1. Search Algorithms (20 pts)

A diagram of a network

AI-generated content may be incorrect.

Using each of the following graph search algorithms presented in lecture, write out the order in which nodes are added to the explored set, with start state *S* and goal state *G*. Break ties in alphabetical order by *the last state in the path*. Additionally, what is the path returned by each algorithm? What is the total cost of each path?

(a) Breadth-first

**Ý tưởng:** frontier là queue (FIFO). Mỗi lần pop path từ frontier, node cuối cùng của path được expand (nếu chưa trong explored), và khi pop thì node đó được **thêm vào explored**. Khi tìm thấy goal (khi pop path có last node = G), dừng:

Chi tiết bước chạy

1. Frontier = [ (S) ]. Explored = [].
2. Pop (S) → last node = S. Add S vào Explored: S.

* Expand S: successors (A, B) (theo alphabetical). Thêm paths: (S→A), (S→B).
* Frontier = [ (S→A), (S→B) ].

1. Pop (S→A) → last = A. Add A vào Explored: S, A.

* Expand A: successor C. Thêm (S→A→C).
* Frontier = [ (S→B), (S→A→C) ].

1. Pop (S→B) → last = B. Add B vào Explored: S, A, B.

* Expand B: successor D. Thêm (S→B→D).
* Frontier = [ (S→A→C), (S→B→D) ].

1. Pop (S→A→C) → last = C. Add C into Explored: S, A, B, C.

* Expand C: successor G. Thêm (S→A→C→G).
* Frontier = [ (S→B→D), (S→A→C→G) ].

1. Pop (S→B→D) → last = D. Add D into Explored: S, A, B, C, D.

* Expand D: successor G. Thêm (S→B→D→G).
* Frontier = [ (S→A→C→G), (S→B→D→G) ].

1. Pop (S→A→C→G) → last = G. Add G into Explored: S, A, B, C, D, G.

* Goal found → dừng.

(b) Depth-first

**Ý tưởng:** frontier là stack (LIFO). Khi expand một node, ta **push** successors theo thứ tự **alphabetical**; vì stak LIFO, successor **được push sau cùng** sẽ là node được pop tiếp. (Dùng quy ước phổ biến: push successors in alphabetical order.)

Chi tiết bước chạy

1. Frontier = [ (S) ] (stack). Explored = [].
2. Pop (S) → last = S. Add S vào Explored: S.

* Expand S: successors A, B (push A then B).
* Frontier (top at right) = [ (S→A), (S→B) ] (S→B là top).

1. Pop (S→B) → last = B. Add B: Explored = S, B

* Expand B: successor D (push).
* Frontier = [ (S→A), (S→B→D) ] (top = S→B→D).

1. Pop (S→B→D) → last = D. Add D: Explored = S, B, D.

* Expand D: successor G (push).
* Frontier = [ (S→A), (S→B→D→G) ]

1. Pop (S→B→D→G) → last = G. Add G: Explored = S, B, D, G.

* Goal found → dừng.

(c) Iterative deepening

**Ý tưởng:** Chạy DFS có giới hạn độ sâu limit = 0,1,2,... lần lượt. Khi một iteration với limit = d tìm ra goal, dừng và trả path. Thông thường IDDFS được mô tả như *tree-search* và không lưu explored across iterations (tức mỗi iteration reset frontier/explored). Ở đây ta trình bày theo cách chuẩn (mỗi iteration reset).

Chạy các iteration

**limit = 0**

* DFS-limited(0) chỉ khám S (depth 0). Explored (iteration 0): S. Không thấy G.

**limit = 1**

* DFS-limited(1): khám theo DFS nhưng không đi quá độ sâu 1.
* Start S → expand successors A, B (sẽ khám A then B). Explored (iteration 1) in order: S, A, B. Không thấy G.

