Giải quyết vấn đề bằng tìm kiếm

Trí tuệ nhân tạo

HK1, 2022 - 2023

Nội dung

1. Giải quyết vấn đề bằng tìm kiếm
2. Chiến lược tìm kiếm cơ bản (uninformed search)
3. Bài tập
4. Chiến lược tìm kiếm với tri thức bổ sung (Informed search)
5. Bài tập

Giải quyết vấn đề bằng tìm kiếm

- Giải quyết vấn đề bằng tìm kiếm: tìm chuỗi các hành động cho phép đạt đến (các) trạng thái mong muốn.
- Các bước chính
  - Xác định **mục tiêu** cần đạt đến (goal formulation)
    - Là tập hợp của các trạng thái đích
    - Dựa trên trạng thái hiện tại (của môi trường) và đánh giá hiệu quả hành động (của tác tử)
  - Phát biểu **bài toán** (problem formulation): với một mục tiêu, xác định các *hành động* và *trạng thái* cần xem xét
  - Quá trình **tìm kiếm** (search process): xem xét các chuỗi hành động có thể $\rightarrow$ chọn chuỗi hành động tốt nhất
- Giải thuật tìm kiếm
  - Đầu vào: một bài toán (cần giải quyết)
  - Đầu ra: một giải pháp, dưới dạng một chuỗi các hành động cần thực hiện

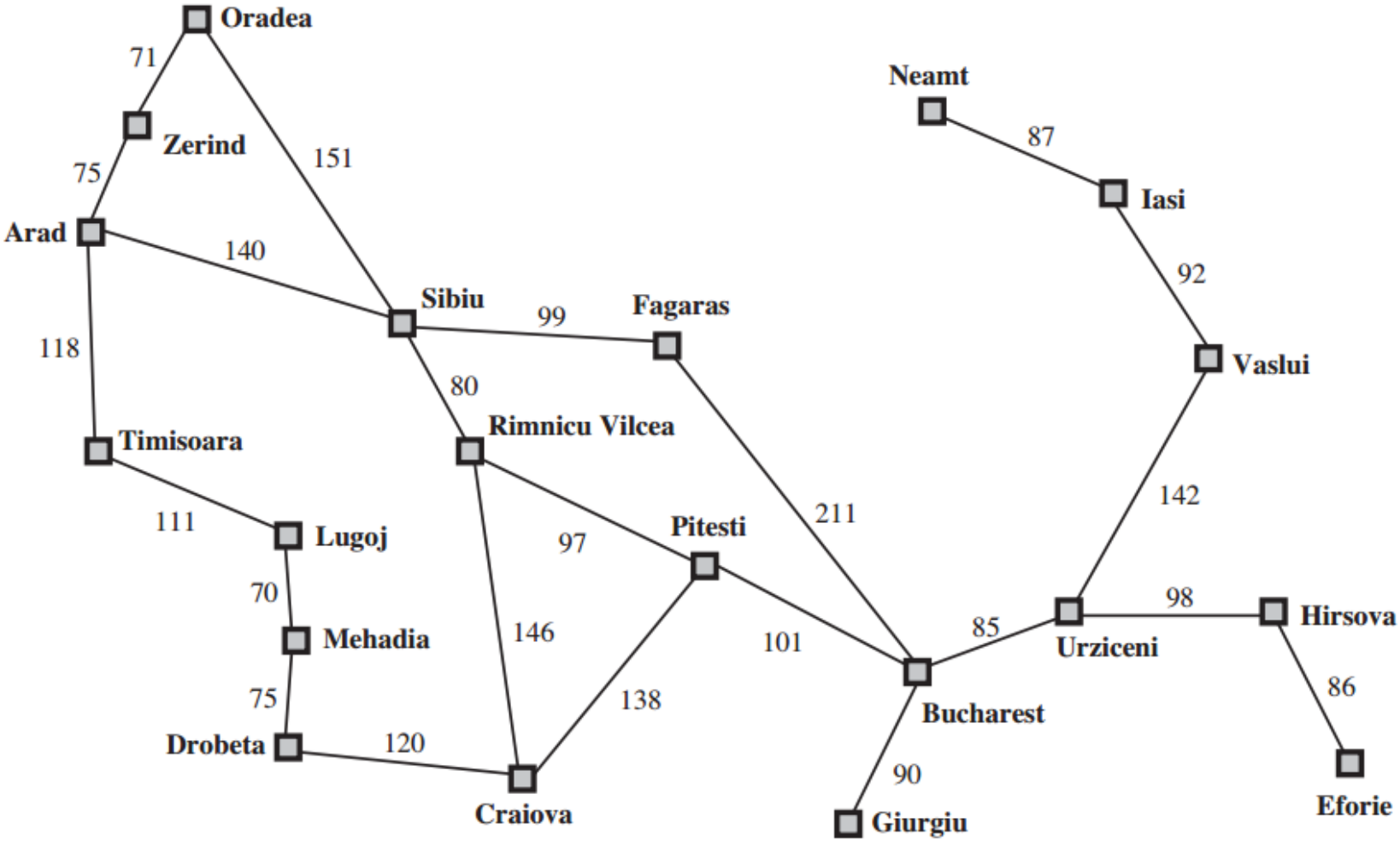
Tác tử giải quyết vấn đề

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **return** an action

static: *seq*, chuỗi hành động, khởi tạo rỗng
state, mô tả trạng thái hiện tại
goal, mục tiêu, khởi tạo *null*
problem, phát biểu bài toán

state ← UPDATE-STATE(*state*, *percept*)
if *seq* là rỗng **then**
goal ← FORMULATE-GOAL(*state*)
problem ← FORMULATE-PROBLEM(*state*, *goal*)
seq ← SEARCH(*problem*)
action ← FIRST(*seq*)
seq ← REST(*seq*)
return *action*

Ví dụ - Giải quyết vấn đề bằng tìm kiếm



Ví dụ - Giải quyết vấn đề bằng tìm kiếm

Một người đang du lịch Romania và

- hiện tại đang ở thành phố Arad
- ngày mai phải bay đến thành phố Bucharest
- sẽ di chuyển bằng xe đến Bucharest

Phát biểu **mục tiêu**:

- Cần phải có mặt tại Bucharest

Phát biểu **bài toán**:

- Các hành động (action): lái xe giữa các thành phố
- Các trạng thái (state): các thành phố (đi qua)

Tìm kiếm giải pháp

- Chuỗi các thành phố cần đi qua

Phát biểu bài toán

Một bài toán có thể được định nghĩa bởi **5** thành phần:

- **Trạng thái bắt đầu (initial state)**
  Ví dụ: trạng thái bắt đầu ở Romania có thể được viết *In(Arad)*
- **Các hành động có thể thực hiện (possible actions)** bởi tác tử.
  - Với một trạng thái s, hàm ACTIONS(s) sẽ trả về các hành động có thể thực hiện từ s. Mỗi hành động được gọi là áp dụng được (applicable) cho s
  - Ví dụ: từ trạng thái *In(Arad)* có thể thực hiện các hành động {*Go(Sibiu)*, *Go(Timisoara)*, *Go(Zerind)*}

Phát biểu bài toán

Một bài toán có thể được định nghĩa bởi **5** thành phần:

- **Mô hình chuyển đổi (transition model)**: kết quả thực hiện hành động
  - hàm RESULT(s,a): trả về kết quả thực hiện hành động a từ trạng thái s
  - *Trạng thái tiếp theo (successor)*: trạng thái đạt được từ một trạng thái cho trước khi thực hiện một hành động nào đó
  - Ví dụ: RESULT(*In(Arad)*, *Go(Zerind)*) = *In(Zerind)*
  - **Không gian trạng thái (state space)**: tất cả trạng thái đạt được từ một trạng thái bắt đầu bởi một chuỗi hành động
  - **Đường đi (path)**: chuỗi trạng thái được xác định bởi chuỗi các hành động

