KHOA CÔNG NGHỆ THÔNG TIN
Bộ môn Trí tuệ nhân tạo

BÀI GIẢNG
Trí tuệ nhân tạo

GV: NGUYỄN ĐẮC HIẾU
Email: dachieu@tlu.edu.vn
Tel: 038.471.2262

Tài liệu tham khảo

1) Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 1994

2) Đinh Mạnh Tường, *Trí tuệ nhân tạo*, Nhà xuất bản khoa học kỹ thuật, 2002

3) Nguyễn Thanh Thủy, *Trí tuệ nhân tạo*, Nhà xuất bản giáo dục, 1997

4) TS. Nguyễn Văn Nam, *Bài giảng điện tử*, Trường ĐH Thuỷ Lợi

Nội dung

- Chương 1: Giới thiệu chung
- Chương 2: Giải quyết vấn đề bằng tìm kiếm
- Chương 3: Logic và suy diễn
- Chương 4: Biểu diễn tri thức
- Chương 5: Biểu diễn tri thức không chắc chắn
- Chương 6: Học máy

Chương 1

Giới thiệu chung

Giới thiệu về trí tuệ nhân tạo

- Khái niệm về trí tuệ nhân tạo
- Lịch sử phát triển của trí tuệ nhân tạo
- Các lĩnh vực ứng dụng

Trí tuệ nhân tạo là gì

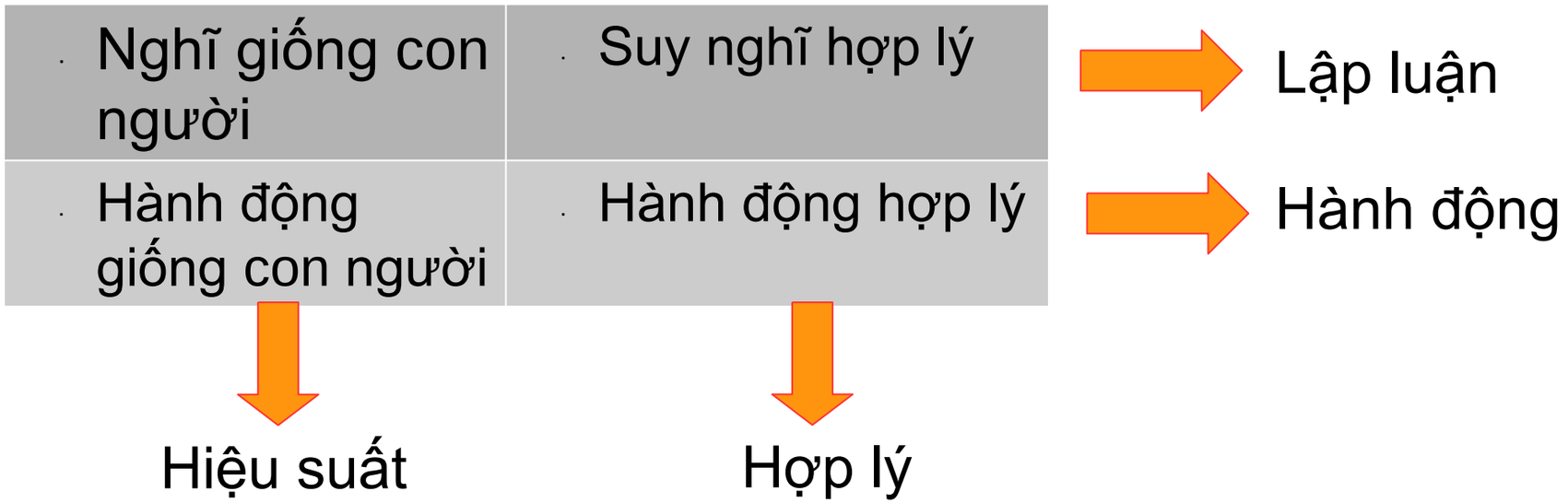
- Trí tuệ là gì?
  - Khả năng tính toán để đạt được mục tiêu
  - Liên quan tới các nhiệm vụ đòi hỏi quá trình xử lý trí óc cao
    - Giải quyết bài toán; nhận dạng mẫu; phân loại; học; lập luận; suy diễn; xử lý ngôn ngữ, tri thức,...

Trí tuệ nhân tạo là gì

- Trí tuệ nhân tạo là gì?
  - Artificial Intelligence (AI)
  - Là trí tuệ được biểu diễn bởi bất cứ một hệ thống nhân tạo nào
  - Là ngành khoa học kĩ thuật làm cho máy móc thông minh
    - Deep Blue đánh bại đại kiện tướng cờ vua Garry Kasparov 1997 (3 hòa, 2 thắng, 1 thua)

Định nghĩa AI

- 4 nhóm định nghĩa, AI là ngành khoa học làm cho máy:



Định nghĩa AI

- (1) Nghĩ giống con người
  - Trí tuệ nhân tạo giúp tạo ra máy tính có khả năng **suy nghĩ**...máy tính có **trí tuệ** theo đầy đủ nghĩa của từ này (Haugeland, 1985)
  - Tự động hoá các hoạt động **phù hợp với suy nghĩ** con người như các hoạt động ra quyết định, giải bài toán, học, ... (Bellman 1978)
- (2) Suy nghĩ hợp lý
  - Sự nghiên cứu các **hoạt động trí não** thông qua việc sử dụng các mô hình tính toán (Charniak and McDormott, 1985)
  - Sự nghiên cứu các tính toán để máy **có thể nhận thức, lập luận** và hành động (Winston 1992)

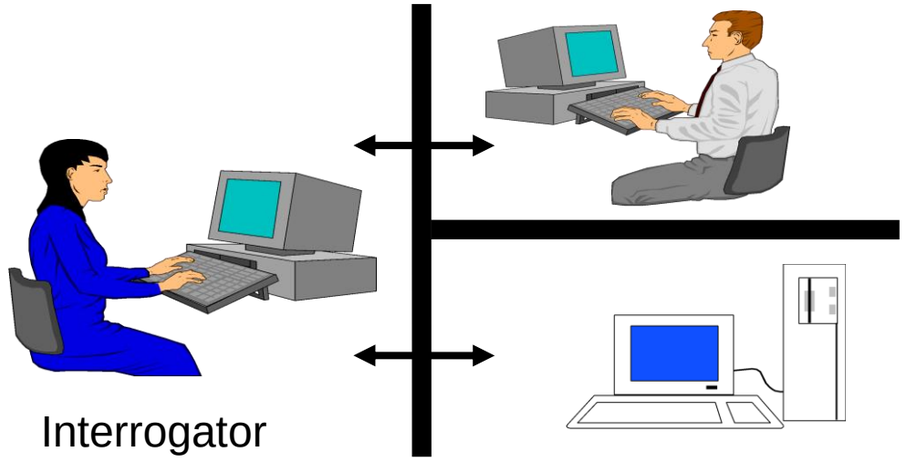
Định nghĩa AI

- (3) Hành động giống con người
  - Nghệ thuật tạo ra các máy **thực hiện** các chức năng đòi hỏi sự thông minh khi được thực hiện bởi con người (Kurzweil, 1990)
  - Sự nghiên cứu để làm cho máy tính **làm được** những việc mà con người còn làm tốt hơn máy tính (Rich và Knight, 1991)

Định nghĩa AI

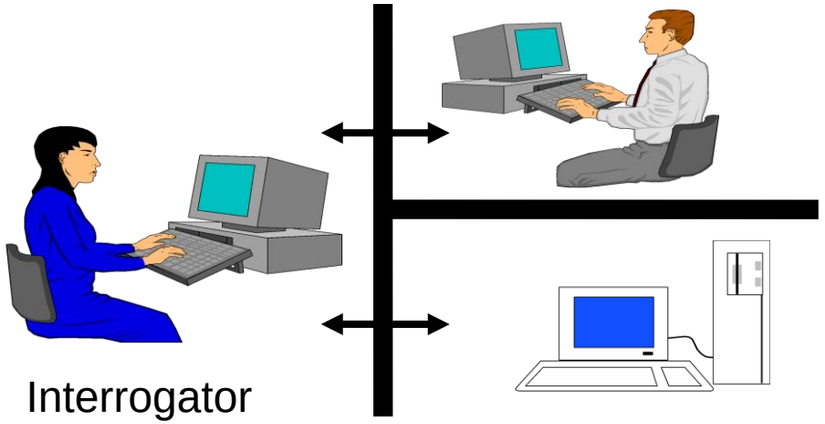
- (4) Hành động hợp lý
  - Lĩnh vực nghiên cứu tìm cách giải quyết và mô phỏng các **hành vi thông minh** trong thuật ngữ các quá trình tính toán (Schalkoff, 1990)
  - Một nhánh của khoa học máy tính liên quan tới sự tự động hoá các **hành vi thông minh** (Luger and Stubblefield, 1993)
  - Nghiên cứu thiết kế các **tác nhân thông minh** (Pulle, Mackworth and Goebel, 1998)
  - TTNT nghiên cứu các **hành vi thông minh** mô phỏng trong các vật thể nhân tạo (Nilsson 1998)

Hành động giống con người



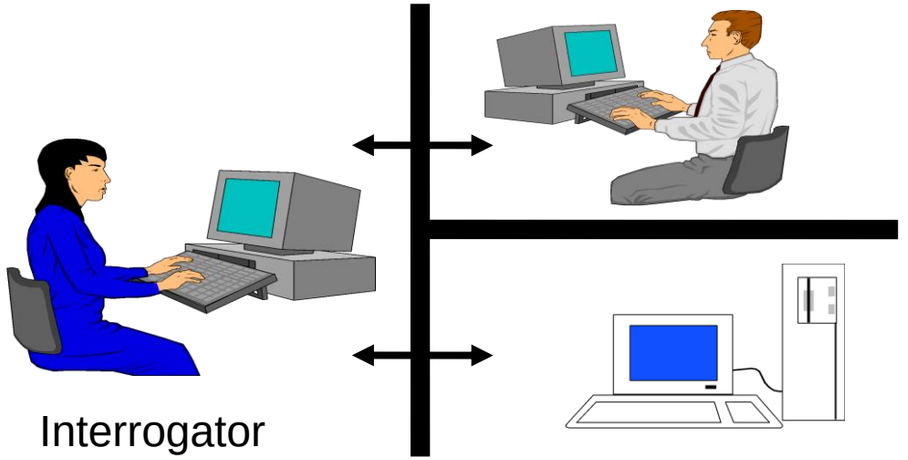
- Turing test
  - Liệu máy tính có thể suy nghĩ? → Máy tính có thể hành động thông minh (như con người)
  - Trò chơi bắt chước người để kiểm chứng hành động thông minh của máy tính

Hành động giống con người



- Turing test
  - Người thẩm vấn (interrogator) cần nhận biết mình đang chơi với người hay máy tính bằng cách đưa ra các câu hỏi và đánh giá các câu trả lời nhận được
  - Người chơi phải trả lời trung thực còn máy tính có thể nói dối để đánh lừa
  - Nếu không phát hiện được thì coi như máy tính thông minh như con người

