Trí tuệ nhân tạo (Artificial Intelligence)

Tìm kiếm không có thông tin bổ sung

By Hoàng Hữu Việt
Email: viethh@vinhuni.edu.vn
Viện Kỹ thuật và Công nghệ, Đại học Vinh

Vinh, 3/2019

Tài liệu

- Tài liệu chính

  [1] Stuart Russell, Peter Norvig. Artificial Intelligence. A modern approach. 3rd ed. Prentice Hall, 2009.

- Tài liệu khác

  [2] Milos Hauskrecht. Artificial Intelligence, 2013.
  people.cs.pitt.edu/~milos/courses/cs1571-Fall2013/

Nội dung

- Giới thiệu
- 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 chiều sâu giới hạn độ sâu (depth-limited search)
- Tìm kiếm sâu dần (iterative deepening depth-first search)
- Tìm kiếm từ hai hướng (bidirectional search)
- Bài tập

Các phương pháp tìm kiếm cơ bản

- Các phương pháp tìm kiếm không có thông tin bổ sung:
  - Là các phương pháp tìm kiếm cơ bản.
  - Tên gọi khác: tìm kiếm mù (blind search).
- Các chiến lược tìm kiếm không có thêm thông tin bổ sung về các trạng thái mà chỉ sử dụng các thông tin trong định nghĩa của bài toán.
- Các chiến lược tìm kiếm chỉ là sinh ra các trạng thái và kiểm tra trạng thái đích.
- Các chiến lược tìm kiếm được phân biệt bởi sự khác nhau về thứ tự các nút (node) được mở rộng.

