d_c + 1) - x_c = 0$. The value exactly expresses the number of actions necessary. Thus, $max(d_c - x_c, 0)$ is admissible for such a pipe since the value is always 1 less than actual but never below 0.
- Non-optimal pipe (i.e., $y_c^* > d_c + 1$): here, we have the same initial case as the shortest pipe. However, with the non-optimal pipe, eventually, we have $x_c \geq d_c + 1 > d_c$. We know that $y_c^* > (d_c + 1) > d_c$, and $x_c = [2, y_c^*]$. We can infer the following.
    - When $x_c = 2$, $d_c - x_c = d_c - 2 < y_c^* - 2$ (since $d_c < d_c + 1 < y_c^*$).
    - When $x_c \leq d_c + 1$, $(d_c + 1) - x_c < y_c^* - x_c$ (since $(d_c + 1) < y_c^*$).
    - When $x_c > d_c + 1$, $(d_c + 1) - x_c < 0$, but $max(d_c - x_c, 0) = 0$, and since $x_c \leq y_c^*$, $y_c^* - x_c \geq 0$.

  In other words, since an actual pipe formed may need to be longer (e.g., to cover more of the grid), the Manhattan distance will be an under-estimate. This means that $d_c - x_c$ may be negative. However, since we take $max(d_c - x_c, 0)$, the resultant value is in the interval $[0, d_c]$.

The resultant value for each colour is thus equal to or less than the true cost (i.e., length of the actual pipe less 2 (accounting for the source cells)).

(iii) [4 marks] Determine a heuristic that dominates the heuristics described in parts **(i)** and **(ii)**. Describe the usefulness all three heuristics. More specifically, determine if these heuristics would indeed be useful in pruning the search space for the A* search algorithm.

Solution:

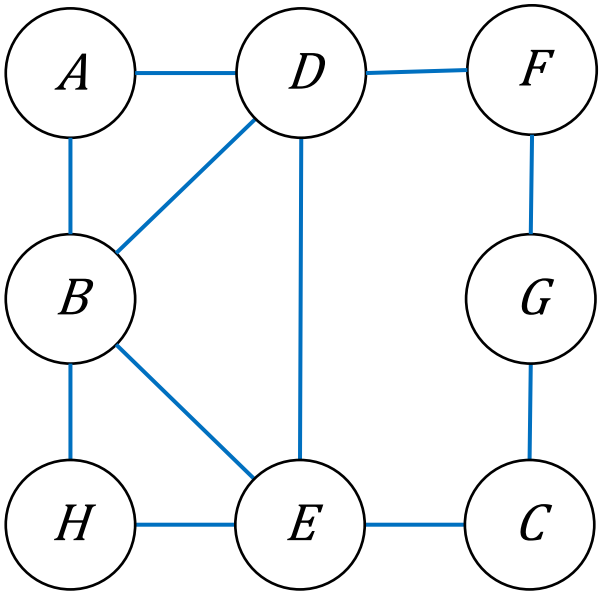
Notice that each problem requires the grid to be filled. Consequently, each path to a solution requires the exact same number of actions, i.e., size of grid - x_c. Such a heuristic would dominate both **(i)** and **(ii)** as it would exactly predict the number of moves that would be required to fill the board.

Unfortunately, these heuristics require the search to traverse to terminal states since they do not consider the one important rule: **pipes cannot intersect**.

Using heuristics **(i)**, **(ii)**, or the heuristic defined above, would thus equate to a systematic search through the search space.

A better solution would be to make inferences about terminal states and to avoid search subtrees that are terminal (similar to the use of inference in CSPs). However, this may be compllicated to implement.

4a. [4 marks] Consider the following graph, where the goal node is ***G***. The evaluation function values of each state are given the table next to the figure. Assume that smaller values are evaluated as better, and that 0 is the smallest value possible.



n	f(n)
A	2
B	7
C	2
D	3
E	8
F	5
G	0
H	1

Assume that a 2-beam local search algorithm is implemented such that states are never revisited (i.e., via graph search version 3 – implemented over the beam). Trace the execution of this algorithm on the above graph when the initial states are ***A*** and ***H***.

Assume that ties are broken based on alphabetical order. Your trace should include the states selected, the successors under consideration, and the states in the reached hash table for each iteration.

Solution:

Should assume graph search version 3. Either early-goal testing (as shown below), or late-goal testing are acceptable.

Beam	Popped (in reached)	Successors Considered
{(A,2), (H,1)}	A, H	{(B,7), (B,7), (D,3), (E,8)}
{(D,3), (B,7)}	A, H, D, B	{~~(A,2)~~, ~~(A,2)~~, ~~(B,7)~~, ~~(D,3)~~, (E,8), (F,5), ~~(H,1)~~}
{(E,8), (F,5)}	A, H, D, B, E, F	{~~(B,7)~~, (C,2), ~~(D,3)~~, ~~(D,3)~~, (G,0), ~~(H,1)~~} Goal Found.
Note nodes struck out implies already visited (i.e., in reached).		

4b. [4 Marks] Assuming an infinite search space with finite branching factor that contains a solution, prove that local search conducted using hill-climbing with k random restarts is incomplete (where k is a constant positive integer). Assume tree search is adopted.

Solution:

In general, completeness requires the algorithm to find a solution or else terminate. The latter is impossible given the infinite search space and tree search implementation. We also need to show that at least in one case, finding a solution given the assumptions is impossible.

Assume that there are c starting points that would lead to a solution being found. We may define the probability of finding this solution as $f(x) = c/x$, where x is the size of the search space. Given that $f(x)$ in the limit of positive infinity is 0, we may define the probability of finding a solution in the given case to be 0.

Another perspective is that c/x is very small but non-zero value. However, for the algorithm to find a solution, it must execute enough restarts, k, such that $k(c/x) \geq 1$. Given small c and infinite x, we need infinite k to ensure this, which also suggests incompleteness.

4c. [2 Marks] Briefly describe how the context given in **Question 1** may be specified as a local search problem by stating a state representation and heuristic.

Solution:

A complete state formulation where each philosopher's algorithm corresponds to a randomised sequence of the $5n$ steps.

The heuristic used could counts the number of deadlocks present – i.e., adjacent philosophers that are both waiting for a fork.

END OF PAPER

BLANK PAGE

BLANK PAGE

BLANK PAGE