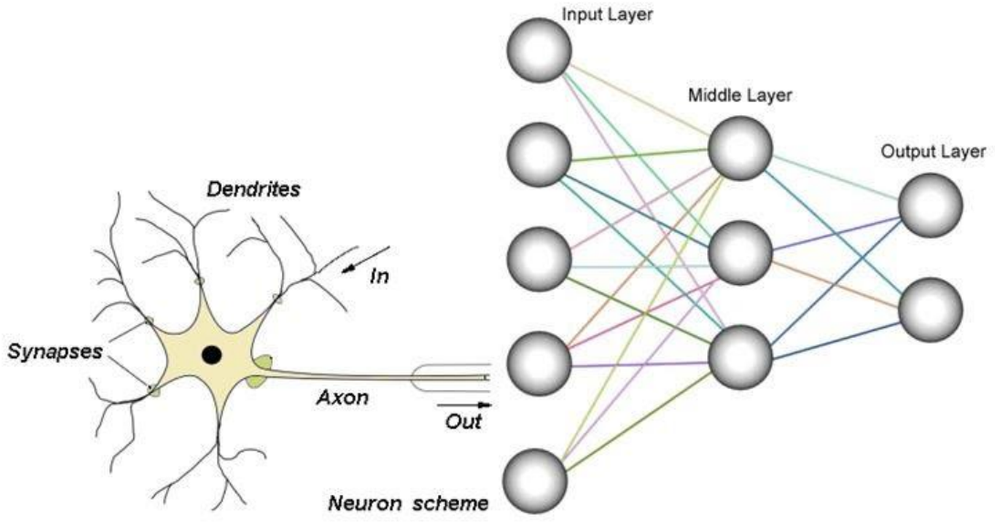
Hành động giống con người



- Turing test
  - Ưu điểm:
    - Loại trừ định kiến thiên vị của người thẩm vấn
  - Nhược điểm
    - Chủ quan, phụ thuộc vào chất lượng câu hỏi của người hỏi

Suy nghĩ giống người

- Tìm hiểu cách suy nghĩ của con người
  - Xử lý ngôn ngữ, nghĩ, học, lập luận,... được thực hiện như thế nào
- Tạo các mô hình tính toán có cách thức suy nghĩ của con người, bao gồm 1 chuỗi các bước lập luận tương tự khi con người giải quyết cùng vấn đề
- Ví dụ: mạng nơron mô phỏng bộ não người bằng cách tạo ra các nơron nhân tạo, xây dựng cơ chế tính toán và học cho các nơron, dùng để phân lớp, nhận dạng, tính toán mang tính dự báo...



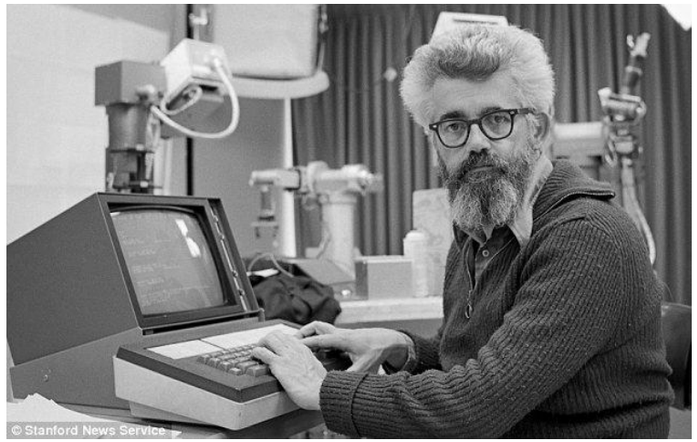
Suy nghĩ hợp lý

- Tập trung vào các cơ chế suy diễn mà có thể chính xác hoặc cho kết quả tối ưu
- Phát triển các hệ thống biểu diễn cho phép suy diễn kiểu như: *“Tất cả mọi người bị viêm răng đều đau răng. An bị viêm răng. Do đó, An bị đau răng ”*
- Hình thức hóa quá trình lập luận dưới dạng hệ thống gồm các luật và thủ tục suy diễn
- Ví du: hệ chuyên gia (chuẩn đoán bệnh, xem tử vi, ...), hệ trợ giúp quyết định, ...
- Vấn đề: không phải vấn đề nào cũng có thể giải quyết bằng lập luận và suy diễn

Hành động hợp lý

- Thực hiện tốt công việc cần làm, cố gắng đạt được mục tiêu cao nhất (không nhất thiết tối ưu) từ những thông tin được cung cấp
- Thực thi các hành vi thông minh (là các hành vi làm tăng cơ hội đạt được các mục đích đề ra)
- Có thể tạm chấp nhận lập luận không hoàn hảo nếu thực hiện được công việc đề ra
- Ví dụ: multi-agent systems, robotics, ...

Lịch sử phát triển của AI



- Những năm 1950: khai sinh ra AI
  - 1950: Turing đề cập tới khái niệm về trí tuệ nhân tạo và đưa ra thí nghiệm Turing
  - 1956: thuật ngữ “trí tuệ nhân tạo” được chính thức đưa ra bởi John McCarthy và được công nhận tại hội nghị Dartmouth
  - Đầu 1950s, các chương trình AI đầu tiên
    - Chương trình chơi cờ của Samuel
    - Chương trình lý luận logic của Newell & Simon
    - Chương trình chứng minh các định lý hình học của Gelernter 1959
  - 1958: John McCarthy cho ra đời ngôn ngữ lập trình trí tuệ nhân tạo LISP

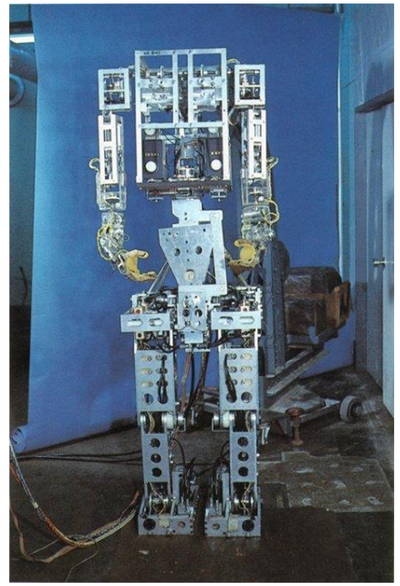
Lịch sử phát triển của AI

- Những năm 1960: khởi đầu của các đề án nghiên cứu AI
  - Bộ giải quyết bài toán tổng quát GPS (General Problem Solver- Newell và Simon)
  - 1965: chương trình chuyên gia phân tích tâm lý ELIZA
  - 1968: chương trình phân tích hình học (Tom Evans)

Lịch sử phát triển của AI

- Lịch sử AI trong những năm 1970

  - 1970: WABOT-1, robot hình nhân đầu tiên, được chế tạo ở Nhật Bản tại Đại học Waseda. Các tính năng của nó bao gồm tay chân có thể cử động, khả năng nhìn và khả năng trò chuyện.
  - 1973: James Lighthill đã tuyên bố: “không có khám phá tạo ra tác động lớn nào trong lĩnh vực này cho đến nay mà sau đó được hứa hẹn” dẫn đến việc giảm đáng kể hỗ trợ nghiên cứu AI từ chính phủ Anh.



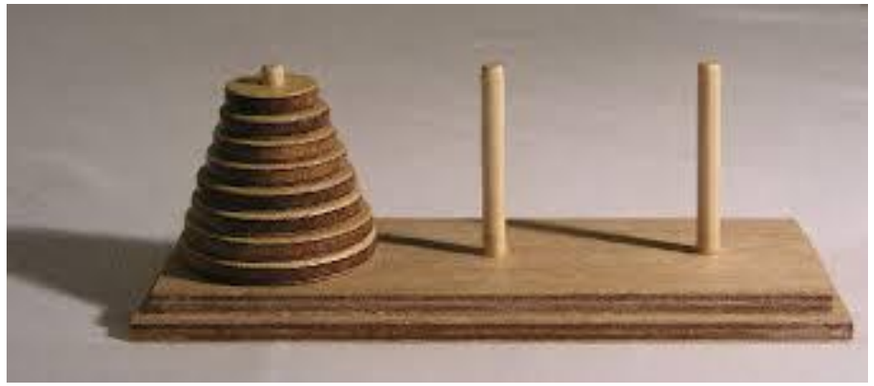
Lịch sử phát triển của AI

- Những năm 1980: AI đi vào vào các ngành kinh tế (AI thương mại) và công nghiệp (máy giặt, máy ảnh)
- 1986: sự phát triển trở lại của các ứng dụng mạng neuron, hệ chuyên gia
- 1995: xuất hiện hệ đa tác tử
- 1997: Deep Blue, máy tính chơi cờ do IBM phát triển trở thành hệ thống đầu tiên chiến thắng trò chơi cờ vua khi đấu với đương kim vô địch thế giới.
- 2016: AlphaGo của Google DeepMind đánh bại nhà vô địch thế giới cờ vây Lee Sedol. Sự phức tạp của trò chơi Trung Quốc cổ đại được coi là một trở ngại lớn để giải tỏa trong AI.