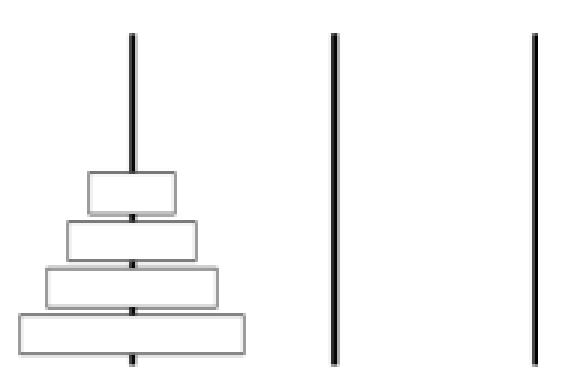
**limit = 2**

* DFS-limited(2): có thể khám nodes ở depth 2 (C, D). Khám theo order: S, A, C, B, D (tuỳ implement). Không thấy G.

**limit = 3**

* DFS-limited(3): có thể khám tới depth 3; G nằm ở depth 3 qua cả 2 đường: S→A→C→G và S→B→D→G.
* Với successor order alphabetical, DFS-limited(3) sẽ khám S → A → C → G trước (vì S→A branch được khám trước S→B branch).
* Khi pop node G, dừng.

Tower of Hanoi (20 pts)



The Tower of Hanoi is a canonical puzzle studying problem solving and formulation. The puzzle starts with *n* disks of different sizes stacked in order of size (see picture above) on a peg, along with two empty pegs. We can move disks freely between the pegs, but larger disks cannot be stacked on top of smaller ones. The goal is to move all disks to the third peg.

We will attempt to formulate Tower of Hanoi as a search problem.

(a) How could we represent this puzzle as a problem, ie. what would the states be?

Một vài cách phổ biến (đều hợp lệ):

* 1. Vector/array chỉ định cọc của mỗi đĩa:
* Mỗi trạng thái là một mảng pos[1..n] sao cho pos[i] ∈ {A,B,C} cho đĩa kích thước i (ví dụ i = 1 là nhỏ nhất, i = n là lớn nhất).
* Diễn giải: tất cả đĩa có cùng giá trị pos nằm trên cùng một cọc; thứ tự trên cọc được suy ra theo kích thước (đĩa lớn hơn nằm dưới đĩa nhỏ hơn).
* Ví dụ (n=3): bắt đầu pos = [A, A, A] (đĩa1 nhỏ nhất tại A, đĩa3 lớn nhất tại A); trạng thái nơi đĩa lớn ở C, đĩa nhỏ ở A: pos = [A, C, C] (đĩa1 ở A, đĩa2 và đĩa3 ở C).
  1. Ba ngăn (lists/stacks) chứa thứ tự đĩa trên mỗi cọc
* Trạng thái = (stackA, stackB, stackC) trong đó mỗi stack là danh sách đĩa từ đáy → đỉnh (hoặc đỉnh → đáy).
* Ví dụ (n=3) bắt đầu: ([3,2,1], [], []).
  1. **Chuỗi/tuple** kết hợp (ví dụ "AA C"), nhưng ý tưởng tương tự.

For the following questions, assume that you have used your state representation in your answer above.

(b) What is the size of the state space in terms of *n*?

* Mỗi đĩa có thể nằm trên 3 cọc độc lập → tổng số cấu hình có thể là .
* Do cách biểu diễn pos[1..n] (ghi vị trí từng đĩa), mỗi ánh xạ đĩa→cọc tương ứng với một trạng thái hợp lệ (thứ tự chồng trên mỗi cọc được suy ra theo kích thước). Vì thế **kích thước không gian trạng thái =** .

(c) What is the starting state?

* Mô tả chung: Tất cả n đĩa trên cọc bắt đầu (ví dụ cọc A), xếp theo kích thước giảm từ đáy lên (đĩa lớn nhất ở đáy).
* Nếu dùng pos thì: pos = [A, A, A, ..., A] (n lần A).
* Nếu dùng stack thì: stackA = [n, n-1, ..., 1], stackB = [], stackC = [].