Tìm kiếm theo chiều rộng

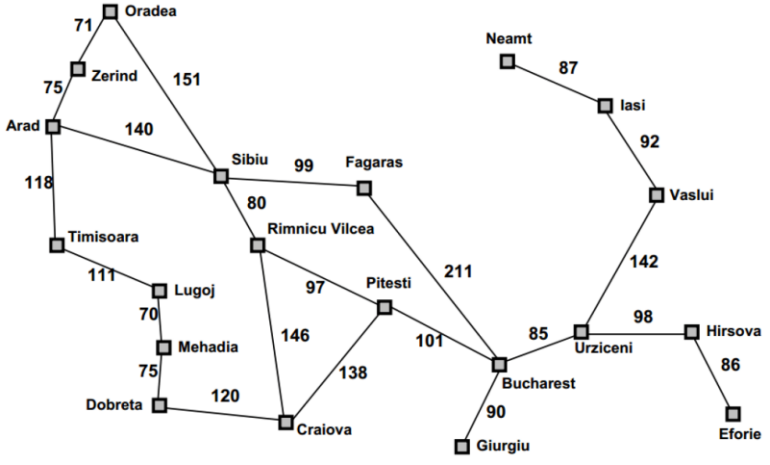
- Nút "nông nhất" (*shallowest*) chưa được mở sẽ được chọn để mở rộng.

```
function BREADTH-FIRST-SEARCH(problem) returns 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 do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)  /* chooses the shallowest node in frontier */
        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
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

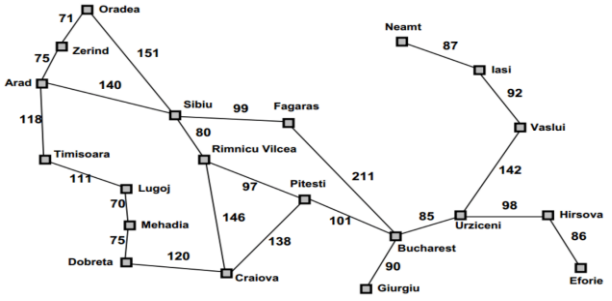
Figure 3.11 Breadth-first search on a graph.

Tìm kiếm theo chiều rộng

- Đặt nút con vào CUỐI (FIFO) danh sách *frontier*
- Ví dụ tìm đường đi từ *Arad* → *Bucharest*



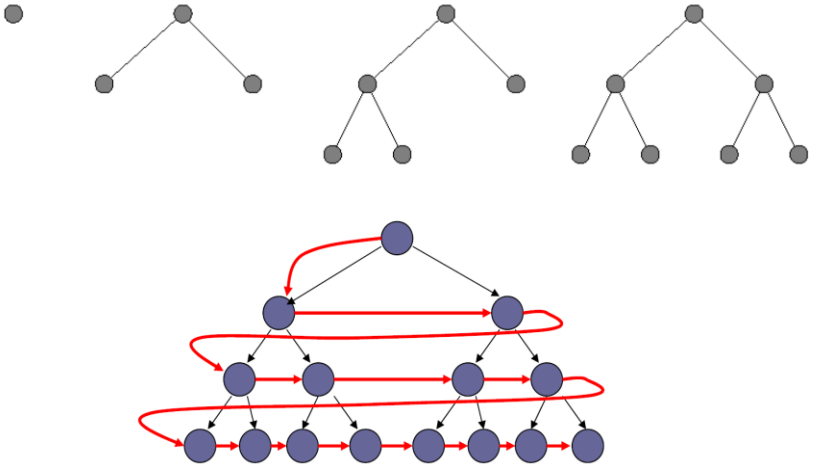
Tìm kiếm theo chiều rộng



- Khởi tạo: *node* = {?}, *frontier* = {?}, *explored* = {?}
- Lặp 1: *node* = {?}, *explored* = {?}, *frontier* = {?}
- Lặp 2: *node* = {?}, *explored* = {?}, *frontier* = {?}
- …
- Kết thúc: đường đi ? cây tìm kiếm ?

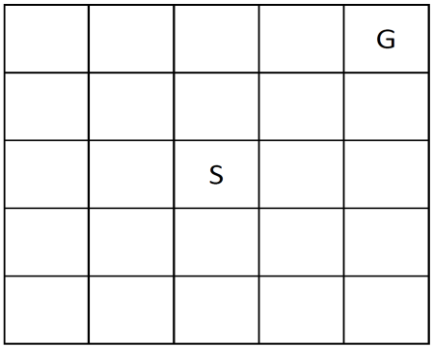
Tìm kiếm theo chiều rộng

- Nút "nông nhất" (*shallowest*) chưa được mở sẽ được chọn để mở rộng.



Tìm kiếm theo chiều rộng

- Ví dụ tìm đường đi cho robot trên một lưới
  - Điểm bắt đầu S và đích đến là G.
  - Mỗi vị trí robot có thể đi đến các ô tiếp theo theo 4 hướng ←,↑,↓,→.
- Nhận xét gì về không gian tìm kiếm của thuật toán?

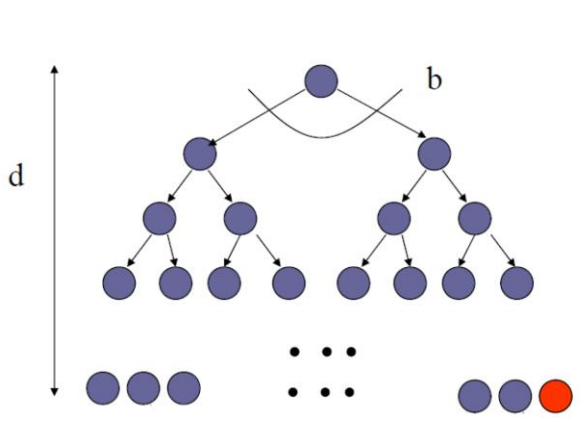


Tìm kiếm theo chiều rộng

- Hoàn chỉnh (completeness): Yes
  - Thuật toán tìm được nghiệm nếu bài toán có nghiệm.
- Tối ưu (optimality): Yes
  - Nghiệm tìm được là tối ưu (ví dụ đường đi qua ít đỉnh nhất).
- Độ phức tạp thời gian (time complexity)?
- Độ phức tạp không gian (space complexity)?
  - Các tham số để tính độ phức tạp tính toán:
    - *b* – số nhánh tối đa (maximum branching factor).
    - *d* – độ sâu tìm được nghiệm.
