Q1

(a)

Right v = 0 (0,0)

Faster v = 1 (1,0)

Slower v = 0 (1,0)

Left v = 0 (1,0)

Faster v = 1 (1,1)

Maintain v = 1 (1,2)

Maintain v = 1 (1,3)

Slower v = 0 (1,3)

Left v = 0 (1,3)

Faster v = 1 (0,3)

Slower v = 0 (0,3)

Right v = 0 (0,3)

Faster v = 1 (0,4)

Slower v = 0 (0,4)

Left v = 0 (0,4)

(b)

State Space Size:

Mỗi trạng thái agent sẽ được xác định bởi 3 yếu tố:

1. Kích thước mê cung: MxN
2. Hướng: Có 4 hướng (Đông, Tây, Nam, Bắc)
3. Vận tốc: Co thể nhận các giá trị v = 0,1,2,…,Vmax=> Tổng cộng Vmax+1 giá trị

Vì thế kích thước không gian trạng thái là : State\_Space\_Size = M\*N\*4\*(Vmax+1)

(c )

Có

Ví dụ khi trạng thái hiện tại là trạng thái đích thì nó sẽ không có trạng thái con nào nữa

(d)

Maximum Branching Factor:

Hệ số phân nhánh tối đa là ba

Left/Right chỉ được thực hiện khi v = 0

Faster/Maintain/Slower chỉ khi các hành động đó hợp lệ về vận tốc tức là không va chạm tường (0<v<Vmax)

Không có trạng thái nào cho 4 đến 5 hoạt động thực hiện đồng thời nên hệ số phân nhánh tối đa là 3

Maximum Branching Example State and Available Actions:

Ví dụ v=0:

Trạng thái: (x,y)=(3,3),  hướng=North,  v=0

Giả sử ô (4,3) (phía trước) trống và Vmax≥1..

**Các actions khả dĩ:** {**left, right, faster**} → branching = 3.

(e)

Example State, Heuristic Value, Actual Cost:

Không

Heuristic Manhattan chỉ đo khoảng cách theo số ô hàng + cột, không xét hướng (orientation) và vận tốc (velocity).

Trong khi đó, agent chỉ có thể quay khi v=0, và mỗi bước di chuyển được nhiều ô tùy theo vận tốc, nên chi phí thực tế (số hành động) có thể ít hơn hoặc nhiều hơn so với khoảng cách Manhattan.

Có thể tồn tại trạng thái mà Manhattan ước lượng lớn hơn chi phí thật, tức là overestimate ⇒ Không admissible.

Ví dụ  
Trạng thái: (x,y,hướng,v) = (0,0,Đông,2)

Goal: (0,3,Đông,1)

Heuristic Mahattan: h(n) = |3-0|+|0-0| = 3

Thực tế

1. Duy trì – (0,0) đến (0,2) v = 2
2. Chậm lại – (0,2) đến (0,3) v =2-1 =1

* Số hành động thực tế: 2
* h(n) > h\*(n) nên Mahattan distance not admissible

(f)

Có

Vì với dụ ở câu e đã nêu

Khi Vmax = 2

Thì h(n)/Vmax = 3/2 = 1.5

h(n)/Vmax <= h\*(n): 1.5<=3

(g)

A\* tree search luôn hoàn chỉnh trên đồ thị hữu hạn nếu chi phí từng bước dương (c>0), bất kể heuristic là admissible hay không.

Lý do: A\* tree search luôn mở rộng các node theo f(n) = g(n)+h(n). Nếu đồ thị hữu hạn, node cuối cùng sẽ được mở rộng, nên chắc chắn sẽ tìm ra goal nếu có đường đi.

Heuristic inadmissible không đảm bảo tìm ra đường đi ngắn nhất, nhưng không ảnh hưởng tới việc tìm ra bất kỳ giải pháp nào.

Nên đáp án sẽ là: No

Dùng heuristic inadmissible trong A\* tree search không làm mất completeness nếu đồ thị hữu hạn và chi phí dương.

(h)

1. Optimality (tính tối ưu):

* Một thuật toán tìm kiếm optimal là thuật toán luôn tìm ra đường đi ngắn nhất (chi phí nhỏ nhất) đến goal.