Ví dụ n=3: Start = ([3,2,1], [], []) hoặc pos = [A,A,A].

(d) From some given state, what legal actions are there?

Mô tả hành động:

* Một hành động là chuyển **top (đĩa nhỏ nhất nằm trên cùng)** của một cọc X sang cọc Y (X ≠ Y), với điều kiện: cọc X không rỗng và (cọc Y rỗng **hoặc** đĩa đỉnh của Y lớn hơn đĩa được di chuyển).
* Kí hiệu hành động: move(X → Y).

Cách tính các action từ trạng thái:

* ấy 3 cọc; với mỗi cọc s nếu s không rỗng, xét hai cọc khác t1,t2. Nếu đĩa trên đỉnh s nhỏ hơn đĩa đỉnh t (hoặc t rỗng), thì move(s→t) là hợp lệ.
* Do đó số hành động hợp lệ phụ thuộc vào trạng thái.

Giới hạn trên / nhận xét về branching factor:

* Mỗi cọc không rỗng cung cấp tối đa 2 hướng di chuyển → tổng giới hạn trên là hành động. (Đây là **giới hạn lý thuyết**; trong thực tế nhiều di chuyển bị cấm vì luật kích thước, nên số hành động hợp lệ thường ≤ 3.)
* Trong thuật toán tối ưu điển hình, số hành động hữu ích khá nhỏ; nhưng để an toàn, ta có thể nói **số hành động hợp lệ từ một trạng thái nằm trong khoảng** tùy cấu hình (thông thường ~= 2 hoặc 3).

Độ phức tạp sinh successor:

* Tạo danh sách các successor dễ thực hiện trong O(1) kiểm tra cho mỗi cặp (s,t) với 3 cọc → tổng O(1) (thực tế O(1) hoặc O(3) tùy cài đặt).

(e) What is the goal test? Remember that this determines whether a given state is a goal state,

* Mục tiêu tiêu chuẩn: **tất cả n đĩa nằm trên cọc đích** (ví dụ cọc C) theo thứ tự kích thước đúng (đĩa lớn ở đáy → đĩa nhỏ trên đỉnh).
* Nếu dùng pos thì goal test: pos[i] == C cho mọi i = 1..n.
* Nếu dùng stack thì goal test: stackC = [n, n-1, ..., 1] và stackA = stackB = [].

1. Designing & Understanding Heuristics (20 pts)

Today, we will be taking a closer look at how the performance of *A*∗is affected b y t he h euristics i t u s es. To do this, we’ll be using the graph below. You may have noticed that no heuristic values have been provided (*Recall:* What is *A*∗ without heuristic values?). This is because we’ll be working together to come up with heuristics ourselves!

In groups, design both an admissible heuristic and a consistent heuristic for the graph below by anno tating each node with a heuristic value. (Note: you do NOT need to find a c losed f orm w ay t o r epresent the heuristic function.)

When you have completed your heuristics, work together to answer the questions below.

A diagram of a network

AI-generated content may be incorrect.

(a) Write down the path found by running *A*∗ using your heuristic on the graph above.

Chọn heuristic admissible & consistent: h(S)=8,h(A)=10,h(B)=5,h(C)=6,h(D)=4,h(G)=0

→ Thỏa mãn cho mọi cạnh ⇒ **consistent**.

**Kết quả A\***:

* Explored: S, B, D, G
* Path: **S → B → D → G**
* Cost = **8**

(b)Work with your group to come up with a heuristic that’s admissible but not consistent.

Chọn heuristic admissible nhưng **không consistent**: h(S)=8,h(A)=10,h(B)=5,h(C)=2,h(D)=4,h(G)=0

→ Vi phạm consistency tại cạnh A→C (vì ).

Heuristic vẫn admissible vì chi phí thật đến G.