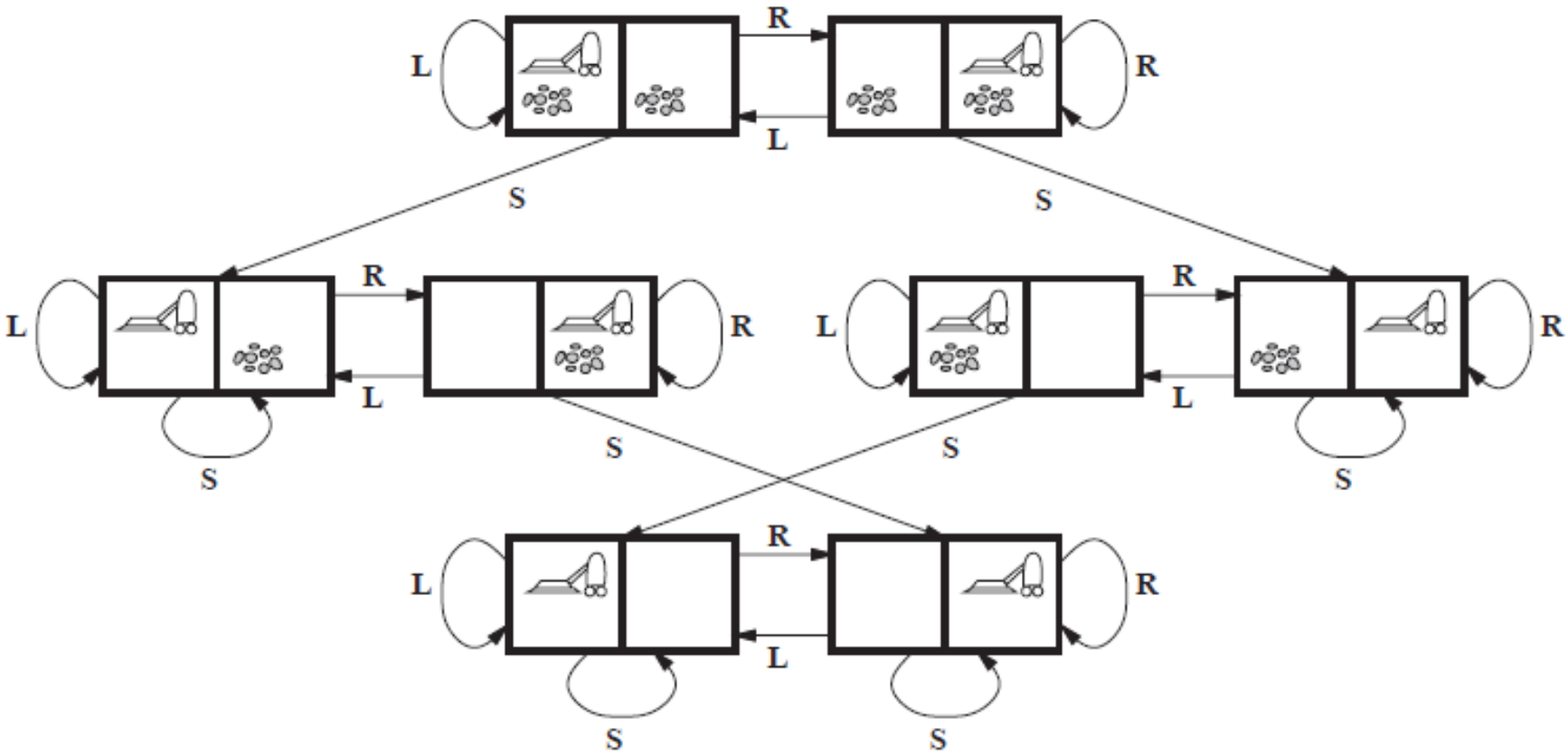
Phát biểu bài toán

Một bài toán có thể được định nghĩa bởi **5** thành phần:

- **Kiểm tra mục tiêu (goal test)**:
  - Trực tiếp: kiểm tra trạng thái s là (một trong các) trạng thái mục tiêu không. Ví dụ: mục tiêu của tác là tập $\{In(Bucharest)\}$
  - Gián tiếp: mục tiêu được chỉ bởi một thuộc tính trừu tượng
- **Hàm chi phí đường đi (path cost)**: trả về chi phí dưới dạng một số cụ thể ứng với một đường đi; tổng chi phí các hành động dọc theo đường đi
  - **Chi phí bước (step cost)** là chi phí thực hiện hành động a ở trạng thái s đạt được trạng thái s', ký hiệu $c(s, a, s')$

Ví dụ bài toán - Máy hút bụi

Bài toán máy hút bụi (vacuum world):



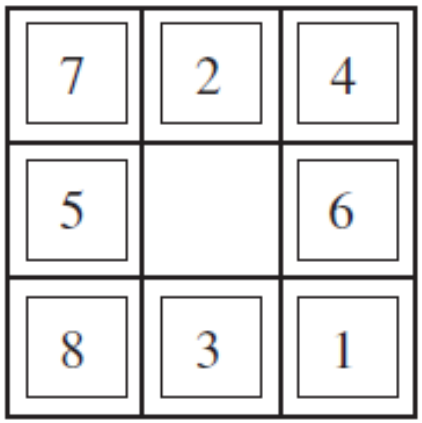
Ví dụ bài toán - Máy hút bụi

Bài toán máy hút bụi (vacuum world):

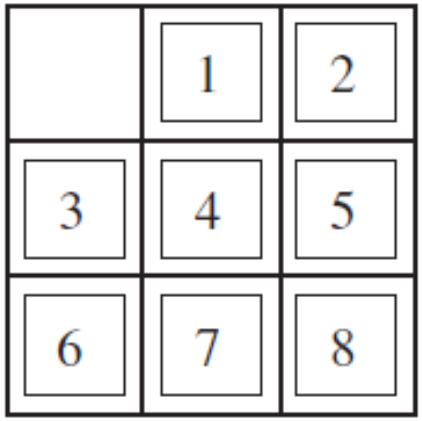
- *Các trạng thái*:
  <Ví trí máy hút bụi, {tình trạng các vị trí}>
- *Trạng thái bắt đầu*: bất kỳ trạng thái nào
- *Các hành động*: Sang trái (Left), sang phải (right), hút bụi (suck)
- *Mô hình chuyển đổi*: các hành động đều mang đến trạng thái mới trừ, Sang trái khi đang ở vị trí ô trái, Sang phải khi đang ở vị trí ô phải, Hút bụi khi đang ở ô sạch
- *Kiểm tra mục tiêu*: tất cả các ô sạch hay chưa
- *Chi phí đường đi*: mỗi bước có chi phí là 1 $\rightarrow$ chi phí đường đi là số bước

Ví dụ bài toán - 8-puzzle

Bài toán 8-puzzle



Start State



Goal State

- *Các trạng thái*
- *Trạng thái bắt đầu*
- *Các hành động*
- *Mô hình chuyển đổi*
- *Kiểm tra mục tiêu*
- *Chi phí đường đi*

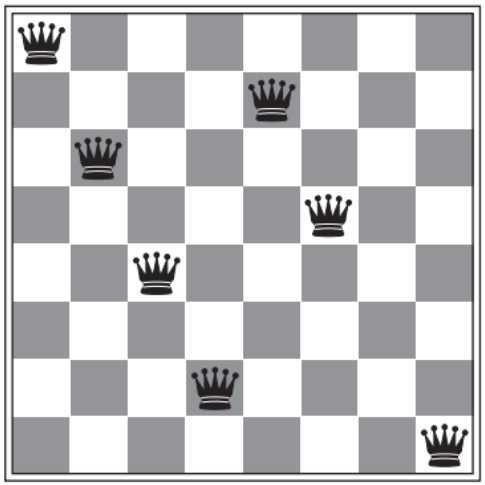
Ví dụ bài toán - 8-puzzle

Bài toán 8-puzzle

- *Các trạng thái*: các vị trí của ô số
- *Trạng thái bắt đầu*: bất kỳ trạng thái nào
- *Các hành động*: di chuyển ô trắng Sang trái (Left), Sang phải (Right), Lên (Up), Xuống (Down)
- *Mô hình chuyển đổi*: Với một trạng thái và một hành động $\rightarrow$ trạng thái mới
- *Kiểm tra mục tiêu*: kiểm tra trạng thái có đúng với cấu hình mục tiêu không
- *Chi phí đường đi*: mỗi bước có chi phí là 1 $\rightarrow$ chi phí đường đi là số bước

Ví dụ bài toán - 8-queens

Bài toán 8-queens



(1) Mô tả hướng tăng trưởng (incremental formulation): bắt đầu với bàn cờ trống, mỗi hàng động ứng với việc thêm một quân hậu

(2) Mô tả hướng hoàn chỉnh (complete-state formulation): bắt đầu với bàn cờ gồm 8 quân hậu và hành động là di chuyển một quân hậu theo một hướng nào đó

Ví dụ bài toán - 8-queens

Bài toán 8-queens (trường hợp 1)

- *Các trạng thái*: sắp xếp từ 0 - 8 quân hậu trên bàn cờ
- *Trạng thái bắt đầu*: bàn cờ trống
- *Các hành động*: thêm 1 quân hậu vào một vị trí trống
- *Mô hình chuyển đổi*: trả về một bàn cờ với 1 quân hậu được thêm vào một ô cụ thể
- *Kiểm tra mục tiêu*: 8 quân trên bàn cờ, không quân nào tấn công quân khác

Ví dụ bài toán - Bài toán đổ nước vào bình

Chúng ta có 1 bình 3 lít và 1 bình 4 lít. Chúng ta có thể đổ nước vào bình, đổ nước ra khỏi bình, đổ nước 1 bình này vào một bình khác (đến khi hết nước hoặc khi bình đó đầy)

