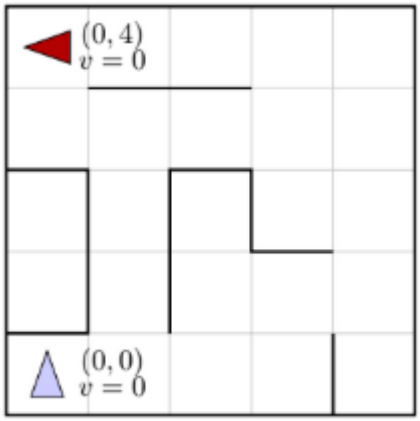
Q1. [26 pts] Search and Heuristics



Imagine a car-like agent wishes to exit a maze like the one shown above. The agent is directional and at all times faces some direction $d \in (N, S, E, W)$. With a single action, the agent can *either* move forward at an adjustable velocity v or turn.

The moving actions are *faster*, *maintain* and *slower*. For these actions, the agent then moves a number of squares equal to its new adjusted velocity. Let v denote the agent's current velocity and let v' denote the agent's new adjusted velocity.

- *Faster* : $v' = v + 1$
- *Slower* : $v' = v - 1$
- *Maintain*: $v' = v$

The turning actions are *left* and *right*, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero. Turning leaves the speed at zero.

- *Left*: change the agent's direction by 90 degrees counterclockwise
- *Right*: change the agent's direction by 90 degrees clockwise

For example, if the agent is currently on (0, 0) facing north with velocity 0 (as pictured) and wants to get to (2, 0) facing east with velocity 0, the sequence of actions will be: *right, faster, maintain, slower*.

Illegal actions include

- Any action that would result in a collision with a wall (i.e. there is a wall between the current position and the position you would be in if you took said action)
- Any action that would reduce v below 0 (slowing when v=0) or above a maximum speed V_{max} • Maintaining a velocity of 0
- Turning when velocity *6*= 0

The agent's goal is to find a plan which parks it ($v = 0$) in the goal direction on the exit square using as few actions (time steps) as possible. Note that the cost of a path is defined by the number of actions the agent takes.

(a) [3 pts] Suppose the agent wants to take the leftmost path (i.e., the one that passes through (1,2)) from the start (0,0) facing north to the goal (0,4) facing west. Write down the shortest sequence of actions for it to take.

Actions:

Khởi đầu: (0,0), Bắc, v=0

1. Trước: (0,0), Bắc, v=0
   Hành động: Faster (v: 0 → 1) — sau khi điều chỉnh v' = 1, di chuyển 1 ô về hướng Bắc.
   Sau: (0,1), Bắc, v=1

2. Trước: (0,1), Bắc, v=1
   Hành động: Maintain (v: 1 → 1) — di chuyển 1 ô về hướng Bắc.
   Sau: (0,2), Bắc, v=1

3. Trước: (0,2), Bắc, v=1
   Hành động: Slower (v: 1 → 0) — sau khi giảm v' = 0, di chuyển 0 ô (không dịch chuyển).
   Sau: (0,2), Bắc, v=0

4. Trước: (0,2), Bắc, v=0
   Hành động: Right — quay phải 90° (chỉ được phép khi v=0), hướng từ Bắc → Đông, v vẫn = 0.
   Sau: (0,2), Đông, v=0

5. Trước: (0,2), Đông, v=0
   Hành động: Faster (v: 0 → 1) — di chuyển 1 ô về hướng Đông.
   Sau: (1,2), Đông, v=1
   (đã đi qua ô (1,2) như yêu cầu đường trái)

6. Trước: (1,2), Đông, v=1
   Hành động: Slower (v: 1 → 0) — v' = 0, di chuyển 0 ô.
   Sau: (1,2), Đông, v=0

7. Trước: (1,2), Đông, v=0
   Hành động: Left — quay trái 90°, hướng Đông → Bắc, v=0.
   Sau: (1,2), Bắc, v=0

8. Trước: (1,2), Bắc, v=0
   Hành động: Faster (v: 0 → 1) — di chuyển 1 ô lên (Bắc).
   Sau: (1,3), Bắc, v=1

9. Trước: (1,3), Bắc, v=1
   Hành động: Maintain (v: 1 → 1) — di chuyển 1 ô lên.
   Sau: (1,4), Bắc, v=1

10. Trước: (1,4), Bắc, v=1
    Hành động: Slower (v: 1 → 0) — v' = 0, di chuyển 0 ô.
    Sau: (1,4), Bắc, v=0

11. Trước: (1,4), Bắc, v=0
    Hành động: Left — quay trái 90°, hướng Bắc → Tây, v=0.
    Sau: (1,4), Tây, v=0

12. Trước: (1,4), Tây, v=0
    Hành động: Faster (v: 0 → 1) — di chuyển 1 ô về hướng Tây.
    Sau: (0,4), Tây, v=1

13. Trước: (0,4), Tây, v=1

Hành động: Slower (v: 1 → 0) — v' = 0, di chuyển 0 ô.
Sau: (0,4), Tây, v=0 ← Mục tiêu đạt được