    - *m* – độ sâu tối đa của không gian trạng thái.

Tìm kiếm theo chiều rộng

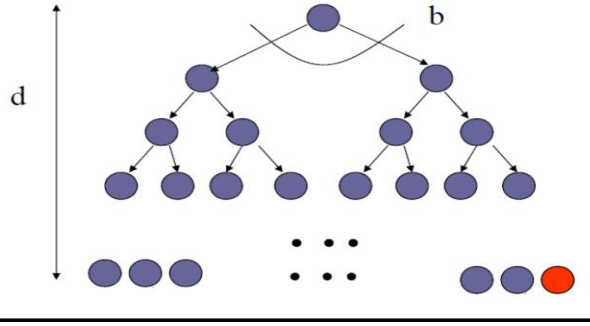
- Độ phức tạp thời gian (time complexity)?
  - Tổng số nút được sinh ra: $b + b^2 + \ldots + b^d$
  - Độ phức tạp thời gian: $O(b^d)$



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
d	2^d (b^d)

Tìm kiếm theo chiều rộng

- Độ phức tạp không gian (space complexity)?
  - Tổng số nút lưu trữ trong *explored*: $1 + b + b^2 + \ldots + b^{d-1}$
  - Tổng số nút lưu trữ trong *frontier*: b^d
  - Tổng số nút lưu trữ: $1 + b + b^2 + \ldots + b^{d-1} + b^d$
  - Độ phức tạp không gian: $O(b^d)$



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
d	2^d (b^d)

Tìm kiếm theo chiều rộng

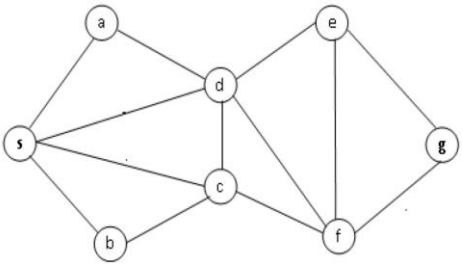
- Ví dụ tính độ phức tạp thời gian và không gian với các giả thiết:
  - Nhân tố nhánh b = 10.
  - Một giây giây sinh ra được 1.000.000 nodes.
  - Một node được lưu trữ là 1000 bytes.
- Thời gian và bộ nhớ khi d = 2, 4, 6, 8,10, 12, 14, 16.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

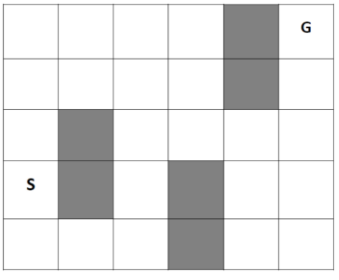
Tìm kiếm theo chiều rộng

- Bài tập

1. Sử dụng thuật toán BFS tìm đường đi từ đỉnh S đến đỉnh G và vẽ cây tìm kiếm.



2. Sử dụng thuật toán BFS tìm đường đi từ đỉnh a đến đỉnh g. Vẽ cây tìm kiếm.



Tìm kiếm với giá cực tiểu

- Khi bài toán có giá thành của tất cả các bước bằng nhau, thuật toán tìm kiếm theo chiều rộng là tối ưu vì nó luôn mở nút "*nông nhất*" chưa được mở.
- Thuật toán tìm kiếm với chi phí cực tiểu nhằm tìm nghiệm tối ưu cho bài toán với mọi hàm giá.
- Định nghĩa hàm giá của nút *n*: $g(n)$ là chiều dài của đường đi từ trạng thái đầu đến nút *n*.
- Chiến lược tìm kiếm: mở rộng nút lá với $g(n)$ bé nhất.

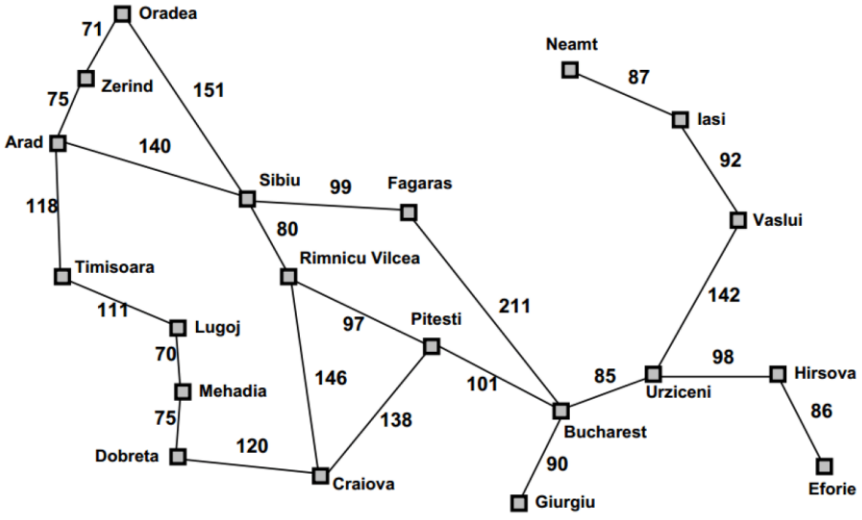
Tìm kiếm với giá cực tiểu

- Thực hiện: mở rộng nút n trong frontier có $g(n)$ là bé nhất.