Lĩnh vực ứng dụng của AI

Game playing

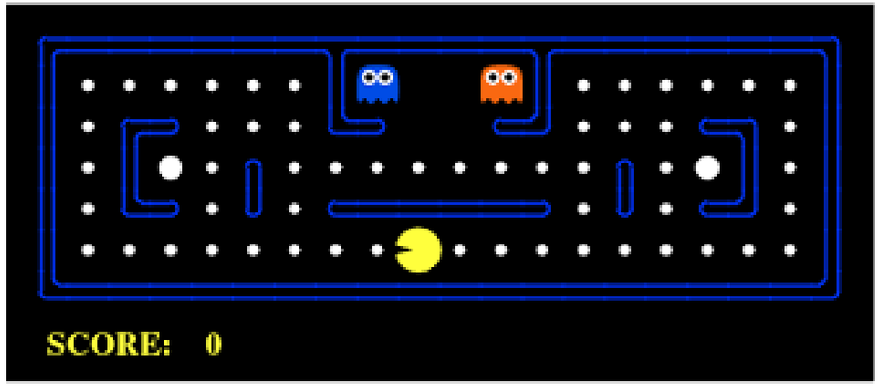
Chess playing



Hanoi tower

1	2	3
8		4
7	6	5

8-puzzle



Pacma

Muti-agent based games

Driver

Football

Haft-life

Sim city

Simulation

Crowd Simulation
in Airports

Flight simulator

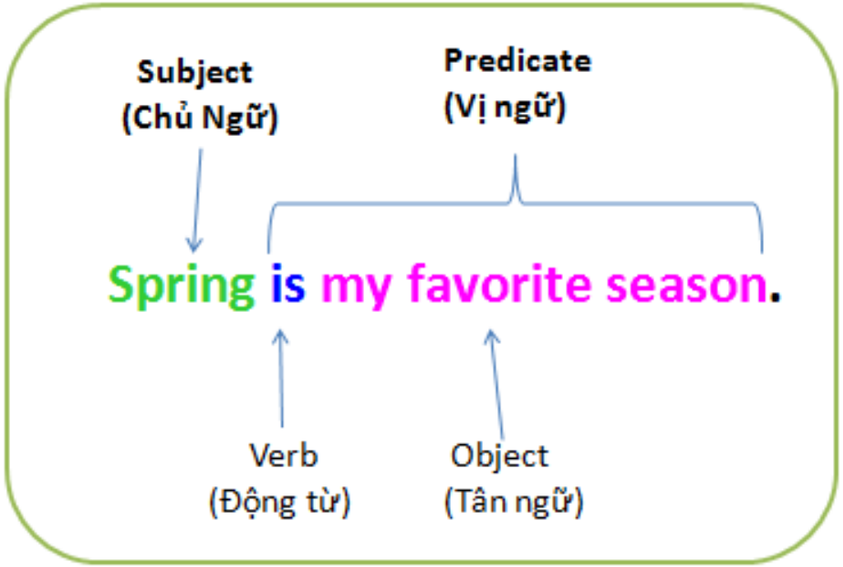
Robotics

Dog robot

Kiva robots in Amazon

Self-driving cars

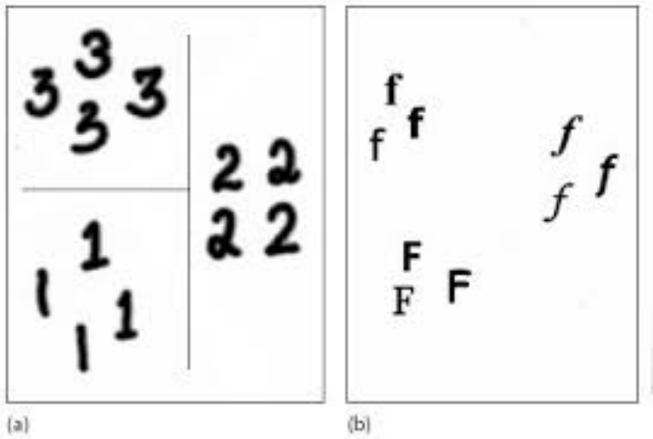
Language/Speech processing

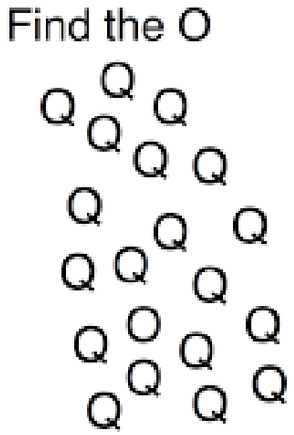


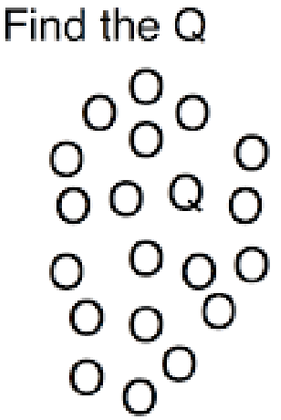
Language processing

Speech recoginition

Pattern recognition



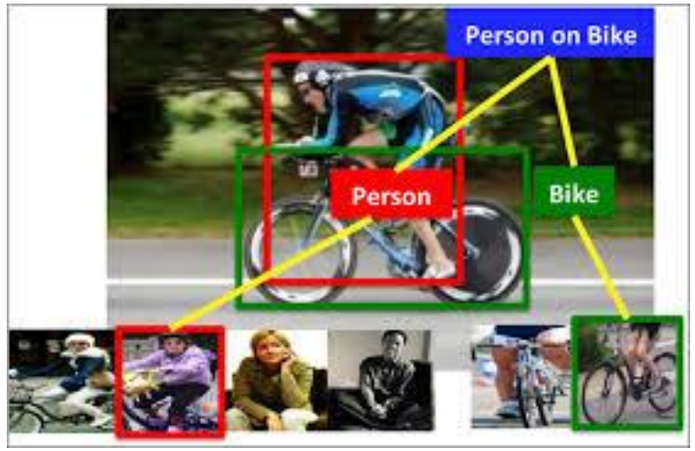


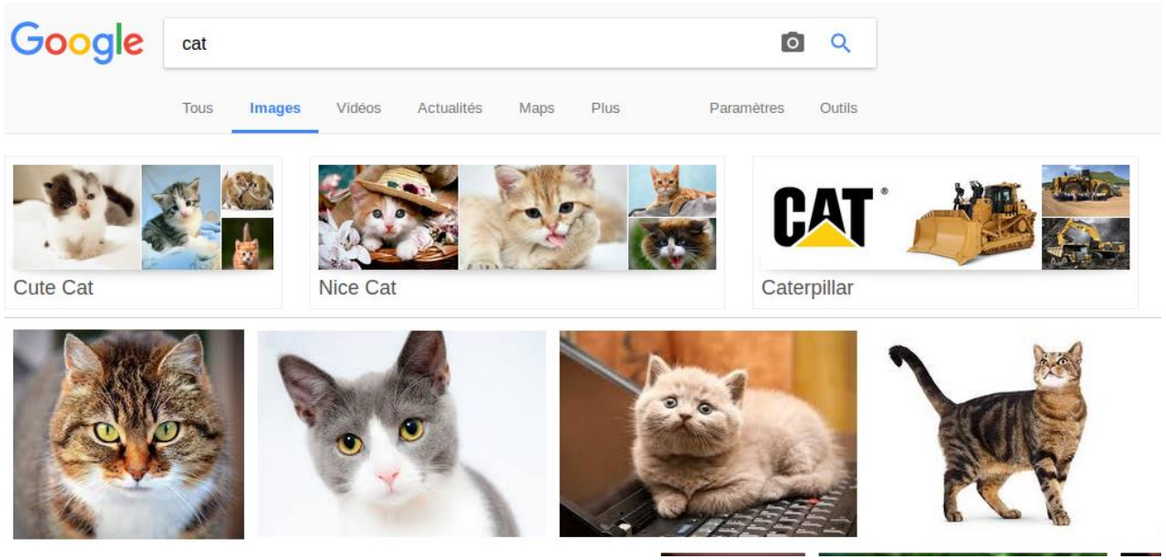


Text recognition

Facebook facial recognition

Computer vision



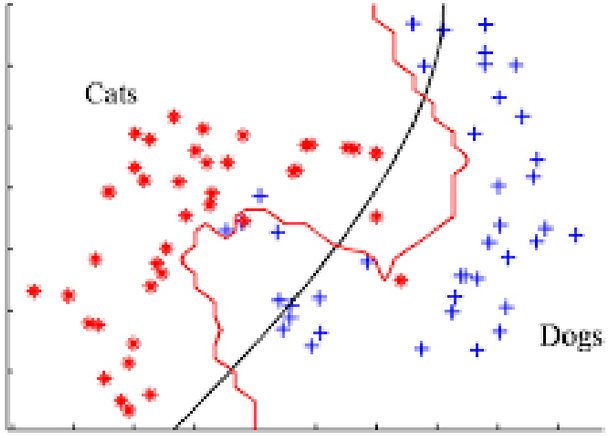


Expert systems

MYCIN expert system

Decision support system

Machine learning



Phân lớp

Dự báo

Phân loại thư rác

Phát hiện thẻ tín dụng giả

Chương 2

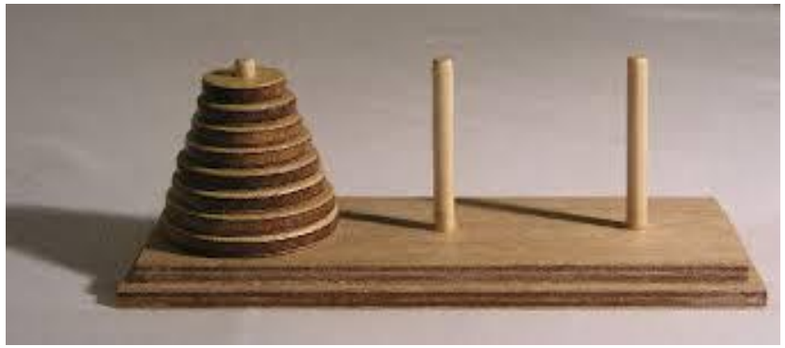
Giải quyết vấn đề bằng tìm kiếm

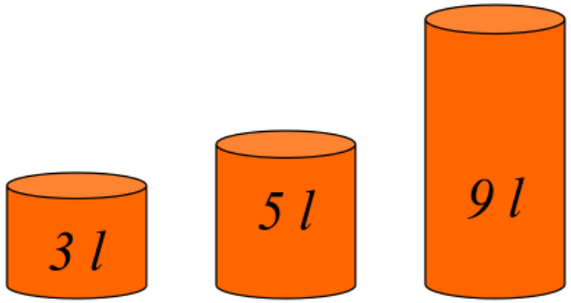
Chương 2: Giải quyết vấn đề bằng tìm kiếm

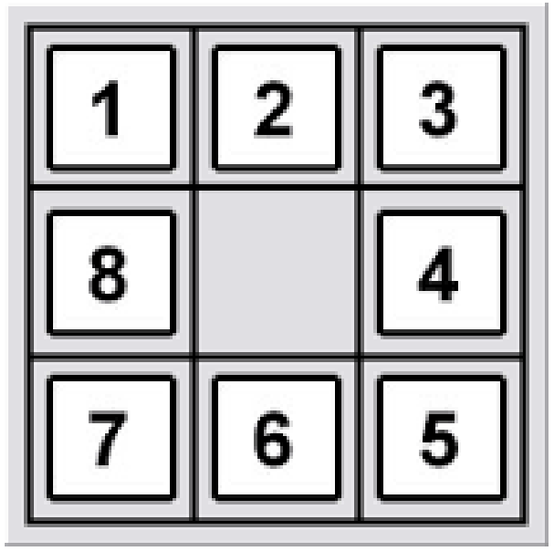
- Các trò chơi trên máy tính
- Biểu diễn bài toán bằng không gian trạng thái
- Giải quyết bài toán bằng tìm kiếm
- Tìm kiếm mù
- Tìm kiếm kinh nghiệm
- Tìm kiếm tối ưu
- Tìm kiếm có đối thủ

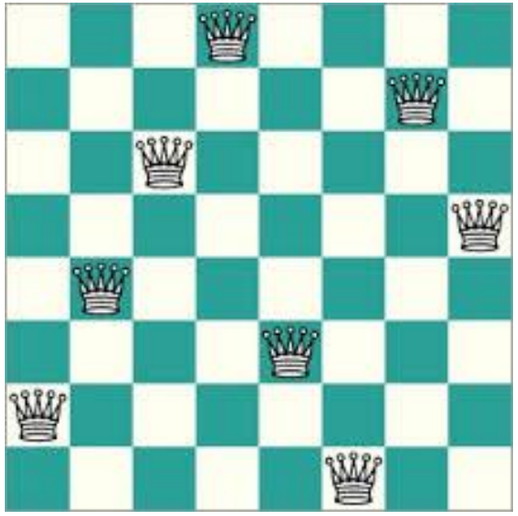
2.1 Các trò chơi trên máy tính

Các trò chơi trí tuệ



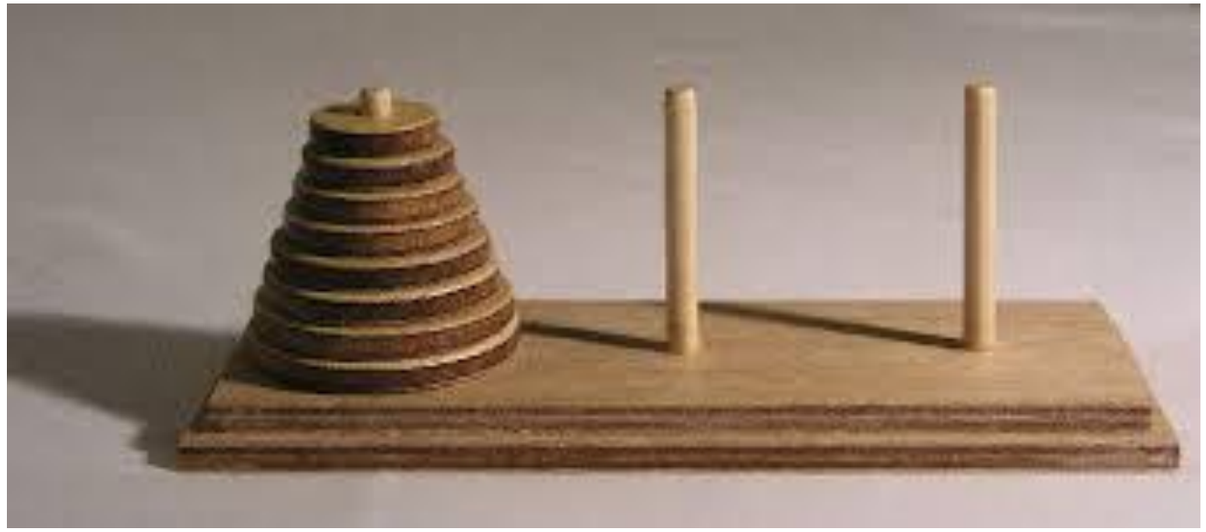






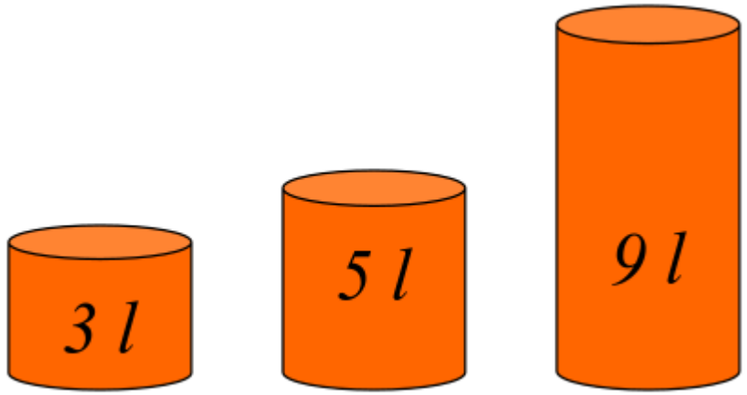
Bài toán tháp Hà Nội

- Cho n cọc và m đĩa ban đầu nằm trên cùng 1 cọc. Cần chuyển toàn bộ các đĩa sang một cọc đích.
- Điều kiện:
  - Mỗi bước chỉ nhấc 1 đĩa.
  - Đĩa nhỏ luôn ở trên đĩa to.