(b) [3 pts] If the grid is M by N and the maximum speed is V_{max}, what is the size of the state space? You should assume that all configurations are reachable from the start state.

State Space Size: |S| = M.N.4.(Vmax + 1)

(c) [3 pts] A "child" of a state *s* is any other state s^0reachable via a legal action from state *s*. Is it possible that a state in the state space has no children? If so, give an example of such a state. If not, briefly explain why every state must have at least one child.

Yes, có thể có một trạng thái không có "child" nào.
Example State or Explanation:

Một child của trạng thái sss là trạng thái s′s's′ đạt được từ sss qua một hành động hợp lệ.
Một hành động hợp lệ chỉ tồn tại nếu:

1. Không va chạm tường,
2. Không làm v < 0 hoặc v > Vmax
3. Không "Maintain" khi v = 0
4. Không "Turn" khi v ≠ 0

Vì vậy, một trạng thái không có child có thể xảy ra nếu không hành động nào hợp lệ.

(d) [4 pts] What is the maximum branching factor of this problem? Draw an example state (x, y, orientation, velocity) that has this branching factor, and list the set of available actions. For example, in the above picture, if the agent was in (0, 0) facing North with a velocity of v = 0, the branching factor would be 2. The agent could turn left or right (but not go faster since it would hit a wall).

Illegal actions are simply not returned by the problem model and therefore not counted in the branching factor. You do not necessarily have to use the example grid above. If you need to include a drawing of your own, label properly and make sure it fits in the solution box.

Maximum Branching Factor: 3

Maximum Branching Example State and Available Actions:
Example State: (2, 2), facing North, v = 0
Available Actions: {Faster, Left, Right}

4

(e) [4 pts] Is the Manhattan distance from the agent's location to the exit's location admissible?

If not, draw an example state (x, y, orientation, velocity) where this heuristic overestimates at that state, and specify: 1) the heuristic value at that state and 2) the actual cost from that state to the goal.

You do not necessarily have to use the example grid above. Make sure to label your drawing, including the goal state (location, orientation, speed) and action sequence, and fit it into the solution box.

No

Example State, Heuristic Value, Actual Cost:
Example State: (0, 0), facing North, v = 3
Heuristic Value: 1
Actual Cost: 3

(f) [4 pts] Is the following heuristic admissible? *Manhattan distance / V_{max}.*

If yes, state why. If not, draw an example state (x, y, orientation, velocity) where this heuristic overestimates

at that state, and specify: 1) the heuristic value at that state and 2) the actual cost from that state to the goal.

You do not necessarily have to use the example grid above. Make sure to label your drawing, including the goal state (location, orientation, speed) and action sequence, and fit it into the solution box.

No

Example State, Heuristic Value, Actual Cost:

Example State: (0, 0), facing North, v = 0
Goal: (0, 4), facing North, v = 0
Vmax = 2

Heuristic Value: Manhattan distance / Vmax = 4 / 2 = **2**
Actual Cost:

- Faster (v = 1)
- Maintain (move 1)
- Faster (v = 2)
- Maintain (move 2 → reach (0,4))
- Slower (v = 1)
- Slower (v = 0)

→ **6 actions total**

(g) [1 pt] If we used an inadmissible heuristic in A* Tree search, could it change the completeness of the search? Assume the graph is finite and the heuristic is non-negative.

No

(h) [1 pt] If we used an inadmissible heuristic in A* Tree search, could it change the optimality of the search? Assume the graph is finite and the heuristic is non-negative.

Yes

(i) [3 pts] Which of the following may be a good reason to use an inadmissible heuristic over an admissible one? Select all that apply.