```
function UNIFORM-COST-SEARCH(problem) returns 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 do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
    if problem.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)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

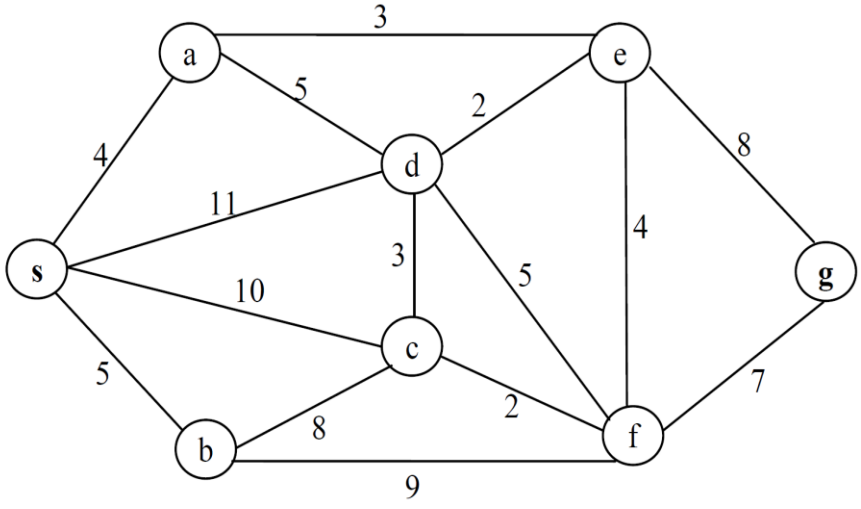
Tìm kiếm với giá cực tiểu

- Ví dụ 1: tìm đường đi ngắn nhất từ Arad→ Bucharest



Tìm kiếm với giá cực tiểu

- Ví dụ 2: tìm đường đi ngắn nhất từ s → g

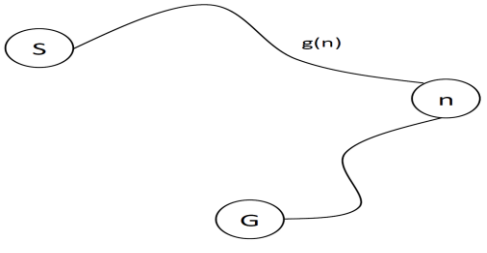


Tìm kiếm với giá cực tiểu

- Hoàn chỉnh (completeness)? Yes
  - Luôn tìm được đường đi.
- Tối ưu (optimality)? Yes
  - Đường đi tìm được là đường ngắn nhất.
- Độ phức tạp thời gian (time complexity)?
  - Tìm tài liệu tham khảo.
- Độ phức tạp không gian (space complexity)?
  - Tìm tài liệu tham khảo.

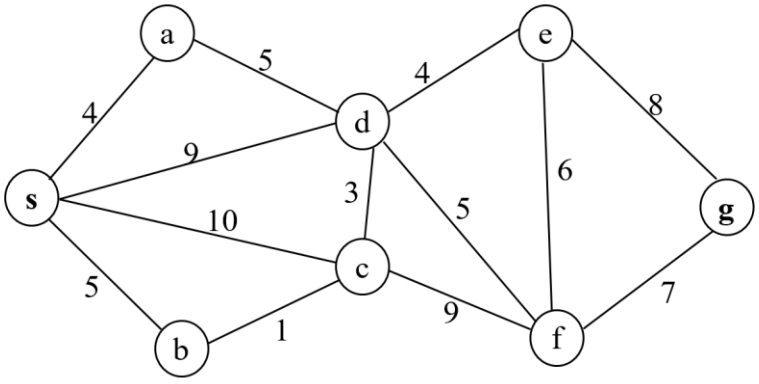
Tìm kiếm với giá cực tiểu

- Nguyên lý tối ưu lập trình động (Dynamic Programming Optimality)
  - Đường ngắn nhất từ trạng thái S đến trạng thái G qua trạng thái n là tổng của đường ngắn nhất đi từ S tới n và đường ngắn nhất đi từ n tới G.
  - Chỉ cần lưu một đường ngắn nhất từ S tới n. Nghĩa là nếu tìm ra một đường mới tới một trạng thái đã xét, ta sẽ chỉ lưu đường ngắn nhất và loại bỏ đường dài hơn.