1. Heuristic inadmissible:

* Nếu heuristic vượt quá chi phí thực tế, A\* có thể ưu tiên mở rộng node sai, dẫn đến tìm ra giải pháp không ngắn nhất.

1. Kết luận:

* Dùng heuristic inadmissible có thể làm mất optimality.

Đáp án: Yes

(i)

1. **“An inadmissible heuristic may be easier to compute, leading to a faster state heuristic computation time.”**

* Đúng. Nếu heuristic inadmissible đơn giản hơn, **tính toán nhanh hơn**, nên A\* chạy nhanh hơn, dù mất optimality.

1. **“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.”**

* Đúng. Một heuristic inadmissible nhưng **gần với chi phí thực tế** sẽ giúp A\* **ưu tiên node đúng hơn**, mở rộng ít node hơn → giảm thời gian tìm kiếm.

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

* Sai. Inadmissible heuristic **không đảm bảo optimality**, dù chi phí ≥ 0.

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

* Sai. Heuristic inadmissible **không dùng để “chặn” vùng đồ thị”**, và việc này không phải lý do chính để chọn inadmissible.

Đáp án: Lý do 1,2

Q2

1. Vì breadth-first search (BFS) mở rộng các nút theo thứ tự tăng dần của **độ sâu**, BFS sẽ mở tất cả các nút có độ sâu nhỏ hơn độ sâu của nút mục tiêu nông nhất nsn\_sns​, và sau đó mở (một phần hoặc tất cả) các nút ở cùng độ sâu d(ns​) cho tới khi gặp mục tiêu. Do đề bài nói “không phải lo ties”, ta có thể mô tả tập các nút đã bị khám phá bằng bất đẳng thức đơn giản:

{n∣d(n)≤d(ns​)}.

1. Uniform-Cost Search mở rộng các nút theo thứ tự ggg tăng dần và dừng khi lần đầu tiên nó mở một nút mục tiêu có chi phí nhỏ nhất g(nc). Vì vậy tập các nút bị khám phá là

{n ∣ g(n) ≤ g(nc​)}.

Giải thích ngắn: UCS sẽ mở tất cả nút có chi phí tích lũy nhỏ hơn chi phí mục tiêu tối thiểu, và (bỏ qua xử lý tie như đề bài cho phép) cũng sẽ mở các nút có cùng chi phí g(n)=g(nc​) đủ để đạt mục tiêu.

1. Với f(n)=g(n)+h(n) và vì h consistent nên A\* mở rộng các nút theo thứ tự tăng của f và dừng khi mở mục tiêu tối ưu nc ​ (với f(nc)=g(nc)+h(nc)). Do đó tập các nút bị khám phá là

{ n ∣ f(n) ≤ f(nc​) }

tức là

{ n ∣ g(n) + h(n) ≤ g(nc​) }

Q3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | A-C-E-G | A-C-E-F-G | A-B-D-E-F-G | Other |
| UCS |  | X |  |  |
| Greedy with heuristic h1 |  |  |  | A–B–D–E–G |
| Greedy with heuristic h2 | X |  |  |  |
| A\* with heuristic h1 |  |  | X |  |
| A\* with heuristic h2 |  | X |  |  |

1. Đường đi tối ưu (do Uniform Cost Search luôn tìm được đường có chi phí nhỏ nhất) là A-C-E-F-G có chi phí là 11
2. [4 điểm] Heuristic h1​ có **admissible** không? Có **consistent** không? (Dịch: h1có bao giờ ước lượng vượt quá chi phí thực còn lại tới đích không? Và có thỏa tính đơn điệu: h(n)≤c(n,a,n′)+h(n′) cho mọi cạnh không?)

**Admissible? — Không.**  
Vì h1(A)=12 trong khi chi phí thực tối thiểu từ A tới G là 11. h1(A) vượt quá chi phí thực => h1​ không thỏa tính admissible. (Ví dụ khác: h1(C)=9 trong khi chi phí thực từ C tới G là 7.)

**Consistent? — Không.**  
Ta chỉ cần một phản ví dụ: cạnh A−B có chi phí c(A,B)=3  
h1​(A)=12 nhưng c(A,B)+h1(B)=3+6=9. Vì 12>9, bất đẳng thức đơn điệu bị vi phạm, nên h1​ không consistent.