- *Các trạng thái*
- *Trạng thái bắt đầu*
- *Các hành động*
- *Mô hình chuyển đổi*
- *Kiểm tra mục tiêu*

Ví dụ bài toán - Tu sĩ và con quỷ

03 tu sĩ và 03 con quỷ muốn qua sông. Chiếc thuyền chỉ có thể chở tối đa 02 người và tối thiểu 01 người qua sông.
Ràng buộc: số tu sĩ không được ít hơn số con quỷ tại các bờ sông

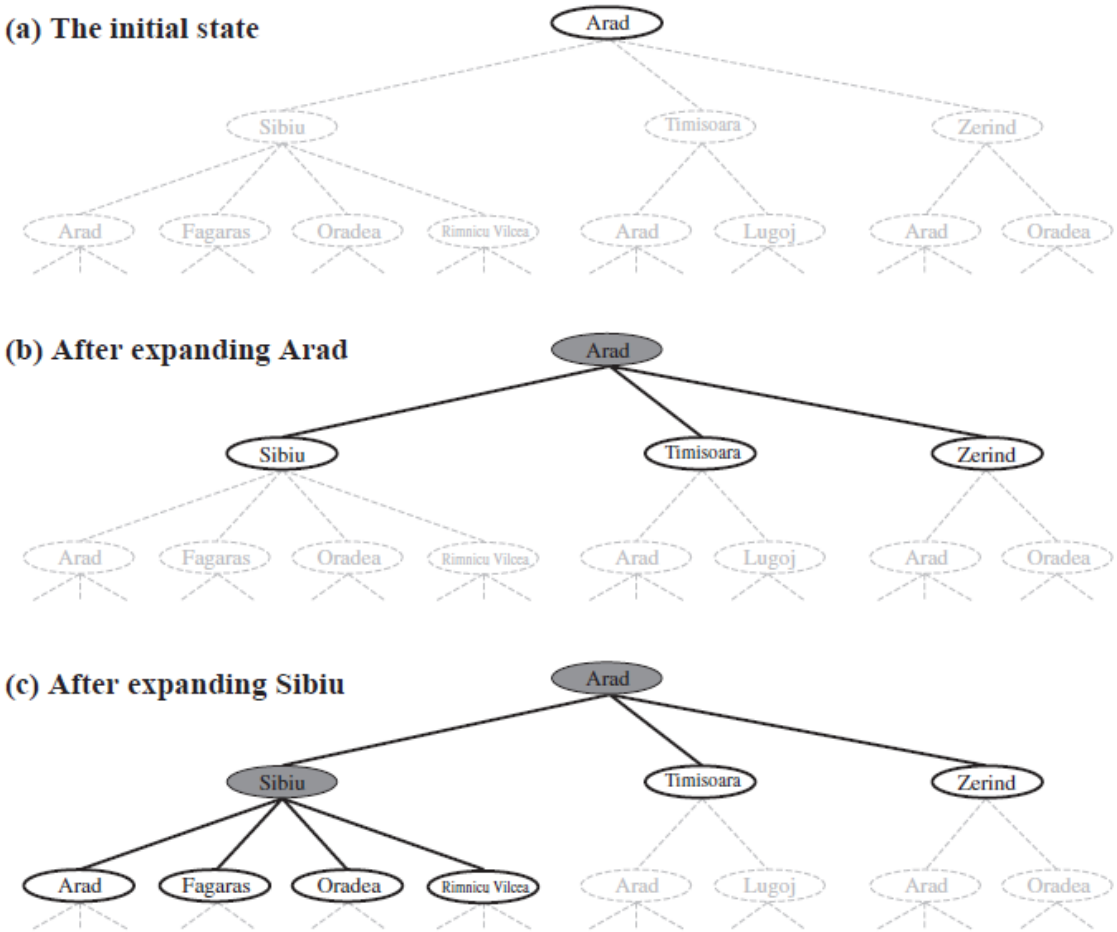
- *Các trạng thái*
- *Trạng thái bắt đầu*
- *Các hành động*
- *Mô hình chuyển đổi*
- *Kiểm tra mục tiêu*

Tìm kiếm giải pháp

Tìm kiếm (search): tìm kiếm một chuỗi các hành động đạt được mục tiêu.

- Giải pháp của một vấn đề là một chuỗi hành động $\rightarrow$ thuật toán tìm kiếm xem xét các chuỗi hành động có thể có
- Các chuỗi hành động: bắt đầu từ trạng thái ban đầu tạo thành cây tìm kiếm

Tìm kiếm giải pháp



Tìm kiếm theo cấu trúc cây

Thuật toán tìm kiếm trên cây (TREE-SEARCH) được trình bày như sau:

function TREE-SEARCH(*problem*) **return** a solution or failure

khởi tạo tập biên (frontier) với trạng thái bắt đầu

loop

if tập biên rỗng **then return** *failure*

chọn một nút lá và bỏ nó ra khỏi tập biên

if nếu nút chứa trạng thái đích **then return** giải pháp tương ứng

mở rộng nút đã chọn, thêm nút kết quả vào tập biên

Tìm kiếm theo cấu trúc đồ thị

Thuật toán tìm kiếm theo đồ thị (GRAPH-SEARCH) được trình bày như sau:

function GRAPH-SEARCH(problem) **return** a solution, or failure

khởi tạo tập biên (frontier) với trạng thái bắt đầu

khởi tạo tập đã duyệt (explored set) là rỗng

loop

if tập biên rỗng **then return** *failure*

chọn một nút lá và bỏ nó ra khỏi tập biên

if nếu nút chứa trạng thái đích **then return** giải pháp tương ứng

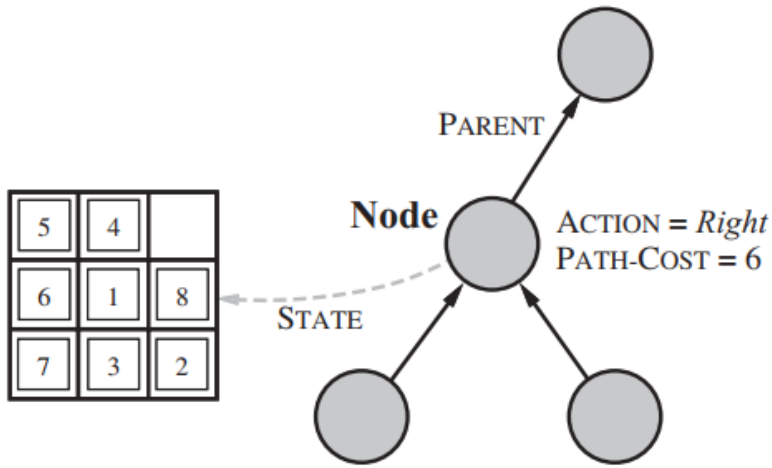
thêm nút đã xét vào tập duyệt

mở rộng nút đã chọn, thêm nút kết quả vào tập biên ***nếu nó không có trong tập biên hay tập duyệt***

Hạ tầng của giải thuật tìm kiếm

Mỗi nút n có cấu trúc gồm 4 phần

- n.STATE: trạng thái tương ứng với nút
- n.PARENT: nút đã sinh ra nút hiện tại
- n.ACTION: hành động thực hiện ở nút cha để sinh ra nút hiện tại
- n.PATH-COST: $g(n)$, chi phí đi từ nút bắt đầu đến nút hiện tại



Hạ tầng của giải thuật tìm kiếm

Với một nút cha, nút con được tìm như thế nào?

function CHILD-NODE(problem, parent, action)
return node
STATE = *problem*.RESULT(*parent*.STATE, action)
PARENT = *parent*
ACTION = *action*
PATH-COST = *parent*.PATH-COST +
problem.STEP-COST(*parent*.STATE, *action*)

Tiêu chí cho chiến lược đánh giá tìm kiếm

Đầy đủ (Completeness)	Chiến lược đảm bảo tìm được giải pháp nếu có không?
Độ phức tạp thời gian (Time Complexity)	Mất bao lâu để tìm được giải pháp
Độ phức tạp về bộ nhớ (Space Complexity)	Bao nhiêu không gian bộ nhớ cần cho việc tìm kiếm
Độ tối ưu (Optimality)	Chiến lược tìm được giải pháp tốt nhất (với chi phí đường đi thấp nhất)?