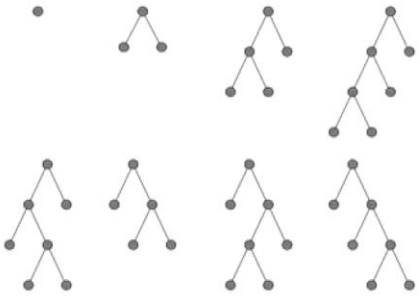
Bài tập

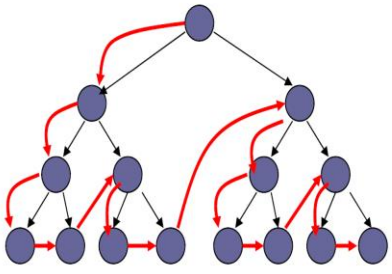
- Tìm đường đi ngắn nhất từ đỉnh s đến đỉnh g.



Tìm kiếm theo chiều sâu

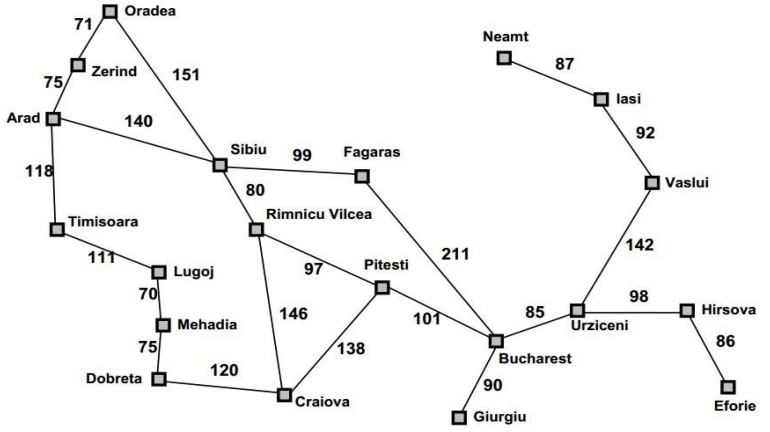
- Nút "sâu nhất" (deepest) trong *frontier* được mở rộng đầu tiên.
- Quay lui (backtrack) khi nút không thể mở tiếp.





Tìm kiếm theo chiều sâu

- Thuật toán giống với tìm kiếm theo chiều rộng, nhưng đặt nút con vào ĐẦU (LIFO) của *frontier*.
- Ví dụ tìm đường đi từ thành phố *Arad* → *Bucharest*

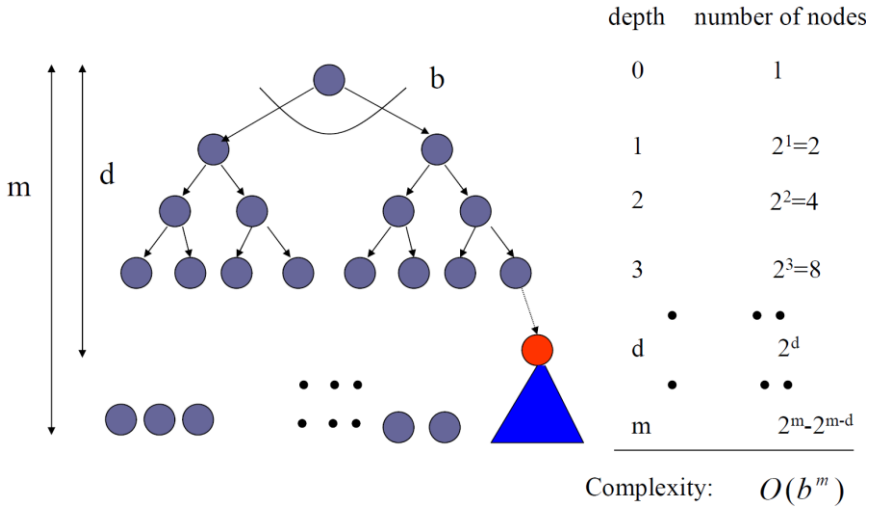


Tìm kiếm theo chiều sâu

- Hoàn chỉnh (completeness)? Yes.
  - Nếu thuật toán thể hiện theo phiên bản TREE-SEARCH, tính hoàn chỉnh là No, vì có thể tạo ra các đường lặp vô hạn.
- Tối ưu (optimality)? No
  - Nghiệm tìm thấy có thể không phải là nghiệm tối ưu (ví dụ không phải là đường ngắn nhất).
- Độ phức tạp thời gian (time complexity)?
- Độ phức tạp không gian (space complexity)?
  - b – số nhánh tối đa (maximum branching factor)
  - d – độ sâu tìm được nghiệm
  - m – độ sâu tối đa của không gian trạng thái

Tìm kiếm theo chiều sâu

- Độ phức tạp thời gian (time complexity)?