Bài toán rót nước

- Cho 3 bình dung tích 3 lít, 5 lít, 9 lít. Cần đong 7 lít
- Tổng quát: cho n bình dung tích m1,m2,...mn. Cần đong được k lít.



Phạm Hoàn Thọ, Phạm Thị Lê Anh, Giáo trình trí tuệ nhân tạo, 2011

Bài toán 8 con hậu

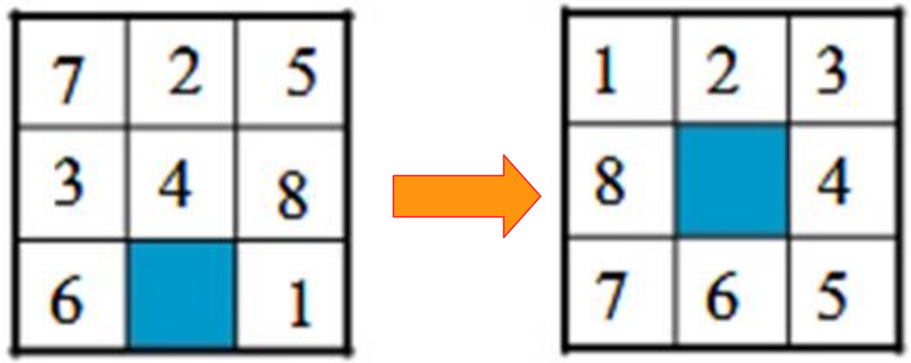
- Đặt 8 con hậu vào bàn cờ 8*8
- Tổng quát: đặt n con hậu vào bàn cờ n*n
- Điều kiện: không có 2 con hậu nào cùng hàng, cùng cột, cùng đường chéo

Bài toán cờ tic-tac-toe

- 2 người chơi lần lượt điền kí hiệu × hoặc ○ của mình lên bàn cờ 3*3
- Người thắng tạo được dãy 3 kí hiệu (thẳng hàng, cột hay chéo) của mình trước

Bài toán 8-puzzle

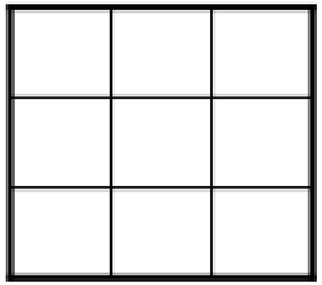
- Cho 1 bảng 3×3 trong đó có 8 ô được đánh số từ 1 đến 8 và 1 ô trống
- Cần dịch chuyển liên tiếp ô trống sang các ô bên cạnh để dẫn đến 1 bàn cờ mong muốn cho trước.



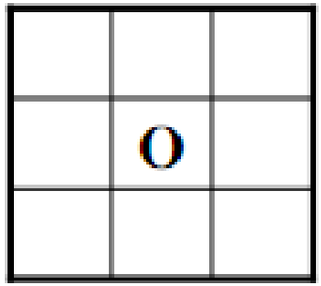
2.2 Biểu diễn bài toán trong không gian trạng thái

Trạng thái

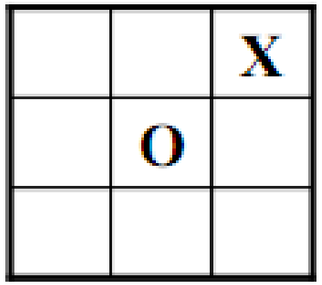
- Trạng thái:
  - Biểu diễn 1 bước nào đó của bài toán.
  - Cách mã hóa bài toán trong máy tính.
- Ví dụ:
  - Tic-tac-toe, 8 con hậu, puzze: tình trạng bàn cờ
  - Tháp Hà Nội: vị trí cọc của của các đĩa



Trạng thái



Trạng thái

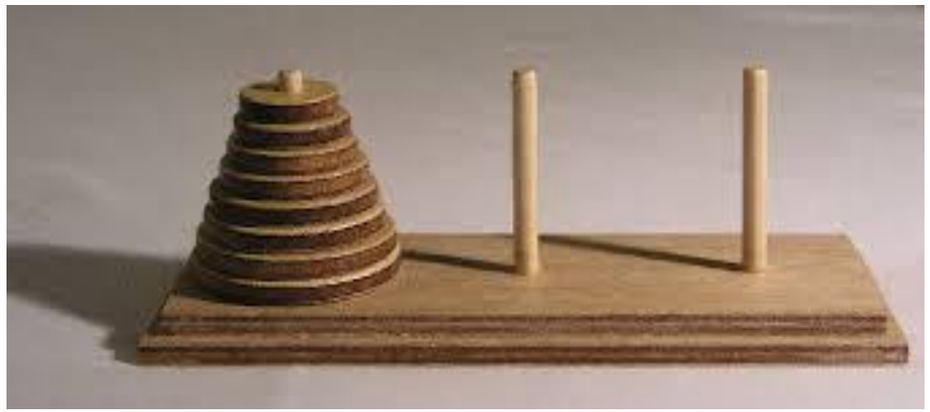


Trạng thái

Trạng thái

Bài toán tháp Hà Nội (3 cọc, 3 đĩa)

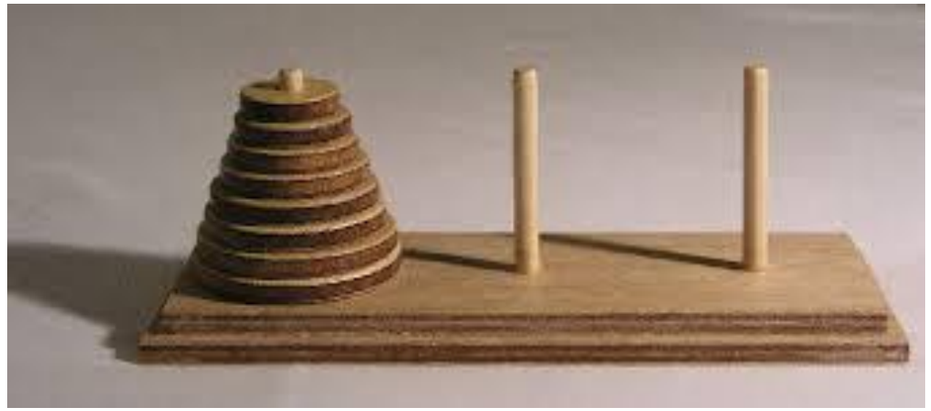
- Trạng thái: vị trí các đĩa
- Biểu diễn: (a,b,c)
  - a,b,c: lần lượt vị trí cọc của đĩa to nhất, thứ 2 và thứ 3.
  - Ví dụ (3,1,2):
    - đĩa to nhất ở cọc 3
    - đĩa to thứ 2 ở cọc 1
    - đia to thứ 3 ở cọc 2
- Ban đầu: (1,1,1)
- Đích: (3,3,3)



Trạng thái

Bài toán tháp Hà Nội (n cọc, m đĩa)

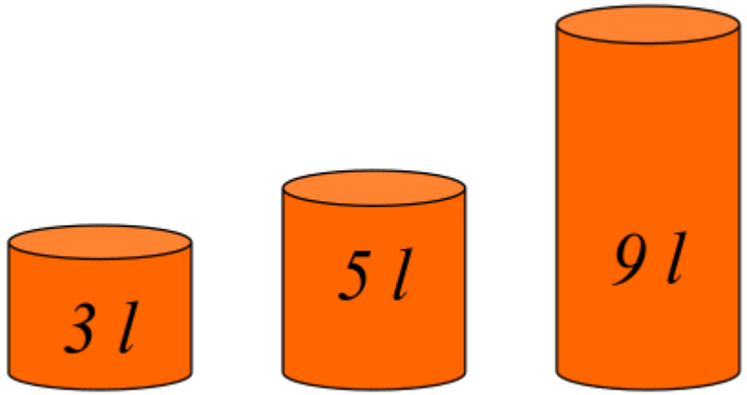
- Trạng thái: vị trí các đĩa
- Biểu diễn: (a1, a2, …, am)
  - ai: vị trí cọc của đĩa to thứ i
  - 1 ≤ ai ≤ n
- Ban đầu: (1,1,...1)
- Kết thúc: (n,n,...n)



Trạng thái

Bài toán rót nước (cụ thể)

- Cho 3 bình dung tích 3 lít, 5 lít, 9 lít. Cần đong 7 lít
- Trạng thái: số lượng nước chứa trong mỗi bình
- Biểu diễn (a1,a2,a3)
  - ai: số lít nước đang chứa trong bình thứ i
- Ban đầu: (0,0,0)
- Đích: (-,-,7)



Phạm Hoàn Thọ, Phạm Thị Lê Anh, Giáo trình trí tuệ nhân tạo, 2011

Trạng thái

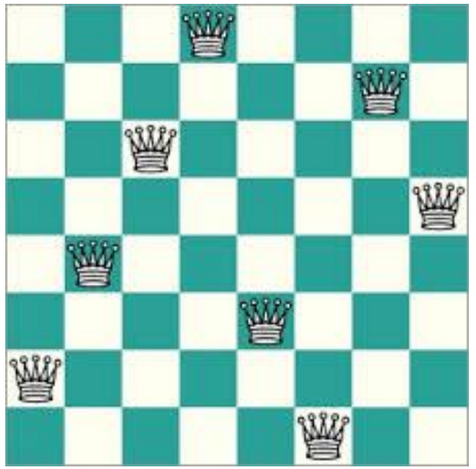
Bài toán rót nước (tổng quát)

- Cho n bình dung tích m1,m2,...mn. Cần đong được k lít.
- Trạng thái: số lượng nước chứa trong mỗi bình
- Biểu diễn: (a1,a2, …, an)
  - 0 ≤ ai ≤ mi là số lít nước đang chứ trong bình thứ i
- Ban đầu: (0,0,…,0)
- Đích: (k,-,…,-) hoặc (-,k,...,-), ..., hoặc (-,-,...,k)

Trạng thái

Bài toán n hậu

- Trạng thái: vị trí các con hậu trên bàn cờ
- Biểu diễn
- Cách 1: (a1,a2,...an)
  - 1 ≤ ai ≤ n là vị trí hàng của con hậu ở cột I
  - Ví dụ: (7,5,3,1,6,8,2,4)
  - Ban đầu: bất kì tình trạng sắp xếp nào
  - Đích: trạng thái thỏa mãn điều kiện ràng buộc



Trạng thái

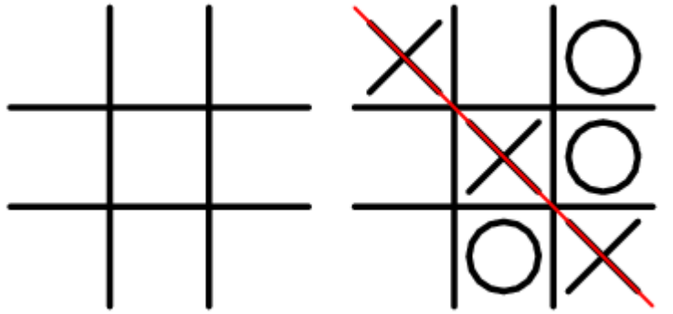
Bài toán n hậu

- Trạng thái: tình trạng của các ô trong bàn cờ
- Biểu diễn
- Cách 2: (a1,a2,...an*n)
  - ai là tình trạng ô thứ i
    - ai = true(T)/false(F) khi ô thứ i có hậu/trống
    - Ví dụ: (F,F,F,**T**,F,F,F,F,
  - …,
  - F,F,F,F,F,**T**,F,F), ...
  - Ban đầu (F,…,F)
  - Đích: trạng thái thỏa mãn điều kiện ràng buộc

Trạng thái

Tic-tac-toe

- Trạng thái: nội dung bàn cờ
- Biểu diễn: (a1,a2,…,a9)
  - ai: tình trạng của ô thứ i
  - ai=0,1 hoặc 2 khi ô thứ i lần lượt trống, đánh dấu bởi ○ hoặc ×
- Ban đầu: (0,0,…,0)
- Đích: (2,0,1,0,2,1,0,1,2), ...



