Họ tên: Nguyễn Văn Anh Tú

MSSV: 3122410445

**1 Search Algorithm (20pts)**

|  |  |
| --- | --- |
| |  | | --- | | 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?

1. Breadth-first

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hàng đợi | Đỉnh u  (lấy từ hàng đợi) | Hàng đợi  (sau khi lấy từ u ra) | Các đỉnh v kề u mà chưa đánh dấu | Hàng đợi sau khi đẩy những đỉnh v vào |
| {S} | S |  | {A,C,D} | {A,C,D} |
| {A,C,D} | A | {C,D} |  | {C,D} |
| {C,D} | C | {D} | {F} | {D,F} |
| {D,F} | D | {F} | {B,G} | {F,B,G} |
| {F,B,G} | F | {B,G} | {E} | {B,G,E} |
| {B,G,E} | B | {G,E} |  | {G,E} |
| {G,E} | G | {E} |  | {E} |

Đường đi: S => D => G. Tổng chi phí: 10

1. Depth-first

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bước | Stack(trước) | Pop | Stack(sau) | Thêm các đỉnh con | Stack mới |
| 0 | {S} | - | - | - | {S} |
| 1 | {S} | {S} | - | {A,C,D} | {A,C,D} |
| 2 | {A,C,D} | {A} | {C,D} | - | {C,D} |
| 3 | {C,D} | {C} | {D} | {F} | {F,D} |
| 4 | {F,D} | {F} | {D} | {E,G} | {E,G,D} |
| 5 | {E,G,D} | {E} | {G,D} | {B,G} | {B,G,D} |
| 6 | {B,G,D} | {B} | {G,D} | - | {G,D} |
| 7 | {G,D} | {G} | {D} | - | DỪNG |

Thứ tự duyệt: S,A,C,F,E,B,G

Đường đi: S => C => F => E => G

Chi phí: 5

1. Iterative deepening

\*Các bước:

L=0: Mở: {S}

L=1:

- S => A (depth1)

- S => C (depth1)

- S => D (depth1)

L=2:

- S => A (không)

- S => C => F (F là depth2 và F khắc G)

- S => D => G (G ở depth2 và tìm thấy đích là G)

=> Dừng.

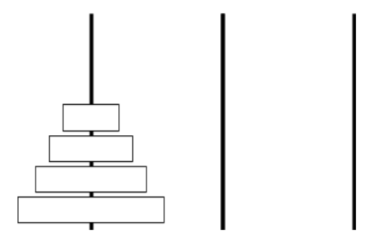
Thứ tự các đỉnh được mở rộng (tổng hợp, theo lần lượt qua các iteration):

* Iter 0: S
* Iter 1: S,A,C,D
* Iter 2: S,A,C,F,D,G (Dừng)

Đường trả về: S => D => G

Tổng chi phí: 10

**2 Tower of Hanoi (20pts)**



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.

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

Có ba cột được ký hiệu là A, B và C lần lượt từ trái sang phải. Ban đầu, cột A chứa n đĩa được xếp chồng lên nhau theo thứ tự kích thước tăng dần từ trên xuống dưới (đĩa nhỏ nhất ở trên cùng, đĩa lớn nhất ở dưới cùng), trong khi hai cột B và C đều trống.

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

1. What is the size of the state space in terms of n?

Kích thước không gian trạng thái là 3n

1. What is the starting state?

Tất cả n đĩa nằm trên cột A, được xếp theo thứ tự từ lớn (dưới cùng) đến bé (trên cùng)

1. From some given state, what legal actions are there?

Hành động hợp lệ là di chuyển đĩa trên cùng của một cột sang một cột khác, với điều kiện: Chỉ di chuyển 1 đĩa 1 lần hoặc đĩa trên cùng của cột đích lớn hơn đĩa đang được di chuyển.

1. What is the goal test? Remember that this determines whether a given state is a goal state, **goalTest(state)**

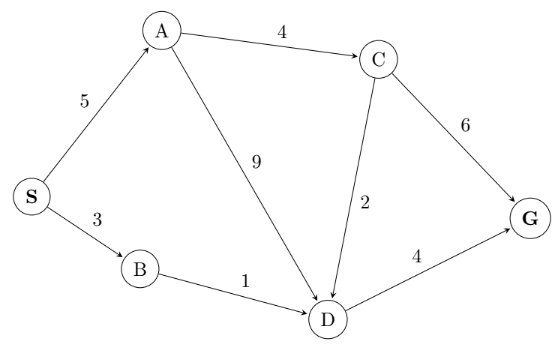
Trạng thái là mục tiêu (goal state) khi tất cả n đĩa đều nằm trên cột C, được xếp theo thứ tự đúng.

