# Q1. [26 pts] Search and Heuristics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (0  *v* | *,* 4)  = 0 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| (0  *v* | *,* 0)  = 0 |  |  |  |

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* ∈ (*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 *Vmax*
* Maintaining a velocity of 0
* Turning when velocity /= 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.

1. [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.

|  |
| --- |
| 1. Right 2. Faster 3. Slower 4. Left 5. Faster 6. Maintain 7. Maintain 8. Slower 9. Left 10. Faster 11. Slower 12. Right 13. Faster 14. Slower 15. Left |

1. [3 pts] If the grid is M by N and the maximum speed is *Vmax*, what is the size of the state space? You should assume that all configurations are reachable from the start state.

**State Space Size =**

1. [3 pts] A “child” of a state *s* is any other state *s′* reachable 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

|  |
| --- |
| Example State or Explanation: Giả sử agent đang ở một ô bị bao quanh bởi tường ở tất cả các hướng, với vận tốc v = 0.  Theo luật bài toán:   * Khi v = 0, không được maintain. * Chỉ có thể thực hiện *Left* hoặc *Right,* nhưng chỉ hợp lệ nếu sau khi rẽ, không va vào tường khi thực hiện hành động tiếp theo. * Nếu mọi hướng đều bị chặn, thì dù có quay hướng nào, agent cũng không thể thực hiện thêm hành động hợp lệ nào   -> Trạng thái này không có child vì không còn hành động hợp lệ nào có thể thực hiện.  Do đó, có thể tồn tại trạng thái không có child, ví dụ như khi agent ở ô bị tường bao quanh bốn phía và có v = 0. |

1. [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   * Ba hành động đi: Faster, Maintain, Slower (nhưng Slower bị cấm nếu sẽ làm v < 0; Faster bị cấm nếu v = Vmax). * Hai hành động rẽ: Left, Right - chỉ khi v = 0. * Maintain bị cấm khi v = 0   Vì vậy trong mọi trường hợp khả khi tối đa ta có:   * Khi v = 0: có thể có Faster + Left + Right -> 3 hành động (nếu không bị tường cản). * Khi 0 < v < Vmax: có thể có Faster + Maintain + Slower -> 3 hành động.   Không có trạng thái nào cho phép cả 3 hành  động di chuyển và 2 hành động rẽ cùng lúc, vì rẽ chỉ khi v = 0 và khi v = 0 Maintain bị cấm. Do đó tối đa là 3.  Ví dụ trạng thái minh họa (mô tả)  Ví dụ A(v = 0)  (x, y) = (2, 2), orientation = North, v = 0.  Giả sử 3 ô phía trước và hai hướng rẽ đều không bị tường chắn — thì các hành động có sẵn là:   * Faster (v:0 -> 1, đi 1 ô) * Left (quay trái, vẫn v = 0) * Right (quay phải, vẫn v = 0)   Ví dụ B (0 < v < Vmax):  (x, y) = (2, 2), orientation = East, v = 1 và ô phía trước đủ chỗ.  Các hành động có sẵn là:   * Faster (v:1->2, đi 2 ô nếu không va chạm) * Maintain (v:1, đi 1 ô) * Slower (v:1->0, đi 0 ô)   Cả hai ví dụ đều có 3 hành động hợp lệ -> minh chứng cho branching tối đa = 3. |
| Maximum Branching Example State and Available Actions:  Ví dụ trạng thái:  (x, y, orientation, v) = (2, 2, North, 0)   * Agent đang ở giữa bản đồ (2, 2), hướng Bắc, vận tốc 0. * Giả sử hướng trước mặt, trái, phải đều trống (không có đường). * Khi v = 0, chỉ có thể rẽ hoặc tăng tốc (Maintain bị cấm).   Các hành động hợp lệ (Available actions)   * Faster -> tăng tốc lên 1 và di chuyển 1 ô về phía Bắc * Left -> quay sang trái 90 độ, hướng đông, vẫn v = 0 |

1. [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:  Giả sử  Trạng thái ban đầu:   * Vị trí: (0,0) * Hướng East * Vận tốc v = 3   Mục tiêu:   * Vị trí (3, 0): cùng hàng, cách 3 ô về phía Đông) * Hướng: bất kì * Vận tốc v = 0   Giá trị Heuristic (Manhattan):  Một chuỗi hành động hợp lệ dẫn tới mục tiêu (ít hành động hơn 3) – minimize số hành động:  Ta có thể đạt khoảng cách 3 ô trong 2 hành động như sau:   1. Slower (từ v = 3 -> v0 = 2), agent di chuyển 2 ô về Đông -> agent ở (2, 0), vận tốc v = 2 2. Slower (từ v = 2 -> v0 = 1), agent di chuyển 1 ô về Đông -> agent ở (3, 0), vận tốc v = 1 3. Slower (từ v = 1 -> v0 = 0), agent di chuyển 0 ô -> dừng tại (3, 0), vận tốc v = 0   Tổng số hành động ở chuỗi trên là 3 – bằng với Manhatance trong ví dụ cụ thể này. Tuy nhiên, nếu ta chọn ví dụ khác (ví dụ khoảng cách lớn hơn và vận tốc ban đầu cao hơn, hoặc tổ hợp tăng/giữ vận tốc hợp lý), ta có thể giảm số hành động với Manhattan vì mỗi hành động có thể di chuyển nhiều ô. Do đó, Manhatta không đảm bảo luôn <= chi phí thực tối thiểu theo số hành động. |

1. [4 pts] Is the following heuristic admissible? *Manhattan distance / Vmax*.