An inadmissible heuristic may be easier to compute, leading to a faster state heuristic computation time. True

An inadmissible heuristic can be a closer estimate to the actual cost (even if it's an overestimate) than an admissible heuristic, thus exploring fewer nodes. True

An inadmissible heuristic will still find optimal paths when the actual costs are non-negative. False

An inadmissible heuristic may be used to completely block off searching part of a graph in a search algorithm. True

Q2. [15 pts] Search Nodes

Consider the tree search (i.e. no explored set) of an arbitrary search problem with max branching factor *b*. Each search node *n* has a backward (cumulative) cost of $g(n)$, an admissible heuristic of $h(n)$, and a depth of $d(n)$. Let n_c be a minimum-cost goal node, and let n_s be a shallowest goal node.

For each of the following, give an expression that characterizes the set of nodes that are explored before the search terminates. For instance, if we asked for the set of nodes with positive heuristic value, you could say: for all *n*, such that $h(n) \geq 0$. Don't worry about ties (so you won't need to worry about > versus ≥). If there are no nodes for which the expression is true, you must write "none."

Note that you are not required to use all the functions given, *f*, *g*, and *h*, in your inequality.

(a) [5 pts] Give an inequality in terms of the functions *g*, *h*, and *d*, as well as the nodes n_c and n_s defined above to describe the nodes *n* that are explored in a breadth-first search before terminating.

Inequality: All *n*, such that :
$d(n)<d(n_s)$

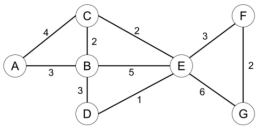
(b) [5 pts] Give an inequality in terms of the functions *g*, *h*, and *d*, as well as the nodes n_c and n_s defined above to describe the nodes *n* that are explored in a uniform cost search before terminating.

Inequality: All *n*, such that:
$g(n)<g(n_c)$

(c) [5 pts] Now for this question, assume the heuristic *h* is consistent. Give an inequality in terms of the functions *g*, *h*, and *d*, as well as the nodes n_c and n_s defined above to describe the nodes *n* that are explored in an A* search before terminating.

Inequality: All *n*, such that:
$g(n)+h(n)<g(n_c)$

Q3. [25 pts] Searching a Graph



Node	h_1	h_2
A	12	11
B	6	7
C	9	6
D	3	4
E	3	5
F	2	1
G	0	0

Consider the graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. The graph is bi-directional so each edge can be traversed from either direction. Please refer to the search algorithms exactly as presented on the lecture slides as the ordering of the actions matters.

(a) [15 pts] For each of the following graph search strategies, mark with an X which (if any) of the listed paths it could return. Note that for some search strategies the specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark all paths that could be returned under some tie-breaking

scheme. If a graph search strategy returns a path not listed, write out the correct path in the *Other* column.

Algorithm	A-C-E-G	A-C-E-F-G	A-B-D-E-F-G	Other
UCS	(i)	(ii) X	(iii)	(iv)
Greedy with heuristic h_1	(v)	(vi)	(vii)	(viii)X
Greedy with heuristic h_2	(ix) X	(x)	(xi)	(xii)
A* with heuristic h_1	(xiii)	(xiv)	(xv) X	(xvi)
A* with heuristic h_2	(xvii)	(xviii)X	(xix)	(xx)

(b) [2 pts] What is the cost of the optimal path for uniform cost search from A to G?

Answer: A–C–E–F–G → 4 + 2 + 3 + 2 = 11

(c) [4 pts] Is h_1 admissible? Is it consistent?

Admissible: No

Consistent: No

(d) [4 pts] Is h_2 admissible? Is it consistent?

Admissible: Yes

Consistent: No

Q4. [23 pts] Search: Multiple Choice and Short Answer Questions

(a) [18 pts] Consider the following true/false questions with each question worth 2 points. For the following search problems, assume every action has a cost of at least , with > 0. Assume any heuristics used are consistent.

Depth-first tree-search on a finite graph is guaranteed to be complete.
False

Breadth-first tree-search on a finite graph is guaranteed to be complete.
True

Iterative deepening tree-search on a finite graph is guaranteed to be complete.
True

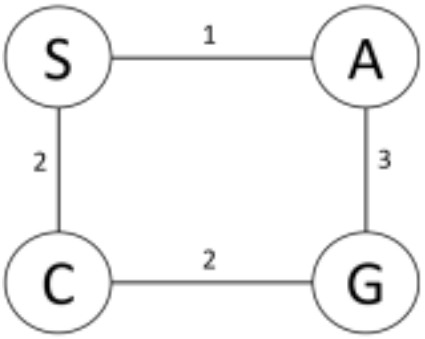
For all graphs without cycles, graph-search contains a larger frontier than tree-search.