Độ phức tạp về thời gian và bộ nhớ được đánh giá bởi:

- b: hệ số phân nhánh tối đa của cây
- d: độ sâu của lời giải có chi phí thấp nhất
- m: độ sâu tối đa của không gian trạng thái (độ sâu của cây)

Chiến lược tìm kiếm cơ bản (uninformed search)

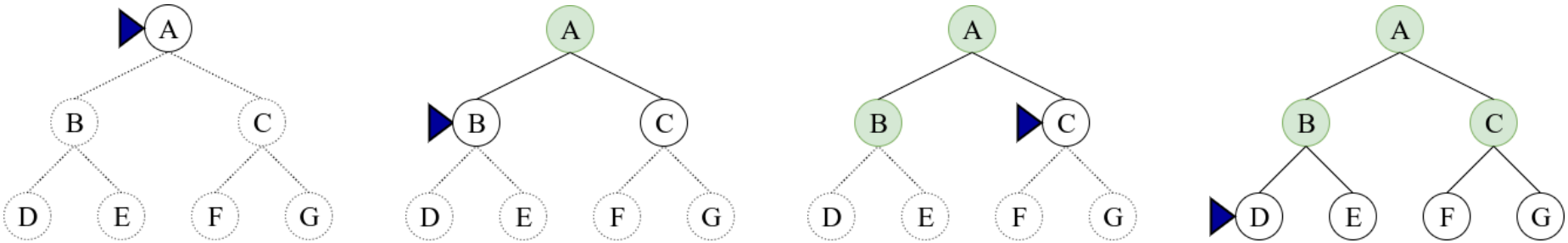
Chỉ sử dụng các thông tin chứa trong định nghĩa của bài toán

- Tìm kiếm theo chiều rộng (Breadth-first search)
- Tìm kiếm với chi phí cực tiểu (Uniform-cost search)
- Tìm kiếm theo chiều sâu (Depth-first search)
- Tìm kiếm giới hạn độ sâu (Depth-limited search)
- Tìm kiếm sâu dần (Iterative deepening search)
- ...

Còn gọi là chiến lược tìm kiếm mù (blind search)

Tìm kiếm theo chiều rộng - BFS

Các nút được mở rộng theo trình tự sinh ra (*frontier* $\rightarrow$ hàng đợi FIFO)



Tìm kiếm theo chiều rộng - BFS

function BREADTH-FIRST-SEARCH(problem) **return** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

if *problem*.GOAL-TEST(node.STATE) **then return** SOLUTION(node)

frontier ← a FIFO queue with *node* as the only element

explored← an empty set

loop

if EMPTY(frontier) **then return** failure

node ← POP(*frontier*)

if *node*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

add *node*.STATE to *explored*

for each *action* in *problem*.ACTIONS(*node*.STATE) **do**

child ← CHILD-NODE(*problem*, *node*, *action*)

if *child*.STATE is not in *explored* or *frontier* **then**

frontier ← INSERT(*child*, *frontier*)

Tìm kiếm theo chiều rộng - BFS

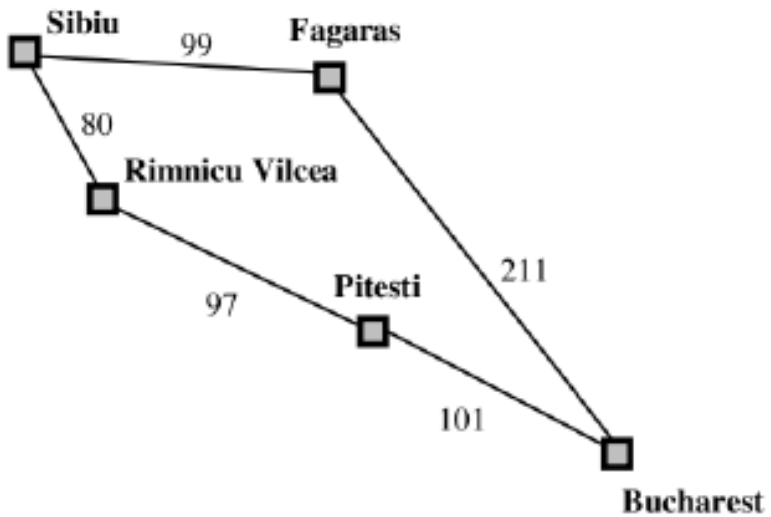
Đầy đủ:
Có (nếu b hữu hạn)
Tối ưu:
Không (trừ khi chi phí các bước là bằng nhau)
Độ phức tạp thời gian:
Số lượng nút tối đa được mở rộng

$$1 + b + b^2 + b^3 + \cdots + b^d + b(b^d - 1) \in O(b^{d+1})$$

Độ phức tạp bộ nhớ:
Mọi nút sinh ra đều được lưu trong bộ nhớ. Không gian cần cho đường biên là $O(b^{d+1})$

Tìm kiếm chi phí cực tiểu

- nếu chi phí bước thực hiện hành động là bằng nhau thì BFS tìm đường đi với chi phí tối ưu.
- nếu chi phí khác nhau (ví dụ: bản đồ lái xe từ một điểm đến một nơi khác có thể khác nhau về khoảng cách), UCS tìm giải pháp tối ưu.
- UCS mở rộng nút với chi phí đường đi thấp nhất $g(n)$



Tìm kiếm chi phí cực tiểu?

function UNIFORM-COST-SEARCH(problem) **return** a solution, or failure

node← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

frontier ← a priority queue ordered by PATH-COST, with *node* as the only element

explored ← an empty set

loop

if EMPTY?(*frontier*) **then return** failure

node ← POP(*frontier*)

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

add *node*.STATE to *explored*

for each *action* **in** *problem*.ACTION(*node*.STATE) **do**

child ← CHILD-NODE(*problem*, *node*, *action*)

if *child*.STATE is not *explored* or *frontier* **then**

frontier ← INSERT(*child*, *frontier*)

else if *child*.STATE is in *frontier* with higher PATH-COST **then**

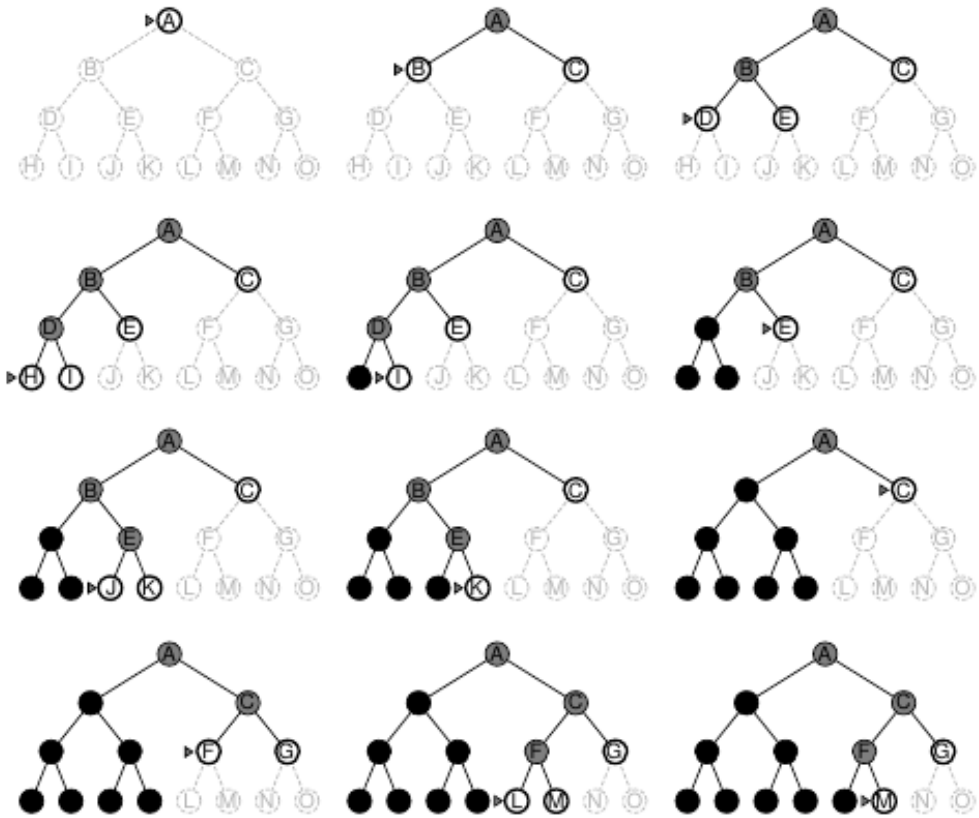
replace that *frontier* node with *child*

Tìm kiếm chi phí cực tiểu

- **Đầy đủ**
  Có (nếu các bước $\geq \epsilon > 0$)
- **Tối ưu**
  Có
- **Độ phức tạp thời gian**
  $O(b^{1+\lfloor \frac{\text{cost}(P^*)}{\epsilon} \rfloor})$, với P^* là giải pháp tối ưu
- **Độ phức tạp không gian**
  tương tự như độ phức tạp thời gian khi toàn bộ cây tìm kiếm được lưu trong bộ nhớ

Tìm kiếm theo chiều sâu

- Phát triển các nút chưa xét theo chiều sâu - Các nút được xét theo thứ tự độ sâu giảm dần
- Sử dụng hàng đợi LIFO