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.

⃝ Yes ⃝ ~~No~~

|  |
| --- |
| Example State, Heuristic value, Actual cost: |

1. [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.

⃝ Yes ⃝ No

1. [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 ⃝ No

1. [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.

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.

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

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

# 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 *nc* be a minimum-cost goal node, and let *ns* 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*) ≥ 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.

1. [5 pts] Give an inequality in terms of the functions *g*, *h*, and *d*, as well as the nodes *nc* and *ns* defined above to describe the nodes *n* that are explored in a **breadth-first search** before terminating.

**Inequality: All** *n***, such that:**

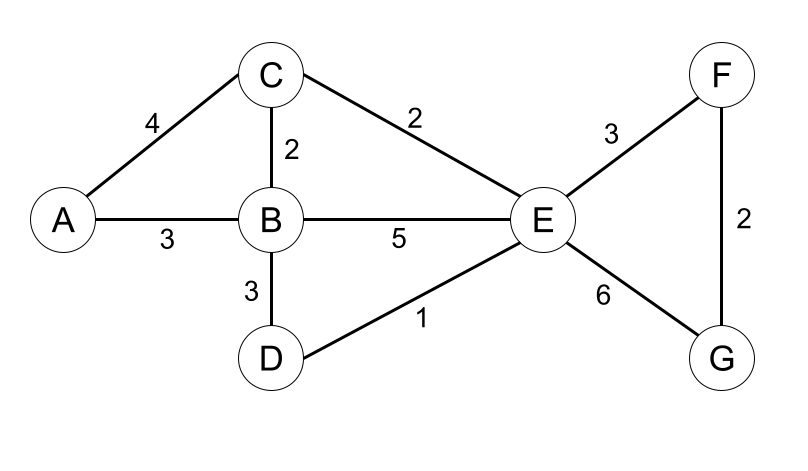
1. [5 pts] Give an inequality in terms of the functions *g*, *h*, and *d*, as well as the nodes *nc* and *ns* defined above to describe the nodes *n* that are explored in a **uniform cost search** before terminating.

**Inequality: All** *n***, such that:**

1. [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 *nc* and *ns* defined above to describe the nodes *n* that are explored in an **A\* search** before terminating.

**Inequality: All** *n***, such that:**

# 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.

1. [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) | (iii) | (iv) |
| Greedy with heuristic *h*1 | (v) | (vi) | (vii) | (viii) |
| Greedy with heuristic *h*2 | (ix) | (x) | (xi) | (xii) |
| A\* with heuristic *h*1 | (xiii) | (xiv) | (xv) | (xvi) |
| A\* with heuristic *h*2 | (xvii) | (xviii) | (xix) | (xx) |

1. [2 pts] What is the cost of the optimal path for uniform cost search from A to G?

**Answer:**

1. [4 pts] Is *h*1 admissible? Is it consistent? Admissible: ⃝ Yes ⃝ No Consistent: ⃝ Yes ⃝ No
2. [4 pts] Is *h*2 admissible? Is it consistent? Admissible: ⃝ Yes ⃝ No Consistent: ⃝ Yes ⃝ No

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

1. [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 *ϵ*, with *ϵ >* 0. Assume any heuristics used are consistent.

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

⃝ **True** ⃝ **False**

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

⃝ **True** ⃝ **False**

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

⃝ **True** ⃝ **False**

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

⃝ **True** ⃝ **False**

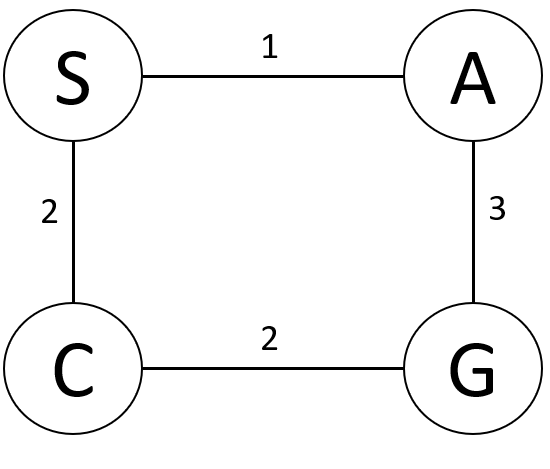
Iterative deepening graph-search has the time complexity of BFS and the space complexity of DFS.

⃝ **True** ⃝ **False**

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** ⃝ **False**

1. [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.



|  |  |
| --- | --- |
| **Heuristic Function** | |
| **State** | *h*(*s*) |
| *S* |  |
| *A* |  |
| *C* |  |
| *G* | 0 |

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

**A**

**B**

**C**

**D**

**E**

**F**

**G**

**H**

**I**

**J**

**K**

**L M**

|  |  |
| --- | --- |
| **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.
   1. [3 pts] What would the resulting order of the subtrees be?

⃝ *B* then *C* then *D* ⃝ *C* then *D* then *B* ⃝ *D* then *C* then *B*

* 1. [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:**

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

* 1. [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.

⃝ Yes ⃝ No

**Explain:**