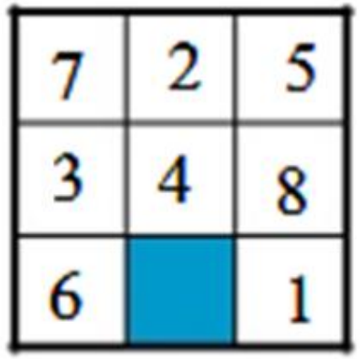
Trạng thái đầu

Trạng thái đích

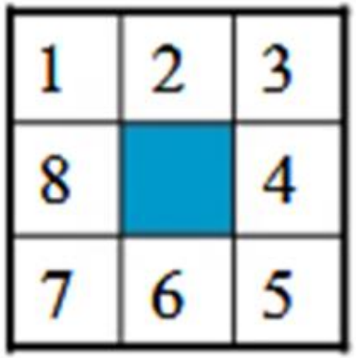
Trạng thái

8-puzze

- Trạng thái: nội dung bảng
- Biểu diễn: (a1,a2,…,a9)
  - ai: nội dung điền vào ô thứ i
  - ai thuộc [0..8] trong đó 0 biểu diễn trường hợp ô trống
- Ban đầu: (7,2,5,3,4,8,6,0,1)
- Đích: (1,2,3,8,0,4,7,6,5)



Trạng thái đầu

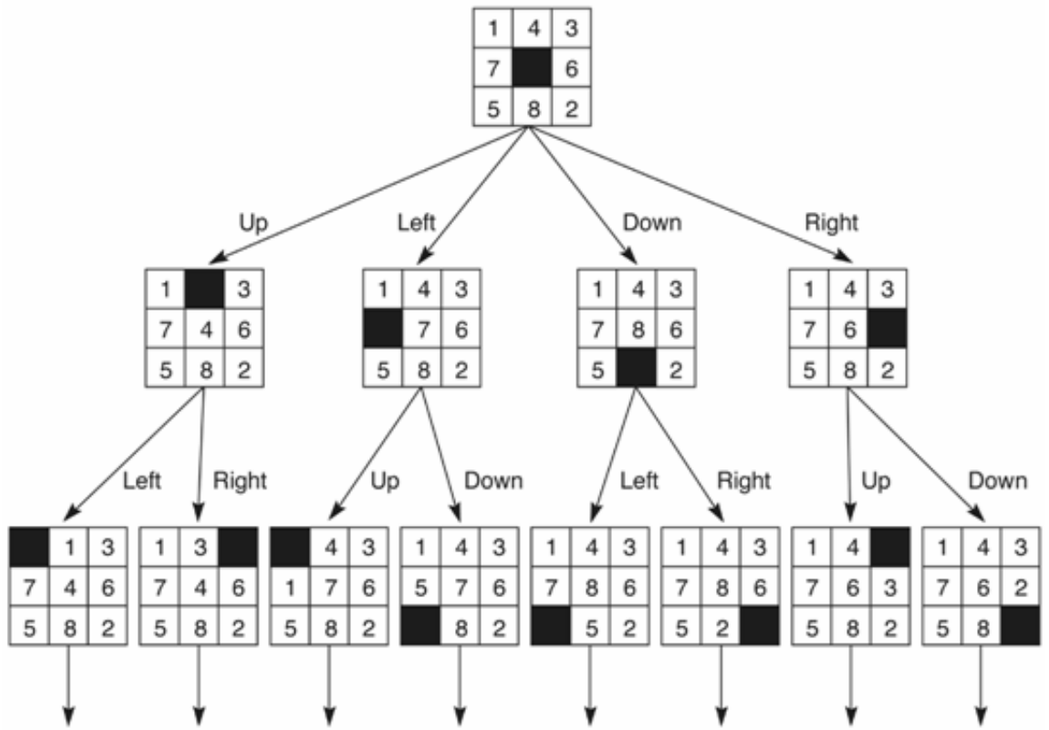


Trạng thái đích

Không gian trạng thái (KGTT)

Là tổ hợp tất cả các trạng thái có thể của bài toán, biểu diễn dưới dạng đồ thị có hướng trong đó:

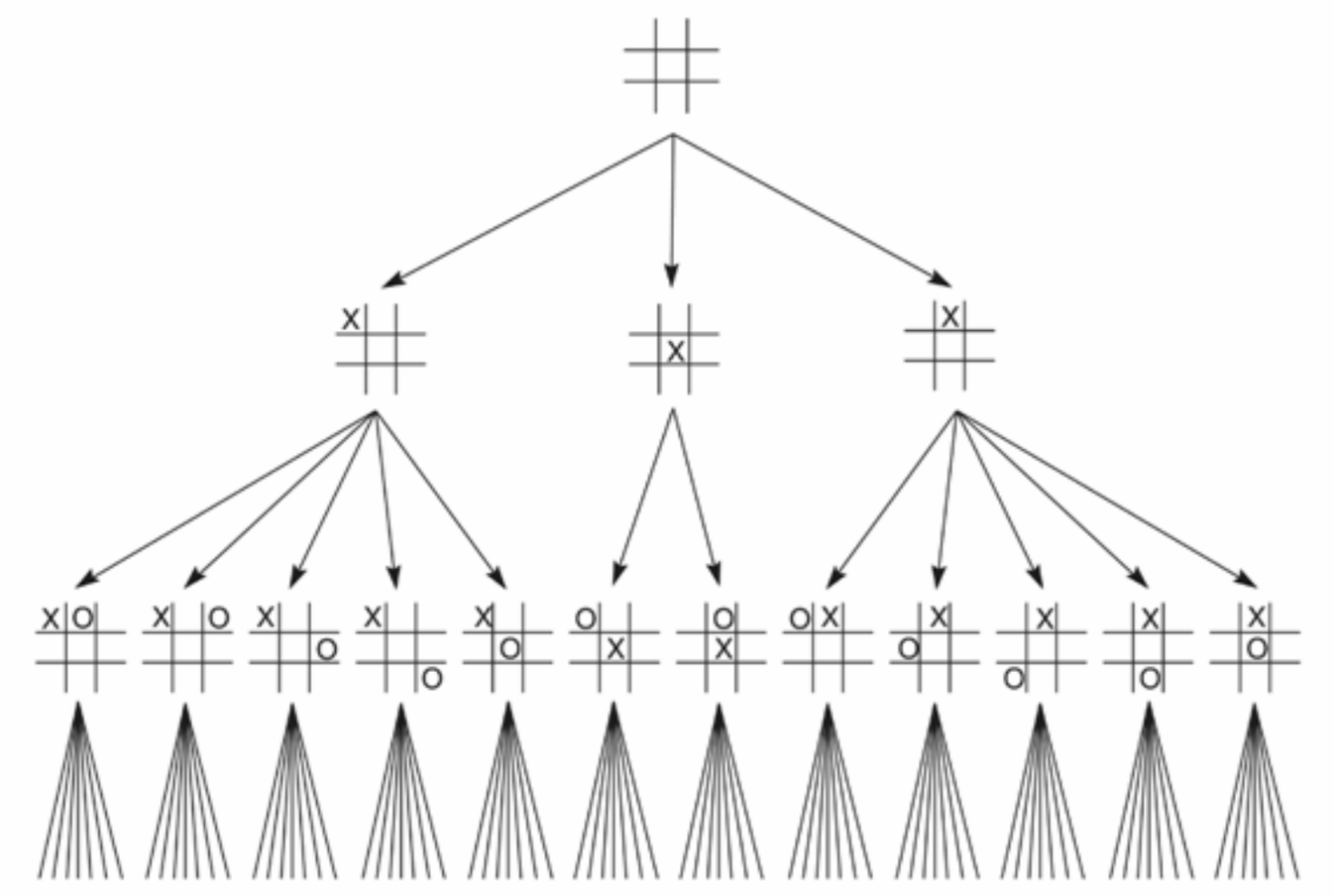
- Mỗi nút là 1 trạng thái
- Mỗi cạnh ứng với 1 phép chuyển đổi trạng thái



Không gian trạng thái của bài toán 8-puzze

https://voer.edu.vn/c/mo-dau/764b3239/70515552

Không gian trạng thái (KGTT)



Không gian trạng thái của bài toán Tic-tac-toe

https://voer.edu.vn/c/mo-dau/764b3239/2ab0f63d

Không gian trạng thái (KGTT)

Để xây dựng không gian trạng thái cho 1 bài toán, ta cần xác định:

1. **Trạng thái đầu**

   Trạng thái xuất phát. Một bài toán có thể có nhiều trạng thái xuất phát

2. **Tập trạng thái đích**

   Trạng thái mà bài toán được giải. Một bài toán có thể có nhiều trạng thái

3. **Các phép chuyển**

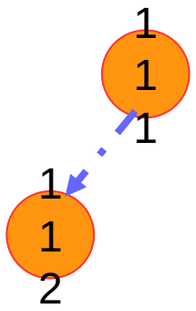
   Thao tác để chuyển từ 1 trạng thái này sang 1 trạng thái khác

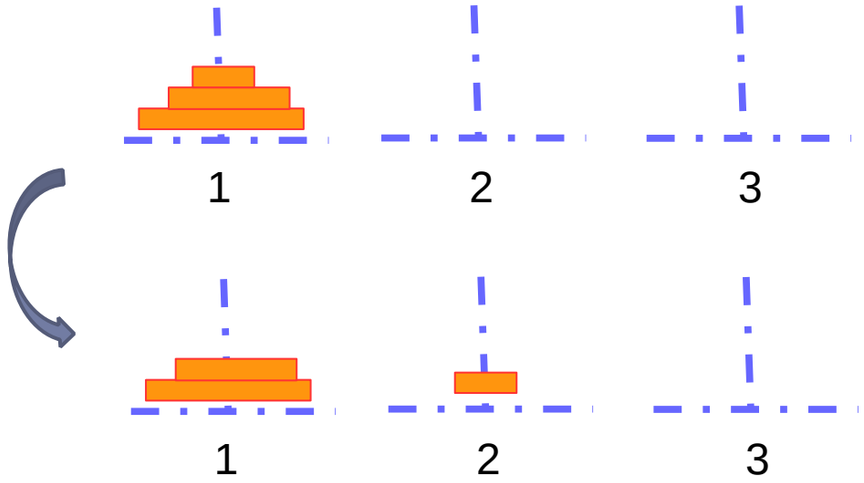
   v gọi là 1 trạng thái kề/con của u nếu có 1 phép chuyển từ u tới v