Tìm kiếm theo chiều sâu

function DEPTH-FIRST-SEARCH(*problem*) **return** a solution, or failure

return RECURSIVE-DFS(MAKE-NODE(*problem*.INITIAL-STATE),*problem*)

function RECURSIVE-DFS(*node*,*problem*) **return** a solution, or failure

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

else

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

child $\leftarrow$ CHILD-NODE(*problem*, *node*, *action*)

result $\leftarrow$ RECURSIVE-DFS(*child*, *problem*)

if *result* $\neq$ *failure* **then return** *result*

return *failure*

Tìm kiếm theo chiều sâu

- **Đầy đủ**
  Không
- **Tối ưu**
  Không
- **Độ phức tạp thời gian**
  $O(b^m)$
- **Độ phức tạp không gian**
  $O(bm)$

Tìm kiếm giới hạn độ sâu

- Dựa trên DFS + sử dụng giới hạn về độ sâu *limit* trong quá trình tìm kiếm

function DEPTH-LIMITED-SEARCH(*problem*, *limit*) **return** a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(*problem*.INITIAL-STATE),*problem*,*limit*)

function RECURSIVE-DLS(*node*,*problem*, *limit*) **return** a solution, or failure/cutoff
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  else if *limit* = 0 **then return** *cutoff*
  else
    cutoff_occurred← false
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← CHILD-NODE(*problem*, *node*, *action*)
      result ← RECURSIVE-DLS(*child*, *problem*,*limit*-1)
      if *result*=*cutoff* **then** *cutoff_occurred*?← true
      else if *result*≠ *failure* **then return** *result*
    if *cutoff_occurred*? **then return** *cutoff*
    elsereturn *failure*

Tìm kiếm theo chiều sâu

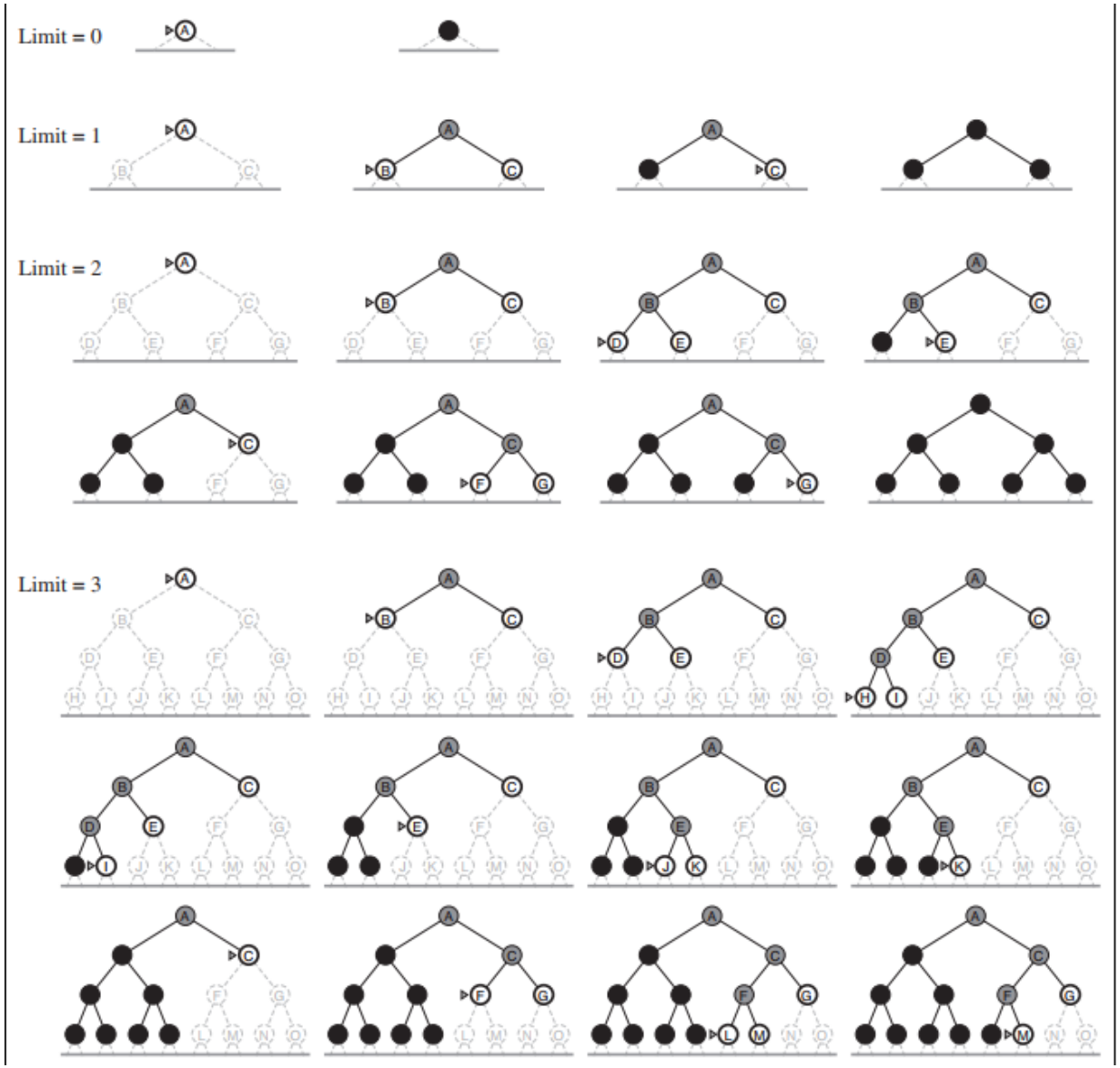
- **Đầy đủ**
  Không (nếu $d > \text{limit}$)
- **Tối ưu**
  Không
- **Độ phức tạp về thời gian**
  $O(b^{\text{limit}})$
- **Độ phức tạp về không gian**
  $O(b \cdot \text{limit})$

Tìm kiếm sâu dần - IDS

- Vấn đề với giải thuật tìm kiếm với độ sâu giới hạn (DLS):
  - Nếu tất cả các lời giải nằm ở độ sâu lớn hơn giới hạn độ sâu *limit* thì không tìm được lời giải
- Giải thuật tìm kiếm sâu dần
  - Áp dụng giải thuật DLS đối với độ sâu ≤ 1
  - Nếu thất bại, tiếp tục với độ sâu ≤ 1
  - ... tiếp tục cho đến khi: 1) tìm được lời giải hoặc 2) toàn bộ cây đã được xét mà không tìm được lời giải
- lặp lại với độ sâu tối đa maxdepth $\rightarrow$ tăng dần

function ITERATIVE-DEEPENING-SEARCH(*problem*) **return** a solution or failure
  for *depth* = 0 **to** inf **do**
    result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result*$\neq$ cutoff **then return** *result*

Tìm kiếm sâu dần

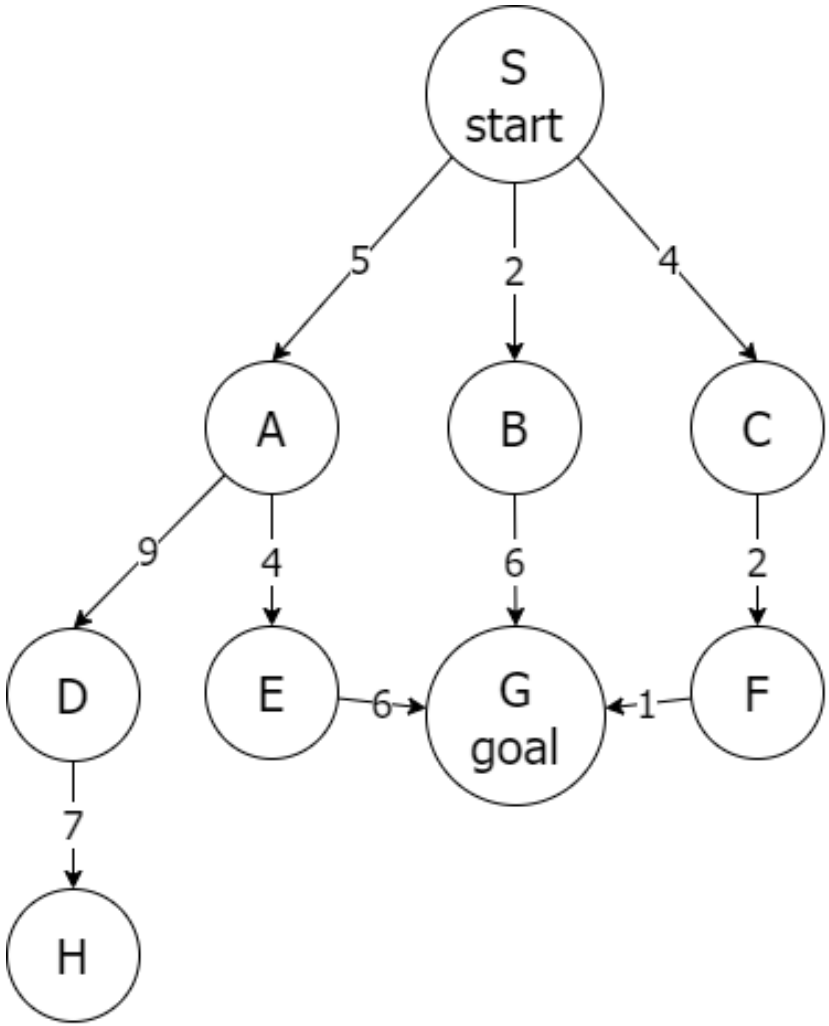


Tìm kiếm sâu dần - IDS

- **Đầy đủ**
  Có
- **Tối ưu**
  Không (trừ khi chi phí các bước bằng nhau)
- **Độ phức tạp về thời gian**
  $O((d+1) + db + (d-1)b^2 + \cdots + b^d) = O(b^d)$
- **Độ phức tạp về không gian**
  $O(bd)$