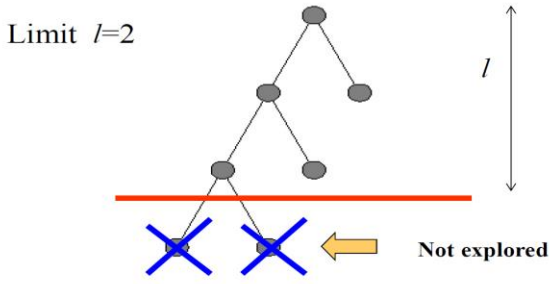


Tìm kiếm theo chiều sâu

- Hoàn chỉnh (completeness)? Yes.
  - Nếu thuật toán thể hiện theo phiên bản TREE-SEARCH, tính hoàn chỉnh là No, vì có thể tạo ra các đường lặp vô hạn.
- Tối ưu (optimality)? No
  - Nghiệm tìm thấy có thể không phải là nghiệm tối ưu (ví dụ không phải là đường ngắn nhất).
- Độ phức tạp thời gian (time complexity)? $O(b^m)$
- Độ phức tạp không gian (space complexity)?
  - Đối với thuật toán dựa trên GRAPH-SEARCH: $O(b^m)$
  - Đối với thuật toán dựa trên TREE-SEARCH: $O(bm)$

Tìm kiếm chiều sâu giới hạn độ sâu

- Trong không gian vô hạn, làm thế nào để loại bỏ tìm kiếm chiều sâu vô hạn trong thuật toán DFS?
- Hạn chế chỉ tìm kiếm tối đa với độ sâu l trong thuật toán DFS.

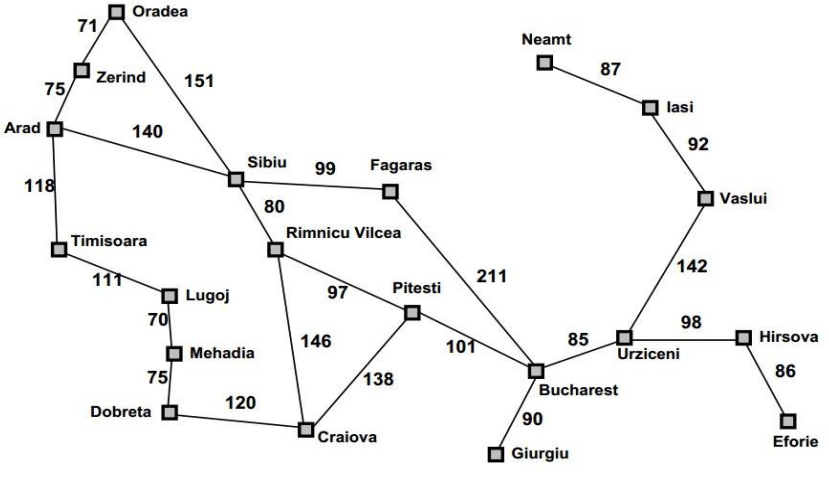


- **Time complexity:** $O(b^l)$
- **Memory complexity:** $O(bl)$

l - is the given limit

Tìm kiếm chiều sâu giới hạn độ sâu

- Ví dụ tìm đường đi từ thành phố Arad đến Bucharest với độ sâu (a) $l = 3$; (b) $l = 5$.



Tìm kiếm chiều sâu giới hạn độ sâu

- Thuật toán đệ quy dựa trên thuật toán TREE-SEARCH

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

function RECURSIVE-DLS(node, problem, limit) returns 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 else return failure
```

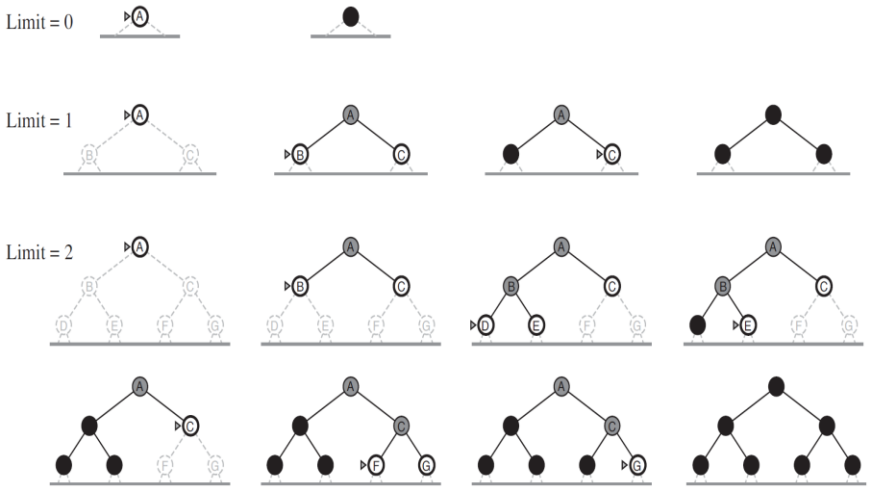
Tìm kiếm lặp sâu dần

- Tên gọi khác là "Iterative deepening search – IDS"
- Dựa trên ý tưởng tìm kiếm giới hạn độ sâu (depth-limited search), nhưng giải quyết khó khăn phải biết trước giới hạn độ sâu.
- Ý tưởng: thử tất cả độ sâu theo thứ tự tăng dần $l = 0$, $l = 1$, $l = 2$,⋯ cho đến khi tìm được nghiệm.
- Thuật toán IDS kết hợp các ưu điểm của DFS và BFS.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result
```