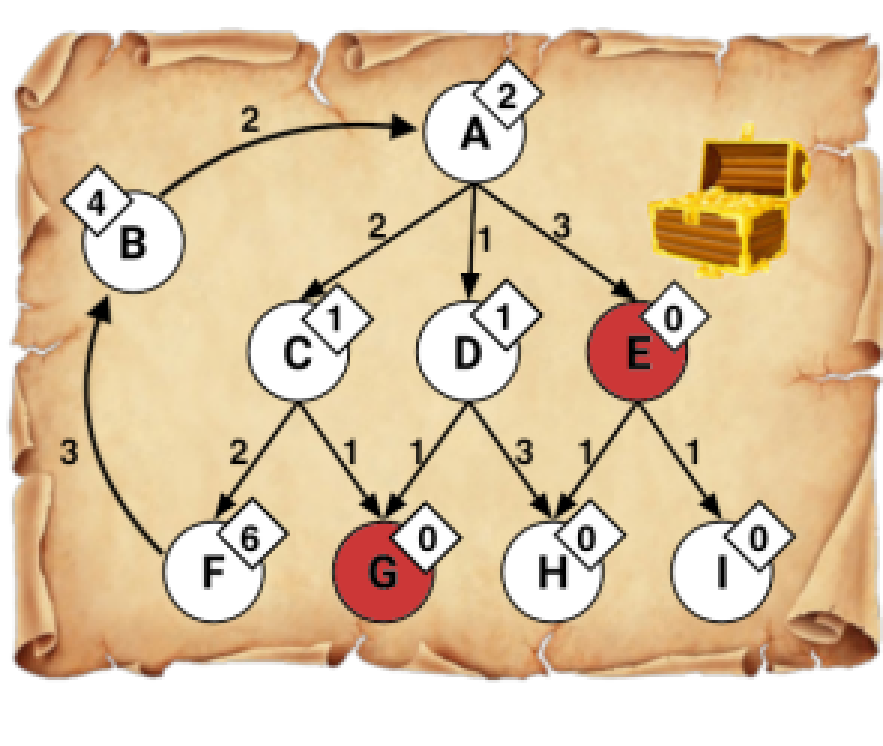
(c) (Bonus) Explain why a consistent heuristic must also be admissible. You may assume that the heuristic value at a goal node is always 0.

Nếu h consistent và , thì theo mọi đường từ n đến G, ⇒ chi phí thật tối ưu.  
→ **Mọi heuristic consistent đều admissible.**

2

1. Treasure Hunting (20 pts)

We are lost at sea, trying to find a h i dden t r easure. W e h a ve a tr easure m a p t h at t e lls u s w h ich p a ths we can take and approximately how far away the treasure is but it’s not very accurate. We do know that the map will never overestimate the distance to the treasure. There is treasure on two different islands (b ut we only need to reach one of them).



*A* is the the island where we are currently and the shaded red states are the locations of the treasure. Arrows encode possible actions from each island, and numbers by the arrows represent action costs. Note that arrows are directed; for example, *A → C* is a valid action, but *C → A* is not. Numbers shown in diamonds are heuristic values that estimate the optimal (minimal) cost to get from that island to any treasure.

Run each of the following search algorithms with graph search and write down the nodes that are added to the explored set during the course of the search, as well as the final path returned and the correspond ing cost of the final path, if applicable. When popping off of the frontier, assume that ties are broken alphabetically.