**3 Designing & Understanding Heuristics (20pts)**

Today, we will be taking a closer look at how the performance of A\* is affected by the heuristicsituses. 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 heuristic 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 closed form way to present the heuristic function)

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



**Admissible heuristic**

| **Node** | **Cost thật đến G** | **Chọn h (admissible)** |
| --- | --- | --- |
| G | 0 | 0 |
| D | 4 | 4 |
| C | 6 | 6 |
| B | 5 (qua D) | 5 |
| A | 10 (A→C→G) | 10 |
| S | 8 (đường tối ưu) | 8 |

Heuristic này không vượt quá chi phí thật => **Admissible**

1. Write down the path found by running A\* using your heuristic on the graph above

**Heuristic (admissible) đã cho:**  
.

**Cạnh (chi phí):**  
S→A=5, S→B=3, A→C=4, A→D=9, B→D=1, C→D=2, C→G=6, D→G=4.

**Tính và tiến trình A\***

1. Khởi: frontier = {S}, .
2. Mở **S** → sinh A, B:
   * A:
   * B:   
     Frontier = {B(f=8), A(f=15)}.
3. Chọn **B (f=8)**, mở B → sinh D:
   * D:   
     Frontier = {D(f=8), A(f=15)}.
4. Chọn **D (f=8)**, mở D → sinh G:
   * G:   
     Frontier = {G(f=8), A(f=15)}.
5. Chọn **G (f=8)** → đây là goal → dừng.

Kết quả:   
- Đường đi A\* tìm được : S => B => D => G. Tổng chi phí: 8

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

| **Node** | **(h(n))** |
| --- | --- |
| S | 8 |
| A | 0 |
| B | 5 |
| C | 6 |
| D | 4 |
| G | 0 |

So với chi phí thật tối ưu tới G:

* .  
  Mọi ở bảng đều ≤ tương ứng ⇒ **không overestimate** ⇒ **admissible**.

Kiểm tra cạnh có trọng số :

Do đó bất đẳng thức nhất quán bị vi phạm trên cạnh ⇒ heuristic **không consistent**.

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

* Heuristic **consistent** nghĩa là:

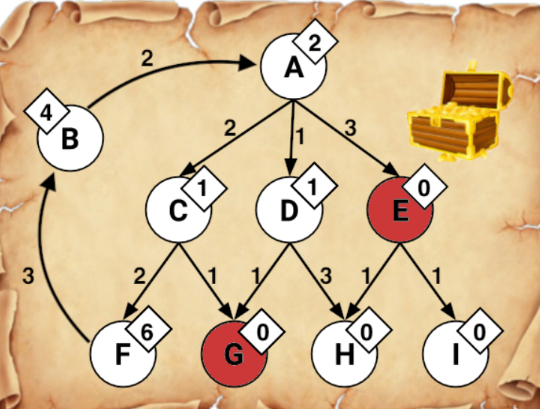
cho mọi cạnh trong đồ thị, và .

* Khi ta áp dụng quy tắc này lặp lại dọc theo **đường đi ngắn nhất từ n đến G**, ta có:
* Vì , nên:

⇒ Do đó, **một heuristic consistent luôn luôn admissible**, vì nó **không bao giờ đánh giá vượt quá chi phí thật đến goal**.

**4 Treasure hunting(20pts)**

We are lost at sea, trying to find a hidden treasure. We have a treasure map that tell us which paths 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 is lands(but we only need to reach one of them)



A is 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 corresponding cost of the final path, if applicable. When popping off of the frontier, assume that ties are broken alphabetically.