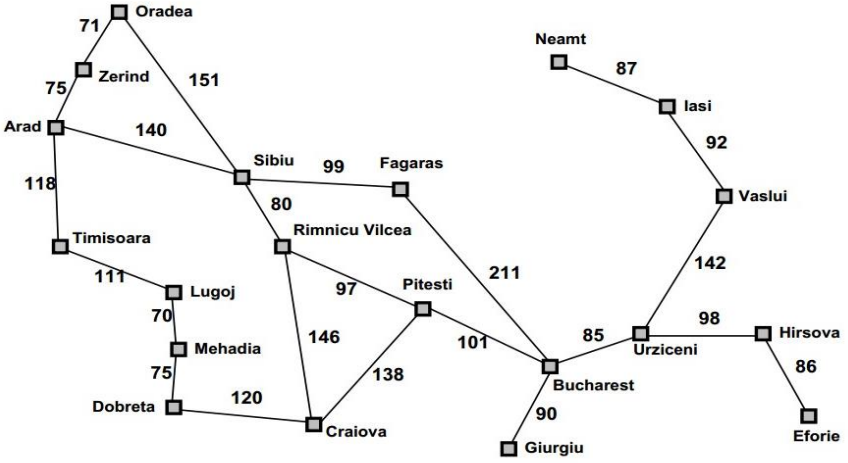
Tìm kiếm lặp sâu dần

- Tăng dần giới hạn độ sâu của thuật toán: l = 0, 1, 2,⋯



Tìm kiếm lặp sâu dần

- Vẽ cây tìm kiếm tìm đường đi từ Arad → Bucharest với l = 1 và l = 2

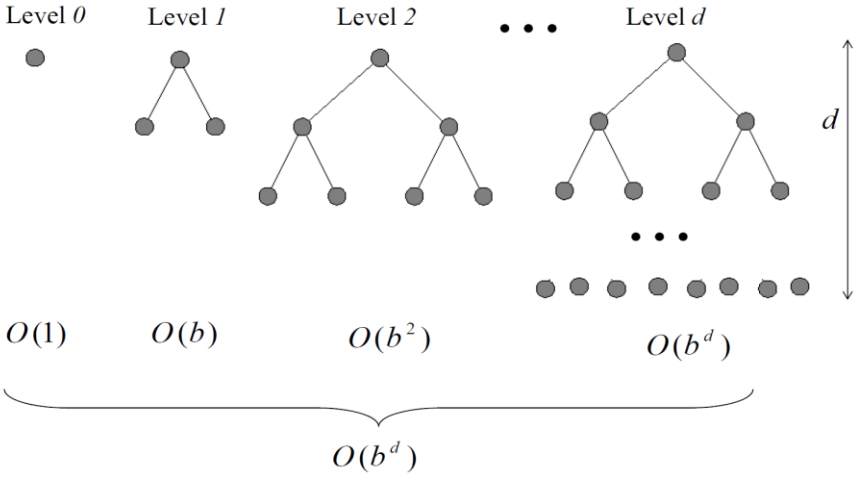


Tìm kiếm lặp sâu dần

- Hoàn chỉnh (completeness)? Yes
  - Thuật toán tìm được nghiệm nếu bài toán có nghiệm (giống như BFS khi độ sâu tăng dần lên 1).
- Tối ưu (optimality)? Yes
  - Nghiệm tìm được là tối ưu (giống như BFS).
- Độ phức tạp thời gian (time complexity)?
- Độ phức tạp không gian (space complexity)?