   Ví dụ:

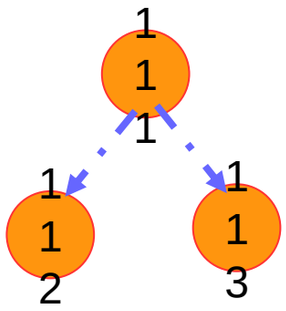
   - 8-puzzle: có 4 thao tác chuyển là up, down, left, right
   - Tic-tac-toe: thao tác chuyển là đánh thêm dấu của người chơi lên bàn cờ
   - 8 hậu: đặt thêm hậu/ thay đổi vị trí của 1 quân hậu
   - Cờ vua: luật chơi

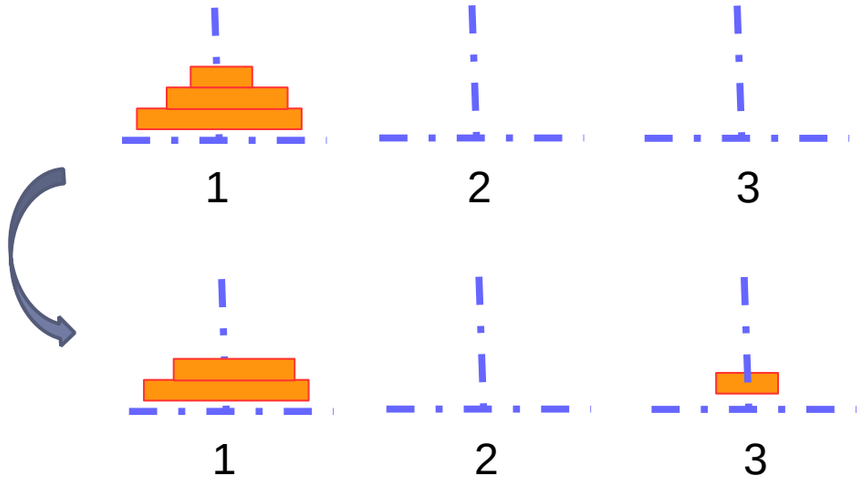
Không gian trạng thái của bài toán tháp Hà Nội



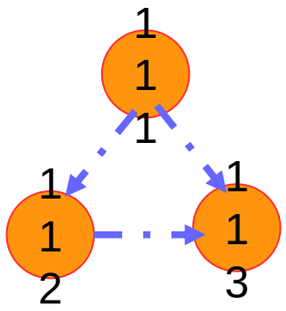


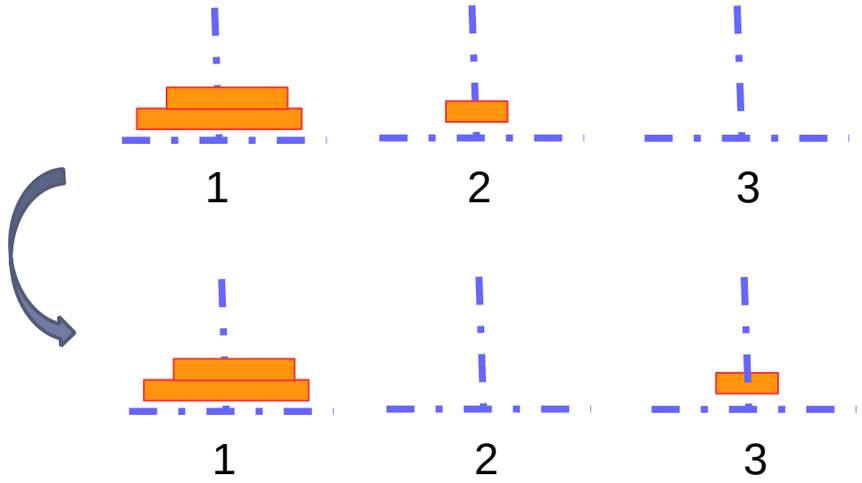
Không gian trạng thái của bài toán tháp Hà Nội



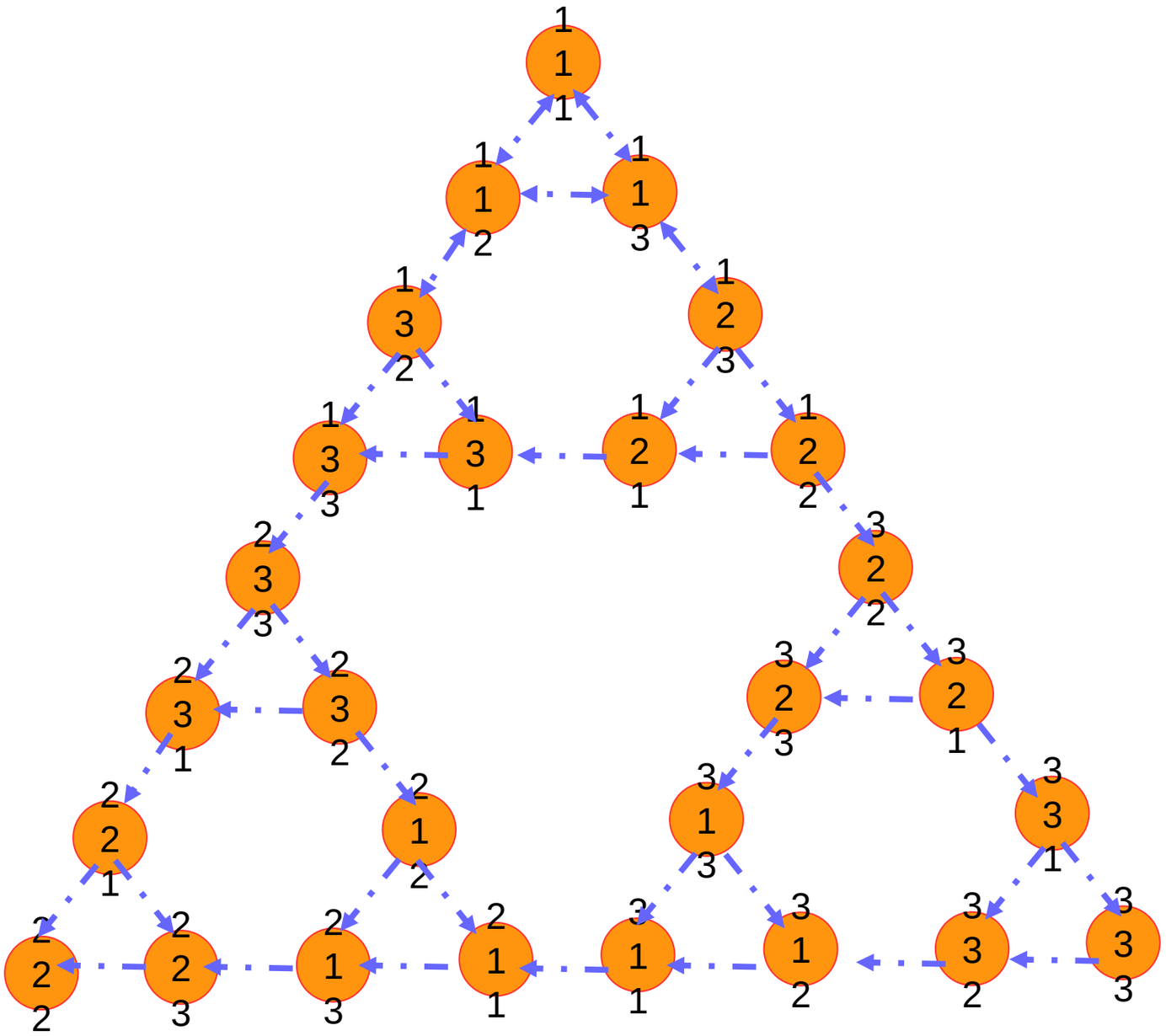


Không gian trạng thái của bài toán tháp Hà Nội



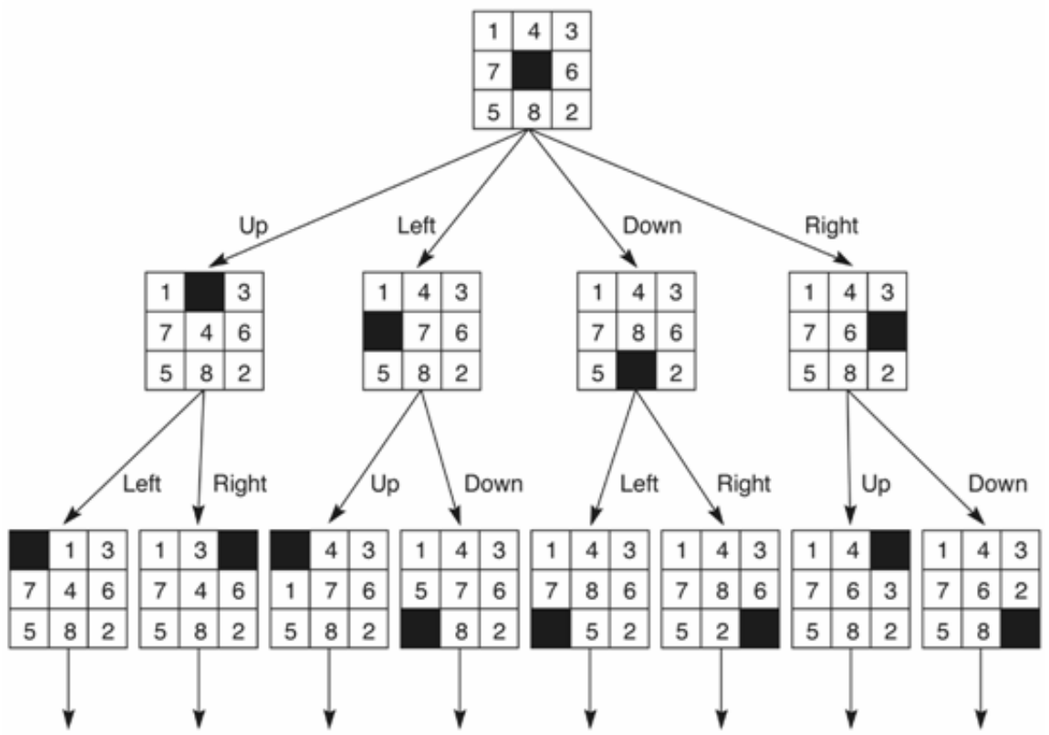


Không gian trạng thái của bài toán tháp Hà Nội



Không gian trạng thái (KGTT)