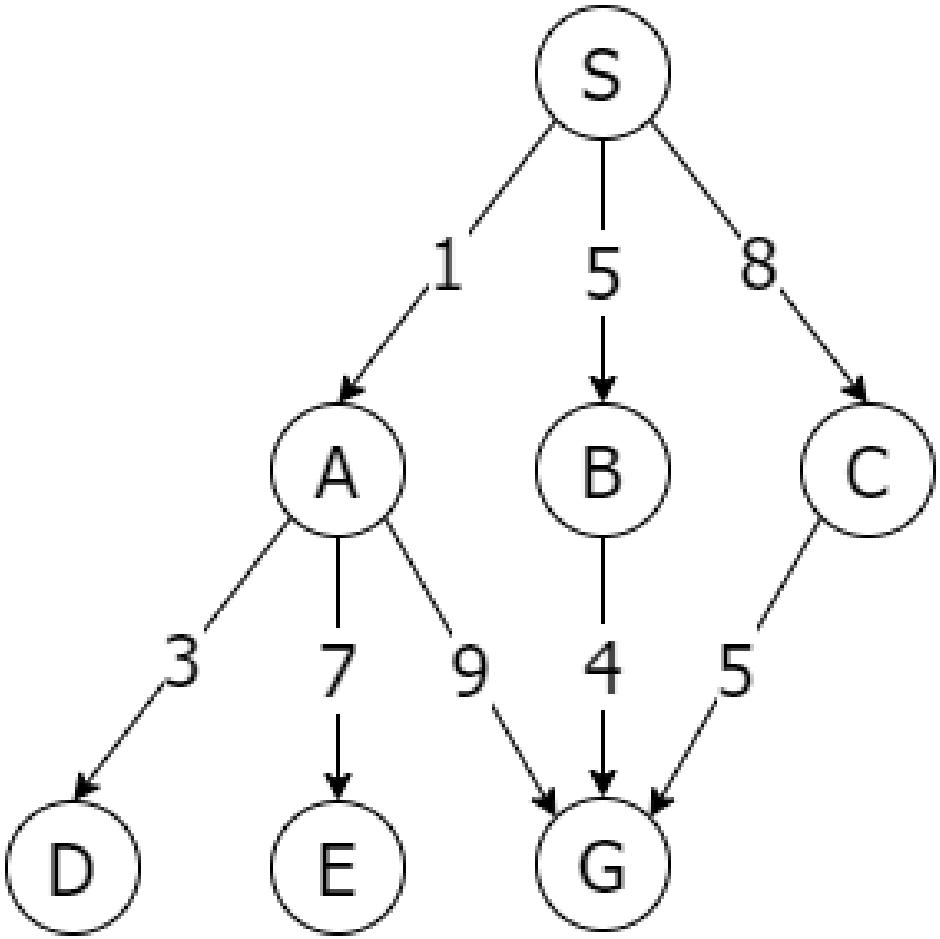
Bài tập 1

Áp dụng giải thuật: BFS, UCS, DFS, IDS



Bài tập 2

Áp dụng giải thuật: BFS, UCS, DFS, IDS



Tìm kiếm với tri thức bổ sung

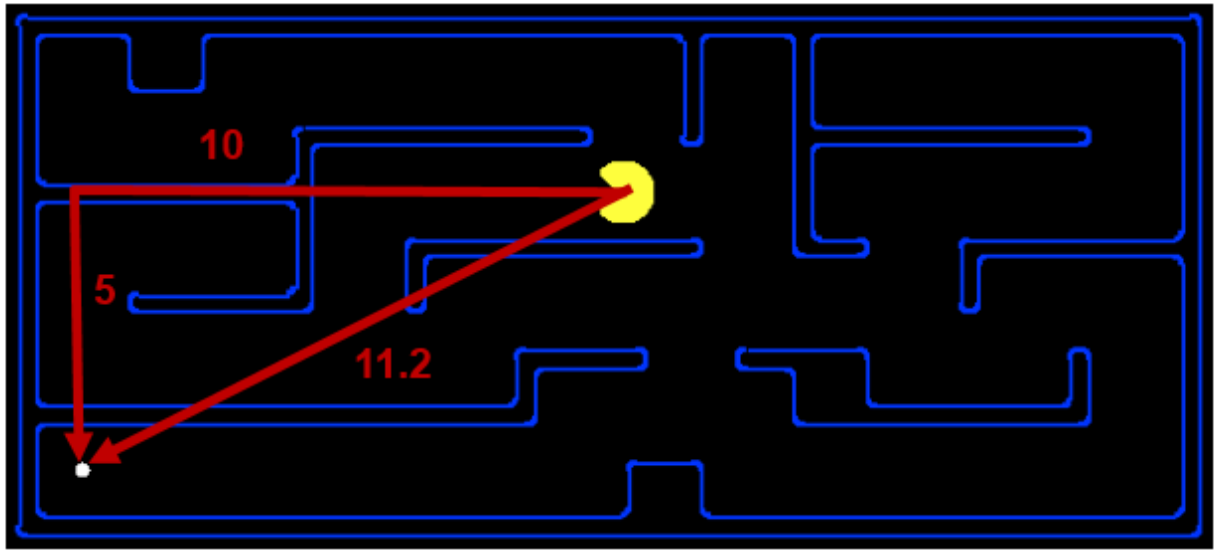
Sử dụng định nghĩa bài toán và *tri thức cụ thể của bài toán*. Còn được gọi là tìm kiếm heuristics

- Phương pháp chung: tìm kiếm tốt nhất (best-first search)
- Tìm kiếm theo chiến lược tham ăn (greedy best-first search)
- Tìm kiếm A*

Tìm kiếm với tri thức bổ sung

Một heuristic là

- hàm ước tính mức độ gần nhau của một trạng thái với trạng thái đích
- được thiết kế cho một vấn đề tìm kiếm cụ thể
- Ví dụ: khoảng cách Manhattan, khoảng cách Euclid...