* + 1. Depth-First Search

Explored set: **A,C,F,B,G**

Path returned: **A,C,G**

* + 1. Breadth-First Search

Explored set: **A,C,D,E**

Path returned: **A,E**

* + 1. Uniform-Cost Search

Explored set: **A,D,C,G**

Path returned: **A,D,G**

* + 1. Greedy Search

Explored set: **A, E**

Path returned: **A, E**

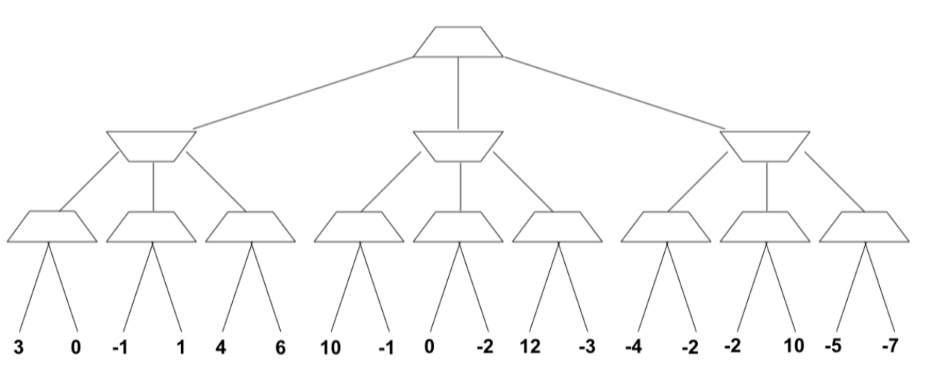
* + 1. A\* search

Explored set: **A,D,G**

Path returned: **A,D,G**

**5 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.



**X**

**X**

**6 True/False section(10pts)**

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

1. Depth-first search always expands at least as many nodes as A\* search with admissible heuristic **=> False**

Trong cây tìm kiếm có một nghiệm rất gần gốc, A\* với heuristic tốt có thể tìm thấy nghiệm sau khi mở vài node,  
nhưng **DFS** có thể đi rất sâu vào nhánh sai trước khi quay lại.  
→ A\* mở ít node hơn DFS.

1. 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 ở **A1**, cần đến **A8**.

* Số move thật = 1 (đi dọc cột A).
* Manhattan distance = 7.  
  → Heuristic **quá lớn**, không admissible.

1. Euclidean distance is an admissible heuristic for Pacman path-planning problems. **=> True**

Pacman di chuyển trên lưới, mỗi ô có chi phí ≥ 1.  
→ Độ dài đường đi thật luôn ≥ khoảng cách Euclidean.  
→ Heuristic này **không bao giờ vượt quá chi phí thật**.

1. The sum of several admissible heuristics is still an admissible heuristic. **=> False**

h₁ = 5, h₂ = 5, chi phí thật = 7  
→ h₁ và h₂ đều ≤ 7 (admissible),  
nhưng h₁ + h₂ = 10 > 7 → không admissible.

1. Admissibility of a heuristic for A\* search implies consistency as well. **=> False**

Giả sử ta có đồ thị có 2 nodes: A => G (chi phí thật là 2)

h(A)=3, h(G)=0

* h(A) = 3 mà chi phí thật là 2 nên không admissible

1. A\* with graph search is always optimal with an admissible heuristic. **=> False**

Nếu dùng **graph search**, đôi khi có thể không tối ưu nếu heuristic **không consistent**.  
Ví dụ:

* Node A → B → Goal, chi phí thật = 2+2=4.
* h(A)=3, h(B)=0.  
  → A\* có thể chọn sai đường do heuristic không đảm bảo tính tăng dần.

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 VM. 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 VE.

1. VM is guaranteed to be less than or equal to VE. **=> True**

Giả sử minimax (người chơi đối kháng) có kết quả:

* Nếu Max chọn Left, Min trả về 2.
* Nếu Max chọn Right, Min trả về 4.  
  → VM = 2 (vì Min cố giảm).  
  Nếu thay Min bằng **chance node** (ví dụ 50%-50%),  
  → VE = (2 + 4)/2 = 3.  
  ⇒ VM = 2 ≤ VE = 3

1. 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 VE **=> False**

Dùng chính sách minimax trong cây có yếu tố ngẫu nhiên:

* Minimax chọn Left (đảm bảo ít nhất 2 nếu đối thủ thông minh).
* Nhưng trong chance tree, Left có thể chỉ mang lại kỳ vọng 1 (nếu xác suất rủi ro cao).  
  → Thực tế nhận < VE ⇒ không đảm bảo đạt ít nhất VE.