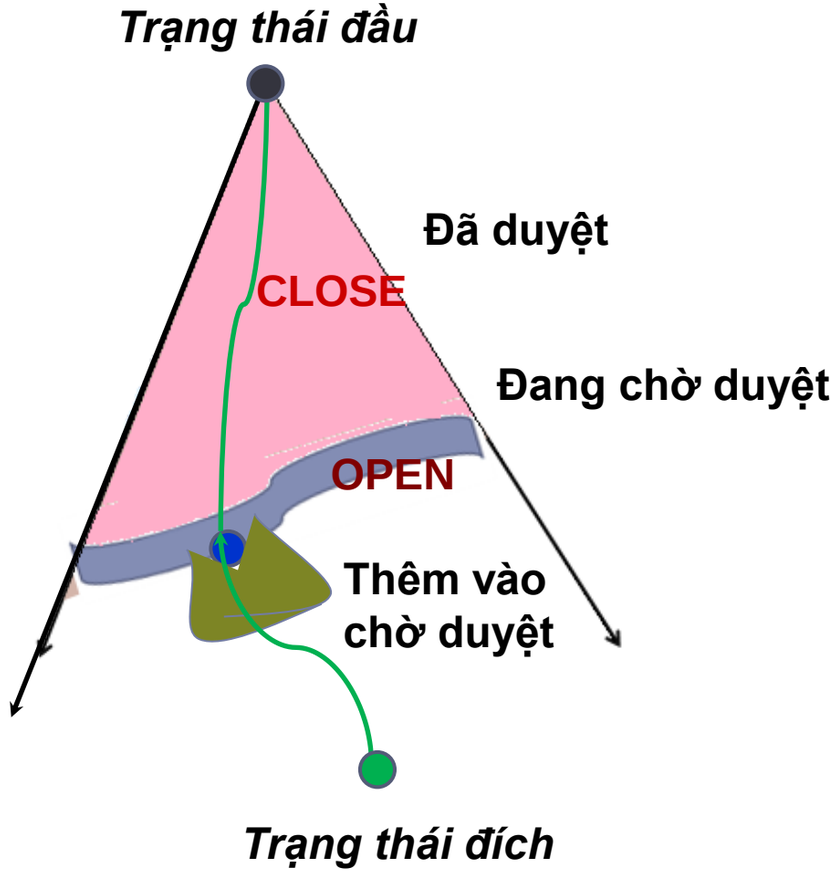
- Có thể là đồ thị chứa vòng lặp
- Không nhất thiết là cây



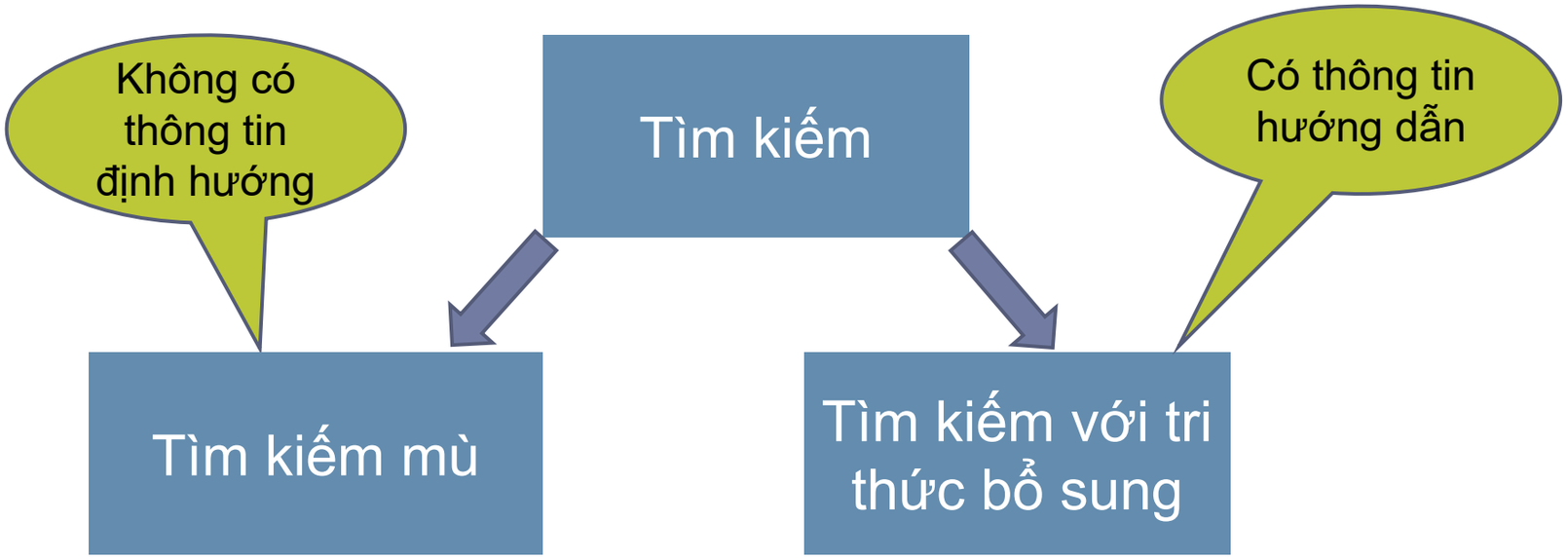
2.3 Giải quyết bài toán bằng tìm kiếm

Giải quyết bài toán bằng tìm kiếm

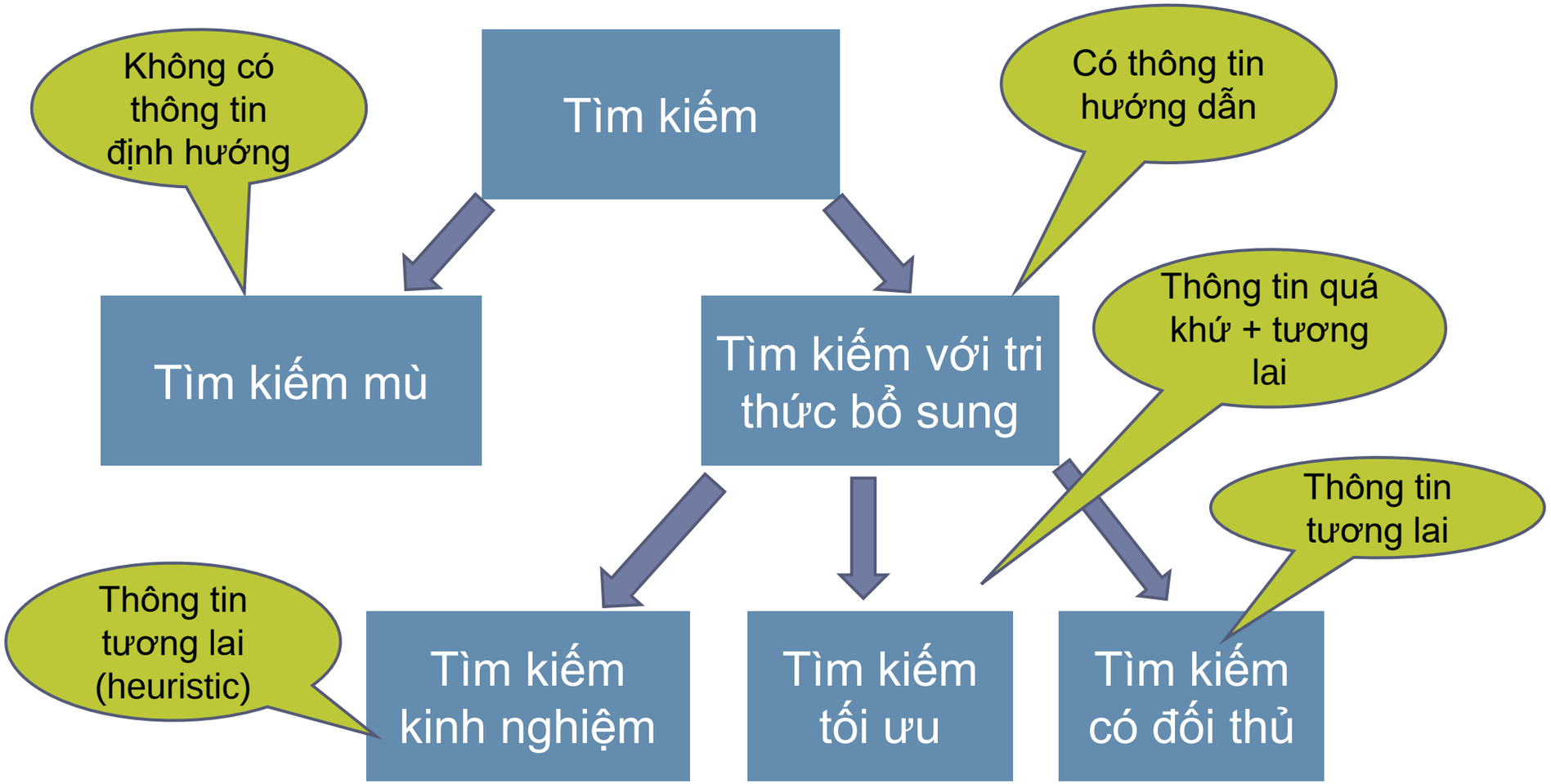
- Là tìm đường đi từ TT đầu tới một TT đích trong KGTT
  - Lời giải: tập các phép chuyển gắn với đường đi tìm được.
- Các chiến lược tìm kiếm khác nhau bởi thứ tự phát triển các nút để duyệt (cách chọn 1 trạng thái tiếp theo của để duyệt)
- Các thông tin được sử dụng để tìm đường:
  - Quá khứ: đánh giá chi phí đường đi từ gốc tới nút đang duyệt
  - Tương lai: ước lượng chi phí đường đi từ nút đang duyệt tới đích



Các chiến lược tìm kiếm



Các chiến lược tìm kiếm



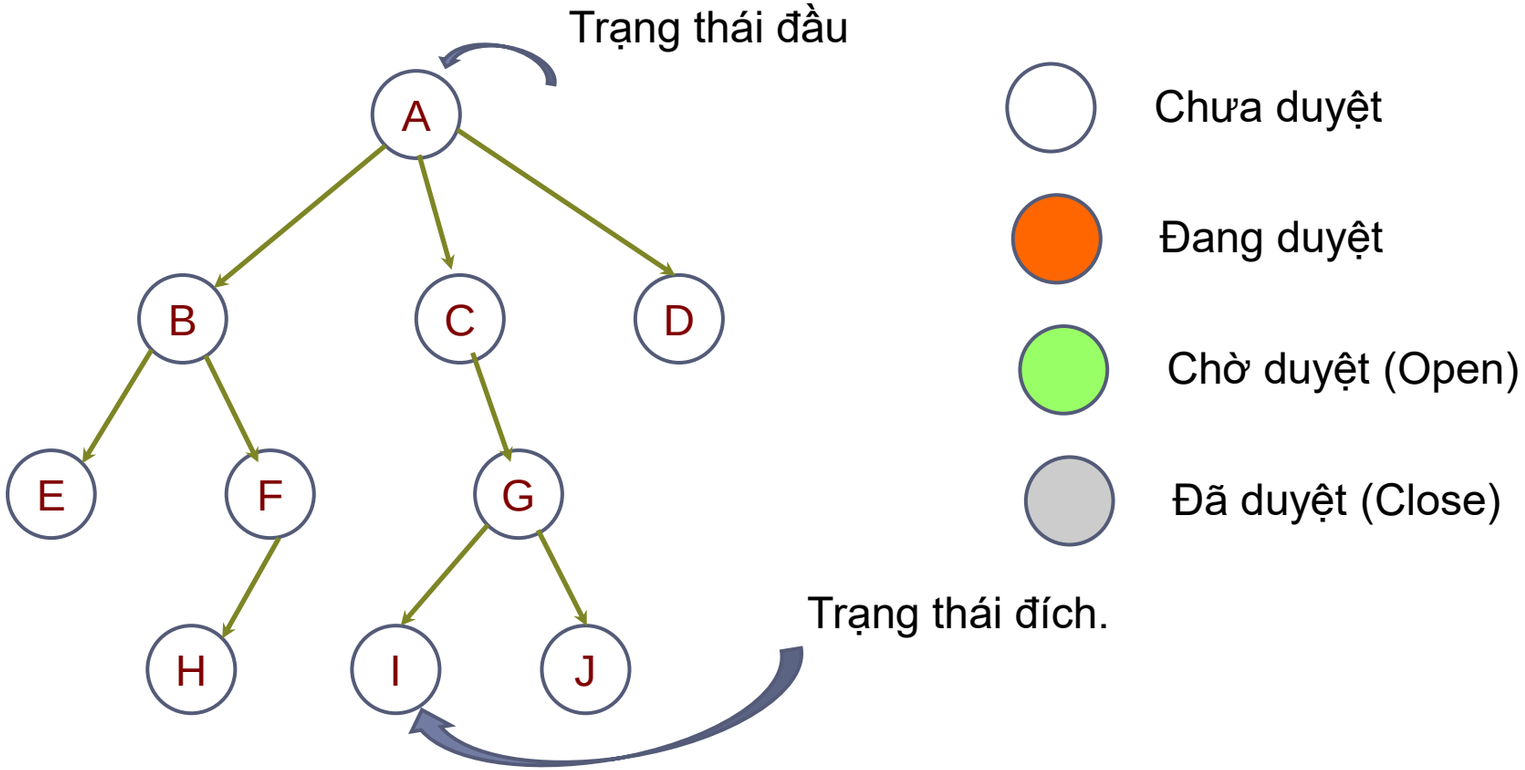
Tìm kiếm mù

Tìm kiếm mù

- Tìm kiếm không có định hướng: không có thông tin đánh giá (quá khứ, tương lai) của các trạng thái
- Từ trạng thái ban đầu, các trạng thái được phát triển theo 1 quy tắc lựa chọn cứng nhắc nào đó cho tới khi:
  - Gặp được 1 trạng thái đích → tìm kiếm thành công
  - Duyệt hết không gian trạng thái → tìm kiếm thất bại
- **Các chiến lược tìm kiếm mù:**
  - Tìm kiếm theo chiều rộng (Breadth-First-Search: BFS)
  - Tìm kiếm theo chiều sâu (Depth-First-Search: DFS)
  - Tìm kiếm theo chiều sâu hạn chế (Limited Depth-First-Search: LDFS)
  - Tìm kiếm sâu lặp (Depth Deepening-Search: DDS)