False

Iterative deepening graph-search has the time complexity of BFS and the space complexity of DFS.
True

If h_1(s) is a consistent heuristic and h_2(s) is a consistent heuristic, then min(h_1(s), h_2(s)) must be consistent.
True

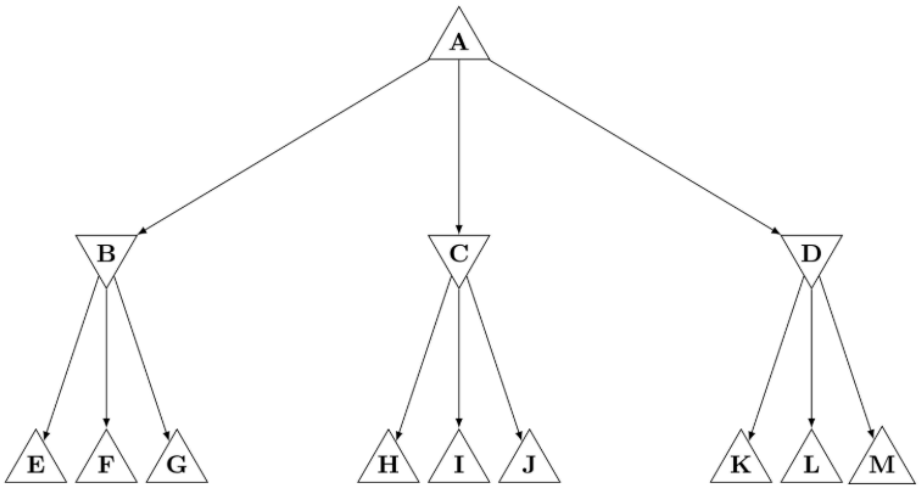
(b) [5 pts] Consider the state space graph shown below. S is the start state and G is the goal state. The costs for each edge are shown on the graph. For the following table below, fill in potential heuristic values such that the heuristic is admissible but not consistent.



State	h(s)
S	4
A	3
C	1
G	0

Heuristic Function

Q5. [11 pts] Alpha-Beta Pruning with Iterative Deepening



Evaluation Function	
State, s	$f(s)$
B	4
C	14
D	19
E	5
F	7
G	8
H	17
I	16
J	15
K	23
L	20
M	26

Iterative deepening may be combined with pruning in game trees to increase the pruning and speed up the search. To see how this works, we are going to prune the above fragment of a tree using the following steps:

1. Select the order of the min-node children, *B*, *C*, *D*, based on the values of the evaluation function of those states, *f*(*B*), *f*(*C*), *f*(*D*). Specifically, using the evaluation function of the children (not the grandchildren), order

the children from best to worst evaluation function value, from the perspective of the parent node.

2. Prune the (sub-)tree using the resulting order and limiting the depth to the grandchildren(E/F/G/...), using the evaluation function on the grandchildren as their value.

(a) [3 pts] What would the resulting order of the subtrees be?

D then C then B

(b) [4 pts] Prune the tree with this new ordering of the child subtrees. (Grandchildren are still visited left to right.) Write in the box below the nodes that would *NOT* be visited because of pruning. Note: It might be helpful to sketch a new version of tree with the child subtrees reordered.

Nodes: F, G

Iterative deepening would then repeat steps (a) and (b) one level deeper, but skipping any subtrees that were pruned.

(c) [4 pts] When alpha-beta pruning is applied to a minimax tree, it is guaranteed to return the same move that standard minimax without pruning would return. This is because it prunes away branches that cannot possibly influence the final decision.

Is this iterative deepening with pruning method also guaranteed to return the same move as standard minimax on the complete tree? Briefly explain your reasoning.

No

Explain: Việc cắt tỉa (pruning) trong phương pháp này phụ thuộc vào các giá trị đánh giá ở mức nông và thứ tự của các cây con được sắp xếp dựa trên những giá trị đó. Nếu các giá trị hoặc thứ tự này không chính xác, một cây con có thể bị cắt tỉa ở mức nông dù nó chứa nước đi tốt nhất khi tìm sâu hơn. Vì phương pháp này bỏ qua các cây con đã bị cắt tỉa trong các lần lặp sâu hơn, nên nó sẽ không bao giờ xem xét lại chúng — do đó có thể bỏ lỡ một nhánh có thể thay đổi quyết định minimax của toàn bộ cây. Vì vậy, phương pháp này không được đảm bảo sẽ đưa ra cùng một nước đi như minimax tiêu chuẩn (không cắt tỉa).