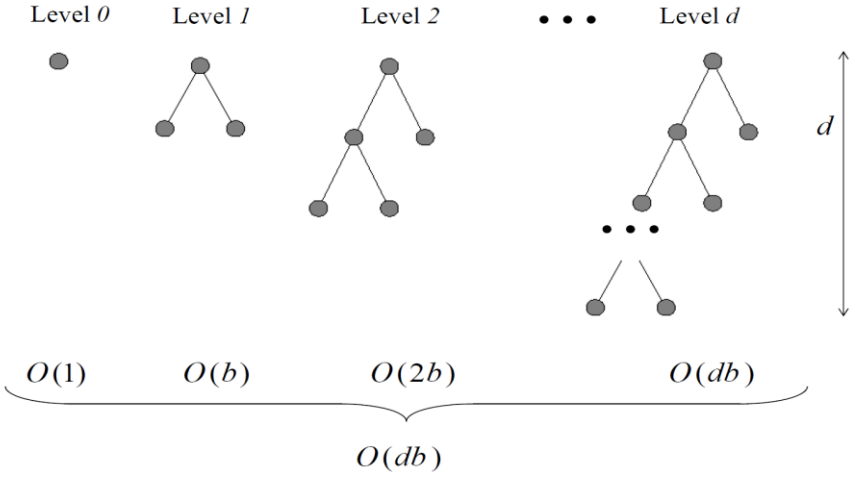
Tìm kiếm lặp sâu dần

- Độ phức tạp thời gian (time complexity)?
  - Giống thuật toán tìm kiếm theo chiều rộng.



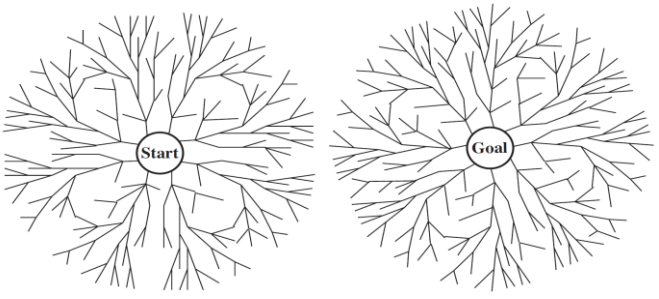
Tìm kiếm lặp sâu dần

- Độ phức tạp không gian (space complexity)?
  - Giống thuật toán tìm kiếm theo chiều sâu trên cây.



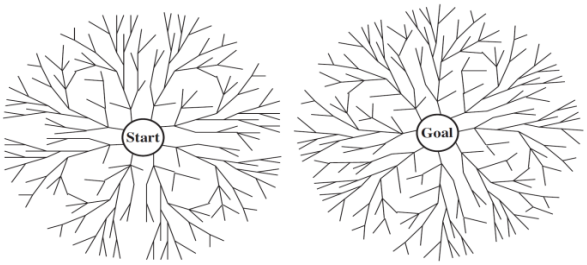
Tìm kiếm từ hai hướng

- Tìm kiếm từ hai hướng - bidirectional search
- Trong một số bài toán, chúng ta thường tìm nghiệm từ trạng thái ban đầu đến một trạng thái đích duy nhất.
- Ý tưởng: đồng thời tìm kiếm từ trạng thái ban đầu và từ trạng thái đích.



Tìm kiếm từ hai hướng

- Tại sao tìm kiếm từ 2 hướng? lợi ích của tìm kiếm từ 2 hướng?
  - Cắt đôi không gian tìm kiếm, do đó độ phức tạp thời gian và không gian: $O(b^d/2)$
- Điều cần thiết là hợp nhất các nghiệm.
  - Nếu một trạng thái được chọn từ 2 phía thì nghiệm bài toán là hợp nhất nghiệm của 2 phía tìm kiếm.



So sánh các thuật toán tìm kiếm

- Xem mục 3.4.7, trang 91, tài liệu [1].

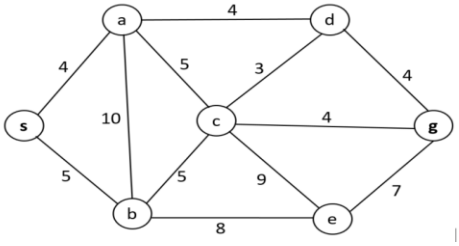
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Bài tập

- Sử dụng các thuật toán BFS, UCS, DFS tìm đường đi từ đỉnh s đến đỉnh g cho các đồ thị sau:

a) Đồ thị 1



b) Đồ thị 2