1. **Admissible? – Có**

Admissible: với mỗi nút nnn giá trị h2(n) không vượt quá chi phí thực tối thiểu từ n đến G. Cụ thể các chi phí thật tới G là  
d(A)=11, d(B)=9, d(C)=7, d(D)=6, d(E)=5, d(F)=2, d(G)=0 so sánh với h2: A=11, B=7, C=6, D=4, E=5, F=1, G=0 → tất cả đều h2(n)≤d(n). Vậy h2 là admissible.

**Consistent? – Không**

Consistent: kiểm tra tính đơn điệu h(n)≤c(n,n′)+h(n′) trên các cạnh có chi phí: tồn tại vi phạm, ví dụ  
h2(A)=11h\_2(A)=11h2​(A)=11 nhưng c(A,B)+h2(B)=3+7=10<11, nên h2(A)>c(A,B)+h2(B). Tương tự h2(E)=5>c(E,F)+h2(F)=3+1=4. Do đó h2h\_2h2​ **không consistent**.

Q4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Câu hỏi | Đáp án | Giải thích |
| 1 | Depth-first **tree-search** complete? | **False** | DFS có thể đi mãi trong một nhánh sâu vô tận (hoặc lặp lại) → không đảm bảo hoàn chỉnh. |
| 2 | Breadth-first **tree-search** complete? | **True** | BFS mở theo độ sâu tăng dần, đồ thị hữu hạn → chắc chắn tìm được goal nếu tồn tại. |
| 3 | Iterative deepening **tree-search** complete? | **True** | Mỗi lần tăng giới hạn độ sâu, sẽ tìm thấy goal nếu tồn tại (vì đồ thị hữu hạn). |
| 4 | For acyclic graphs, **graph-search frontier > tree-search frontier** | **False** | Nếu không có chu trình, graph-search và tree-search duyệt các nút giống nhau → frontier không lớn hơn. |
| 5 | Iterative deepening **graph-search** có time như BFS và space như DFS | **True** | Đây là tính chất chính: IDDFS = BFS về thời gian, DFS về bộ nhớ. |
| 6 | (min(h1,h2)) của 2 heuristic consistent cũng consistent | **True** | Lấy min của hai heuristic consistent vẫn giữ bất đẳng thức (h(n) \le c(n,n') + h(n')). |

|  |  |
| --- | --- |
| Heuristic Function | |
| State | h(s) |
| S | 4 |
| A | 2 |
| C | 2 |
| G | 0 |

Q5

1. D then C then B vì f(B) = 4 < f(C) = 14 < f(D) = 19
2. Các đỉnh con của DCB được ghé thăm từ trái sang phải

D: K=23, L=20, M=26 ⇒ D trả min=20 → cập nhật α = 20 tại A.

C: H=17, I=16, J=15

Thăm H = 17 => tại nút MIN của C: best = 17. Vì best (17) ≤ α (20) → prune các cháu còn lại I và J.

Kết quả: I, J không được thăm. C trả 17 (α còn là 20).

B: E=5, F=7, G=8

Thăm E = 5 => best = 5. Vì best (5) ≤ α (20) → prune F và G.

Kết quả: F, G không được thăm.

Vậy các nút KHÔNG được thăm vì pruning là: I, J, F, G.

Alpha–beta pruning (bình thường) chỉ cắt đi những nhánh chắc chắn không ảnh hưởng đến kết quả cuối cùng, nên kết quả (nước đi) luôn giống hệt với Minimax đầy đủ.

Iterative deepening + pruning thì khác:

* + - Ở mỗi vòng lặp, nó chỉ tìm đến một độ sâu giới hạn.
    - Việc sắp xếp thứ tự con (ordering) cho vòng sau dựa trên giá trị ước lượng từ vòng trước.
    - Vì giá trị ở vòng nông chưa chính xác, nên thứ tự con có thể sai, dẫn đến pruning nhầm ở vòng sâu.
    - Khi đó, một số nhánh có thể bị cắt bỏ sớm, dù chúng có thể thay đổi kết quả cuối cùng nếu được thăm.

Iterative deepening with pruning không được đảm bảo trả về cùng nước đi với standard minimax trên cây đầy đủ, vì việc ước lượng và sắp thứ tự dựa trên độ sâu giới hạn có thể dẫn đến pruning sai lệch.

Đáp án: No