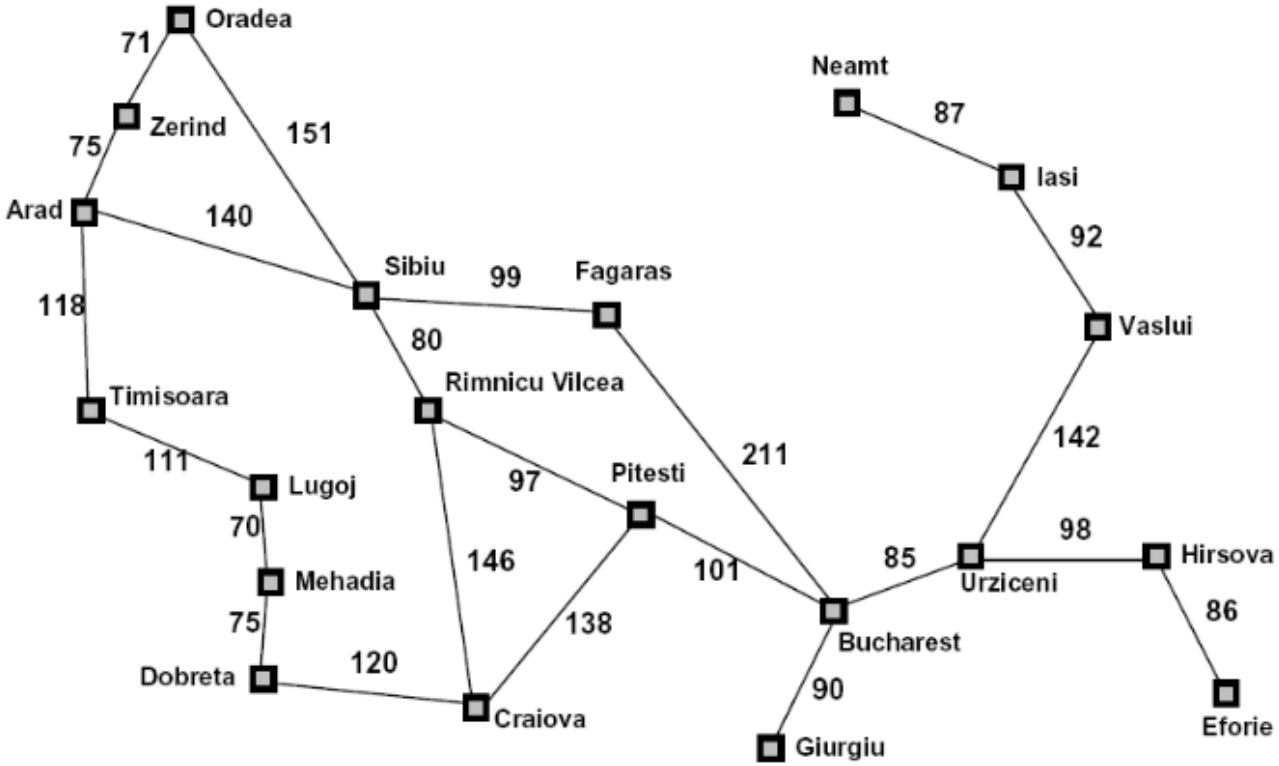
Best-first search

- một trường hợp của TREE-SEARCH hay GRAPH-SEARCH
- nút được chọn mở rộng dựa vào một **hàm đánh giá (evaluation function)** $f(n)$
- nút được đánh giá thấp nhất được mở rộng trước
- triển khai tương tự UCS nhưng thay hàm g thành f
- f bao gồm hàm *heuristic* $h(n)$
  trong đó $h(n) =$ ước lượng chi phí đường đi ngắn nhất từ trạng thái nút n đến trạng thái đích
- nếu n nút đích, $h(n) = 0$

Tìm kiếm tham ăn - Greedy best-first search

- mở rộng nút gần mục tiêu nhất $\rightarrow$ có khả năng dẫn đến giải pháp nhanh nhất
- $f(n) = h(n)$
- Ví dụ: Trong bài toán tìm đường từ Arad đến Bucharest, sử dụng $h_{SLD}(n) =$ ước lượng khoảng cách đường thẳng ("chim bay") từ thành phố hiện tại n đến Bucharest

Greedy best-first search - Ví dụ



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Greedy best-first search - Đặc điểm

- **Hoàn chỉnh**
  Không
- **Tối ưu**
  Không
- **Độ phức tạp về thời gian**
  $O(b^m)$ (hàm heuristic tốt có thể cải thiện lớn)
- **Độ phức tạp về không gian**
  $O(b^m)$ (lưu trữ tất cả nút trong bộ nhớ)

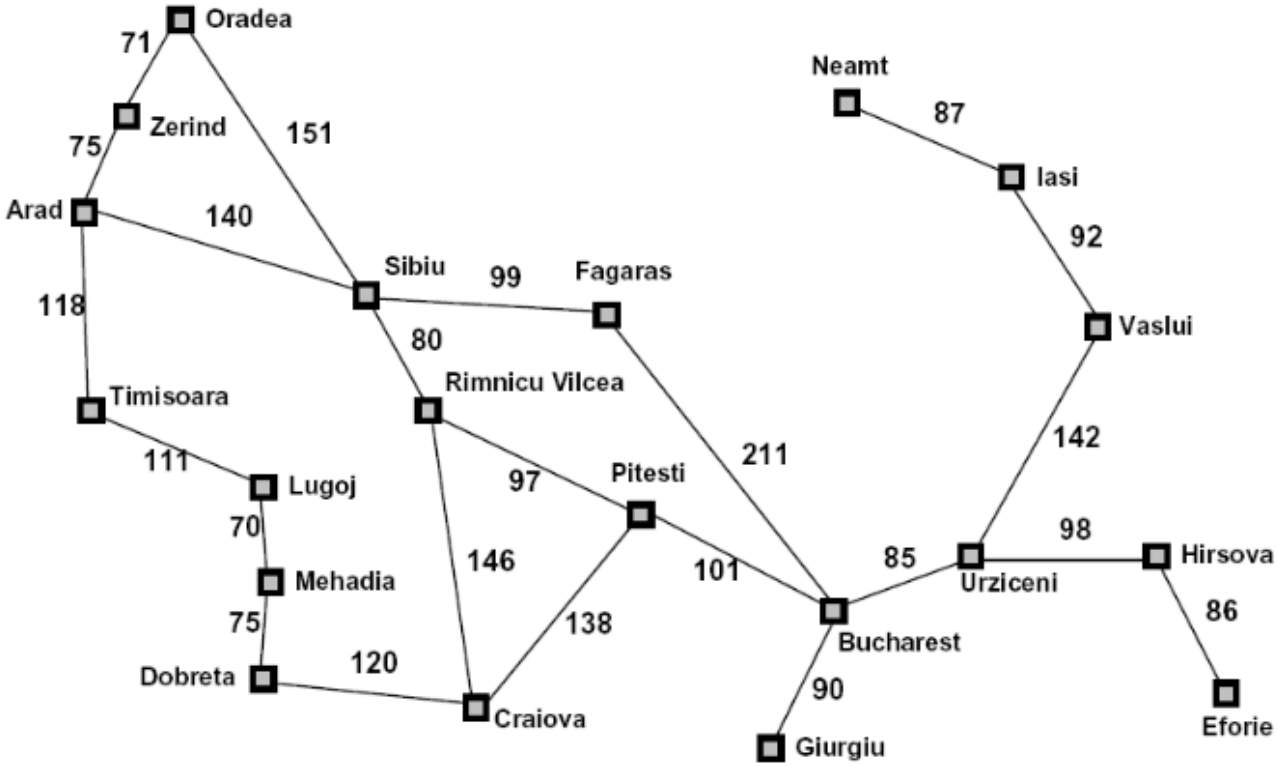
Tìm kiếm A*

- $f(n) = g(n) + h(n)$
  với $g(n)$ chi phí từ nút gốc đến nút n
- $f(n) =$ chi phí ước lượng của giải pháp "rẻ nhất"qua n

Cho $h^*(n)$ là chi phí thực sự của đường tối ưu từ n đến đích. Một ước lược $h(n)$ được xem là chấp nhận được nếu $\forall n : 0 \leq h(n) \leq h^*(n)$

A* yêu cầu h là *chấp nhận được (admissible)* $\approx$ ước tính chi phi "lạc quan"so với chi phí thực sự

Tìm kiếm A* - Ví dụ



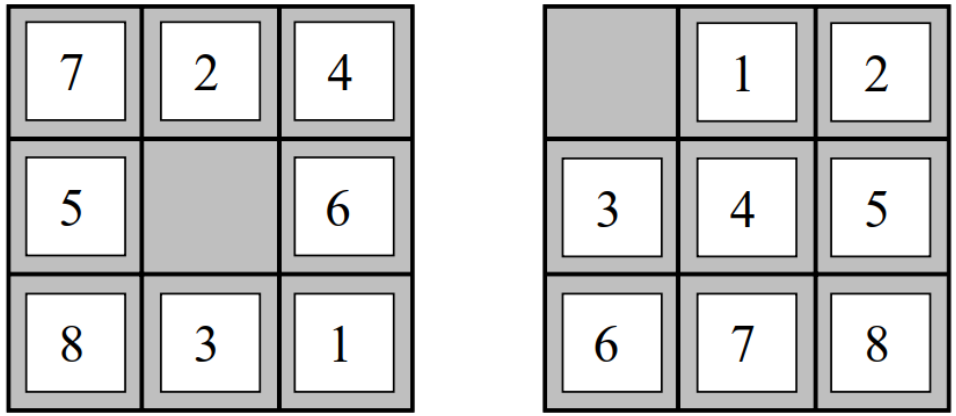
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Tìm kiếm A* - Đặc điểm

- Nếu *không gian các trạng thái là hữu hạn* và *giải pháp để tránh việc xét lại các trạng thái* $\rightarrow$ hoàn chỉnh nhưng không đảm bảo tối ưu
- Nếu *không gian các trạng thái là hữu hạn* và *không có giải pháp để tránh việc xét lại các trạng thái* $\rightarrow$ không hoàn chỉnh
- Nếu *không gian các trạng thái là vô hạn* $\rightarrow$ không hoàn chỉnh