(a) Depth-First Search Explored set: A,B,C, F, G

Path returned:

A → B → C → F → G

**Cost:**  
A→B(2) + B→C(3) + C→F(2) + F→G(?) -> Dừng ở G (kho báu), tổng **cost = 2 + 3 + 2 = 7**

(b) Breadth-First Search

Explored set:

A, B, C, D, E

Path returned:

A → E (gặp kho báu đầu tiên ở tầng 1)

**Cost:** 3

(c) Uniform-Cost Search Explored set: A, D,E

Path returned and cost: A → D → G (goal gần nhất theo cost)

**Cost:** 1 (A→D) + 1 (D→G) = **2**

(d) Greedy Search Explored set: A, D, G

Path returned and cost:

A → D → G

**Cost:** 2 (giống UCS, vì D có h=1 nhỏ nhất)

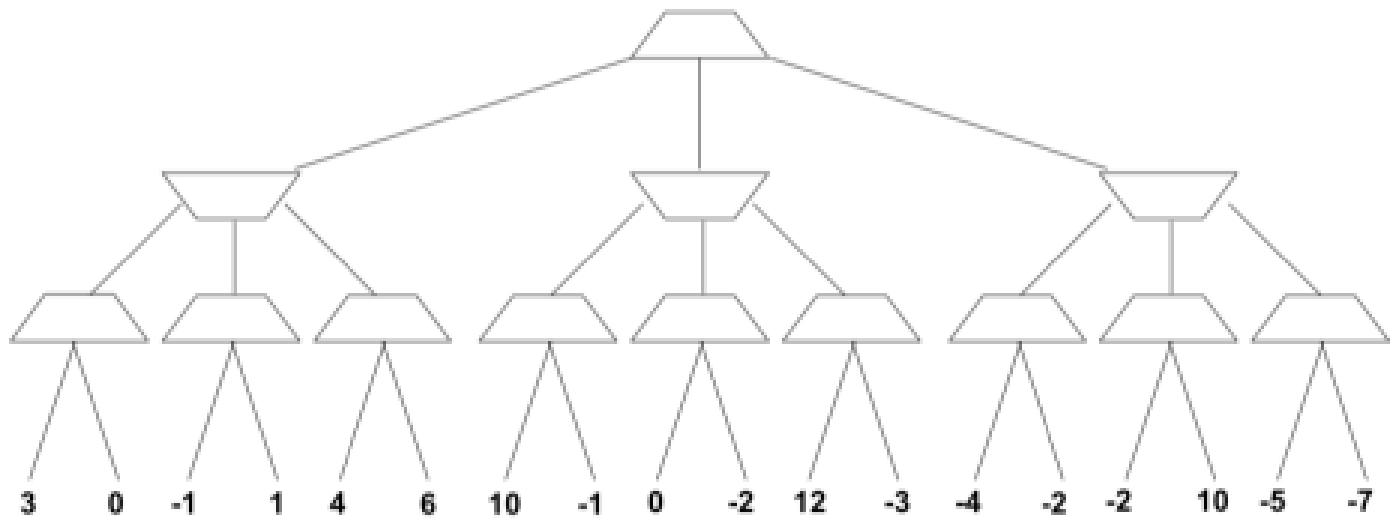
(e) A∗ Search Explored set: A, D, G

Path returned and cost: A → D → G

**Cost:** 1 + 1 = **2**

1. Adversarial Search (10pts)

Consider the following game tree, where the root node is a maximizer. Using alpha beta pruning and visiting successors from left to right, record the values of alpha and beta at each node. Furthermore, write the value being returned at each node inside the trapezoid. Put an ‘X’ through the edges that are pruned off.



Lá theo thứ tự trái→phải: [3, 0] , [-1, 1] , [4, 6] , [10, -1] , [0, -2] , [12, -3] , [-4, -2] , [-2, 10] , [-5, -7]

**Kết quả chính:**

* **Root (Max)** → **1**
* **Min1** → 1 (X13 bị cắt lá 6)
* **Min2** → 1 (X23 bị cắt toàn bộ – lá 12, –3)
* **Min3** → 1 (X32, X33 bị cắt toàn bộ)

**Các giá trị con (Max nodes):**  
X11 = 3, X12 = 1, X13 = 4, X21 = 10, X22 = 1, X23 (pruned), X31 = 1

**Edges bị prune:**

* Lá **6** (X13)
* Cả **X23** (12, –3)
* Cả **X32** (–2, 10) và **X33** (–5, –7)

**Kết quả cuối cùng:**  
**Root value = 1** và các cạnh trên bị đánh **X** (pruned).

1. True/False Section (10pts)

For each of the following questions, answer true or false and provide a brief explanation (or counterexample, if applicable).

(a) Depth-first s earch a lways e xpands a t l east a s m any n odes a s *A* ∗s earch w ith a n a dmissible heuristic.

False. DFS có thể tìm được goal rất nhanh (một nhánh ngắn) và chỉ mở ít node, trong khi A\* với heuristic kém vẫn có thể mở nhiều node trước khi tới goal. (VD: cây nơi goal là lá sâu ở nhánh trái — DFS mở ít node; A\* có thể mở nhiều nhánh trung tâm.

(b) Assume that for a single move, a rook can move any number of squares on a chessboard in a straight line, either vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the smallest number of moves to move the rook from square A to square B.

False. Rook mất ≤2 nước đi giữa hai ô (0 nếu cùng ô, 1 nếu cùng hàng/cột, else 2). Manhattan distance (số ô) có thể lớn hơn số nước đi cần (ví dụ từ (0,0)→(3,3): Manhattan = 6 nhưng rook chỉ cần 2 nước). Vậy Manhattan **không** luôn là lower bound cho số nước đi ⇒ không admissible.

(c) Euclidean distance is an admissible heuristic for Pacman path-planning

True. Euclidean (khoảng thẳng) giữa vị trí Pacman và mục tiêu là một lower bound cho độ dài đường đi thực tế (vì đoạn thẳng là đường ngắn nhất trong mặt phẳng). Do đó nó không vượt quá chi phí ngắn nhất → **admissible**.

(d) The sum of several admissible heuristics is still an admissible

False. Tổng của nhiều heuristic admissible có thể overestimate. (VD: h1(n) ≤ h\*(n) và h2(n) ≤ h\*(n) nhưng h1(n)+h2(n) có thể > h\*(n).)

(e) Admissibility of a heuristic for *A*∗search implies consistency as well.

False. Admissible (h(n) ≤ h\*(n)) không ngụ ý consistency. Có heuristics thỏa admissible nhưng vi phạm bất đẳng thức tam giác (ví dụ đã dùng ở phần trước: giảm h(C) đủ nhiều để làm A→C vi phạm

(f) *A*∗ with graph search is always optimal with an admissible heuristic.

False. A\* với **graph-search** chỉ đảm bảo tối ưu nếu heuristic là **consistent** (hoặc nếu triển khai cho phép re-open khi tìm đường rẻ hơn). Với heuristic chỉ admissible nhưng không consistent và không re-open, A\* graph-search có thể trả kết quả không tối ưu

For (g) and (h), consider an adversarial game tree where the root node is a maximizer, and the minimax value of the game (i.e., the value of the root node after running minimax search on the game tree) is V*M*. Now, also consider an otherwise identical tree where every minimizer node is replaced with a chance node (with an arbitrary but known probability distribution). The expectimax value of the modified game tree is V*E*.

(g) V*M* is guaranteed to be less than or equal to V*E*.

True. Thay mỗi minimizer (min) bằng chance (expectation) không giảm giá trị cho maximizer, vì cho mọi tập giá trị ta có . Do đó minimax value .

(h) Using the optimal minimax policy in the game corresponding to the modified (chance) game tree is guaranteed to result in a payoff of at least V*E*.

False. Chính sách minimax tối ưu cho cây minimax không đảm bảo thu về ≥ trong cây có chance nodes. (Counterexample: hành động mà minimax chọn bảo đảm một giá trị thấp nhưng có hành động khác có kỳ vọng cao; trong cây có xác suất, kỳ vọng của chính sách minimax có thể < .)