Hàm ước lượng chấp nhận được



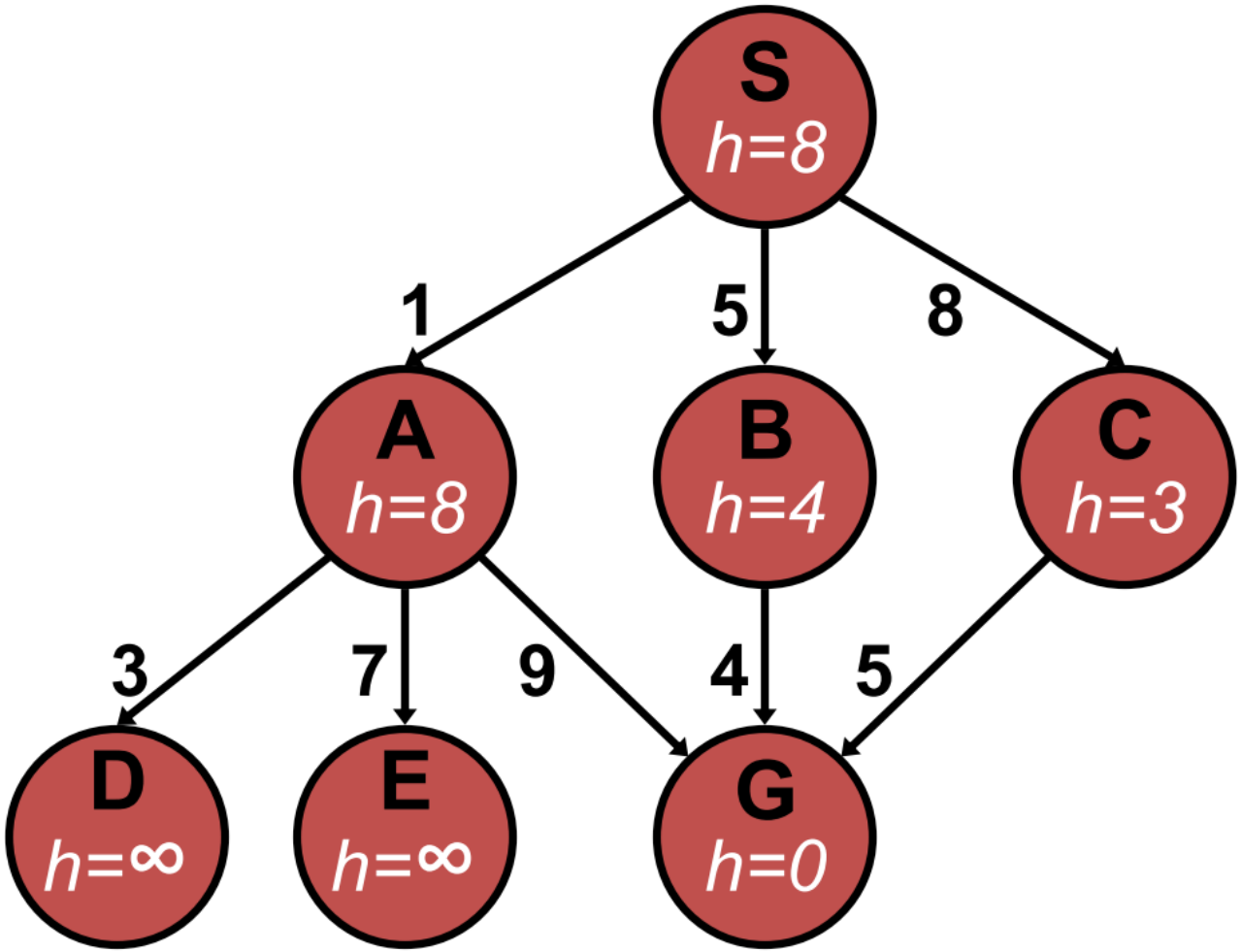
- $h_1(n) =$ số các ô số nằm sai vị trí
- $h_2(n) =$ tổng khoảng cách dịch chuyển ngắn nhất ô số nằm về đúng vị trí
- $h_3 = 0$
- $h_4 = 1$

Ước lượng ưu thế

- Ước lượng h_2 được gọi là ưu thế hơn / trội hơn (dominate) ước lượng h_1 nếu
  - $h_1(n)$ và $h_2(n)$ đều là ước lượng chấp nhận được
  - $h_1(n) \leq h_2(n)$ với tất cả các nút n
- Nếu ước lượng h_2 ưu thế hơn $h_1 \rightarrow h_2$ tốt hơn cho quá trình tìm kiếm
- Ví dụ trong bài toán 8 ô số: chi phí tìm kiếm = số lượng trung bình của các nút phải xét và với độ sâu $d = 12$
  - IDS: 3.644.035 nút phải xét
  - A* (sử dụng h_1): 227 nút phải xét
  - A* (sử dụng h_2): 73 nút phải xét

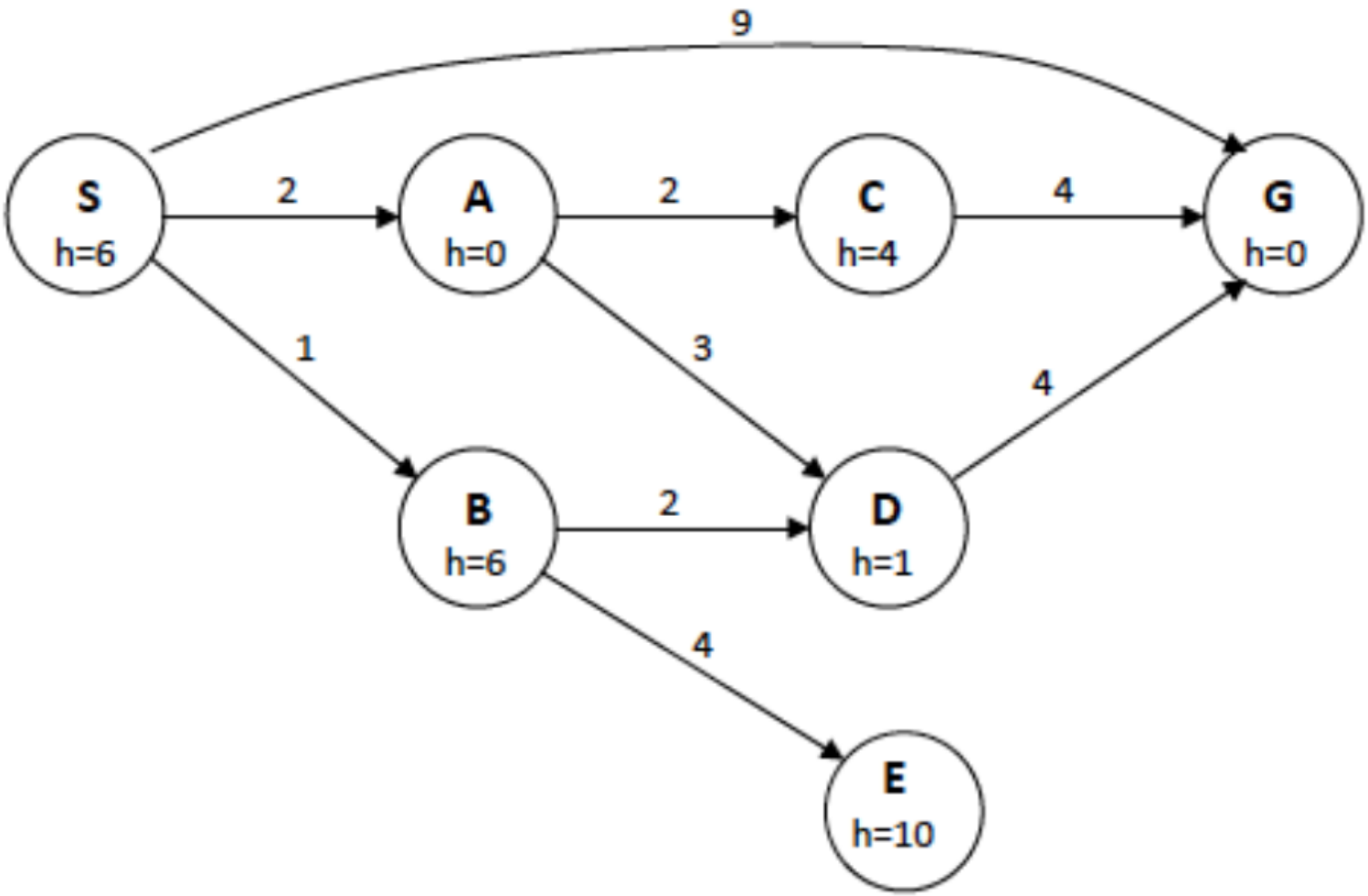
Bài tập 3

Áp dụng giải thuật: Greedy-best first search, A* search



Bài tập 4

Áp dụng giải thuật: Greedy-best first search, A* search



Bài tập 5

Áp dụng giải thuật: Greedy-best first search, A* search (Start: A, Goal: B)