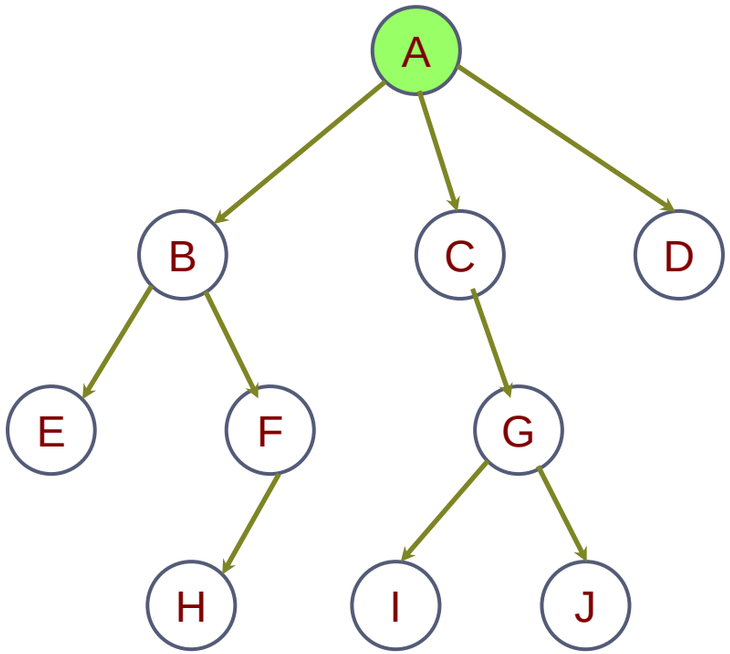
Tìm kiếm theo chiều rộng

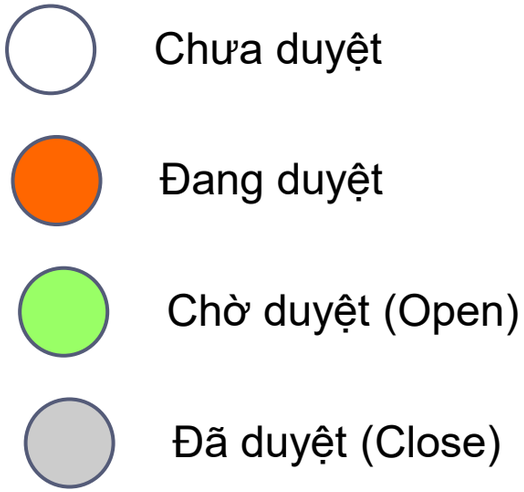
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo



Tìm kiếm theo chiều rộng

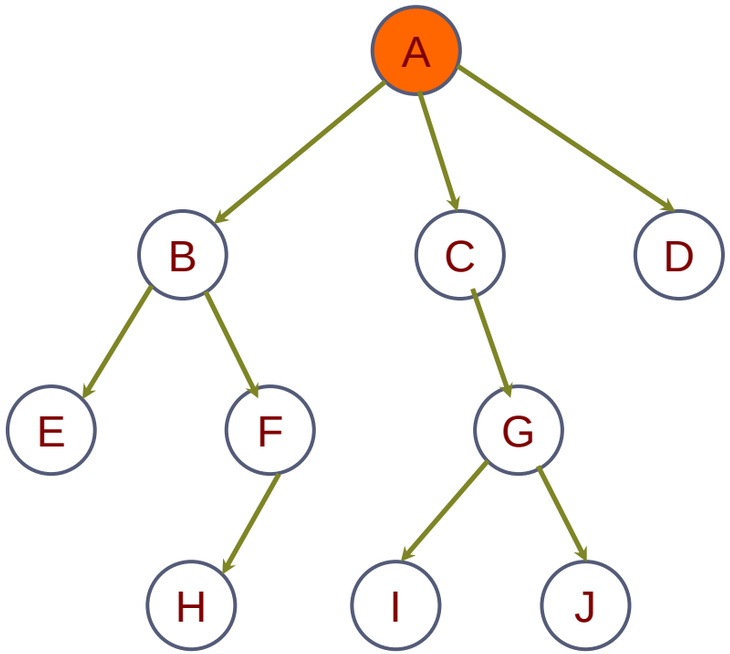
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

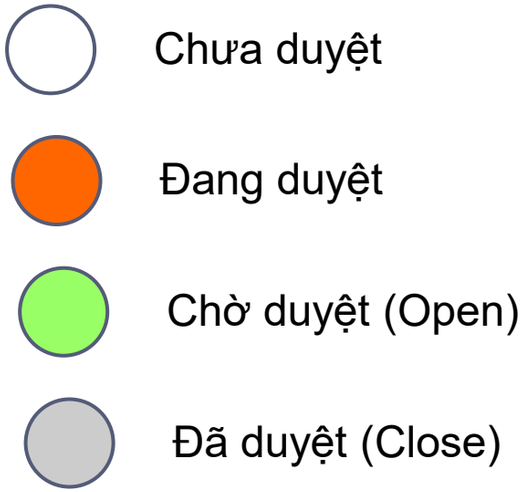




Tìm kiếm theo chiều rộng

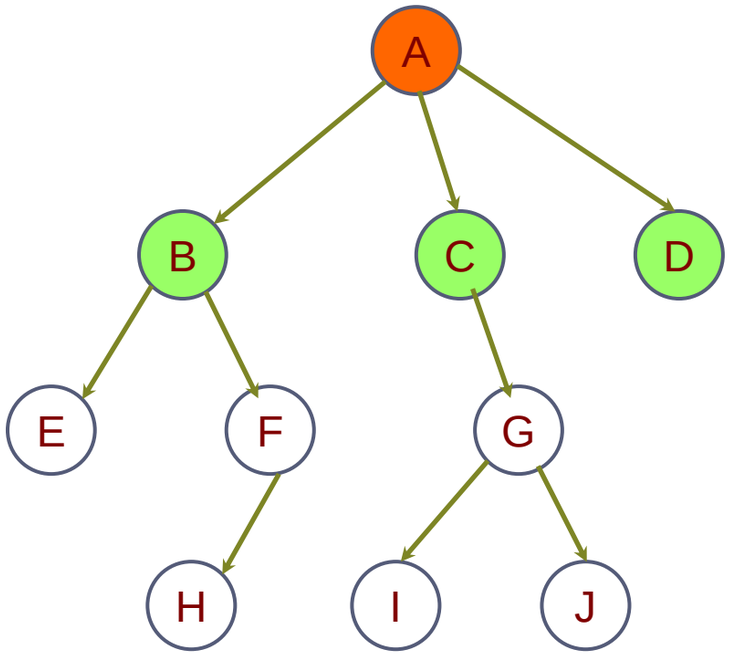
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

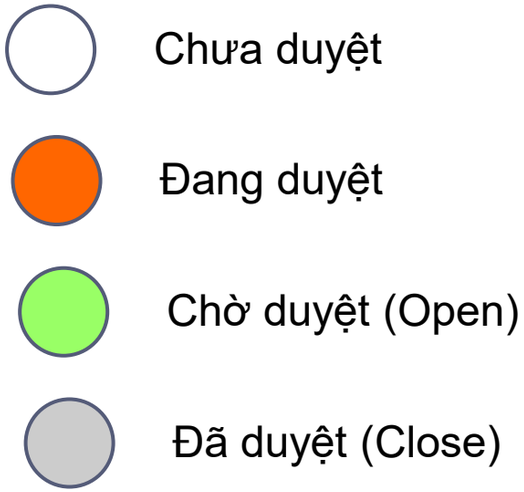




Tìm kiếm theo chiều rộng

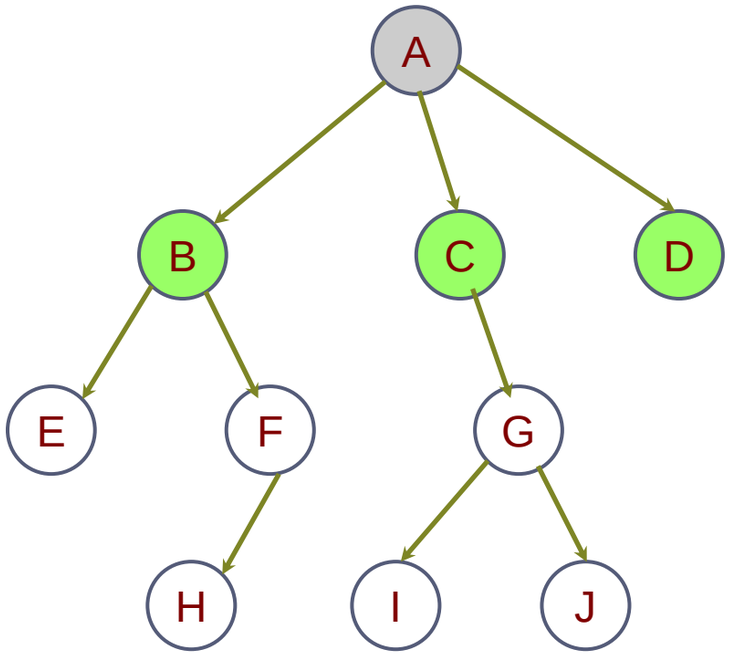
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

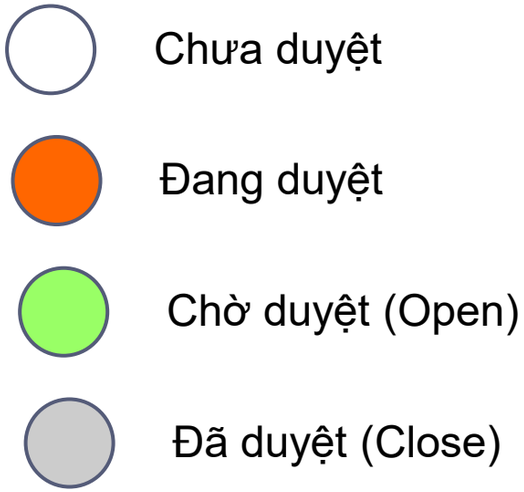




Tìm kiếm theo chiều rộng

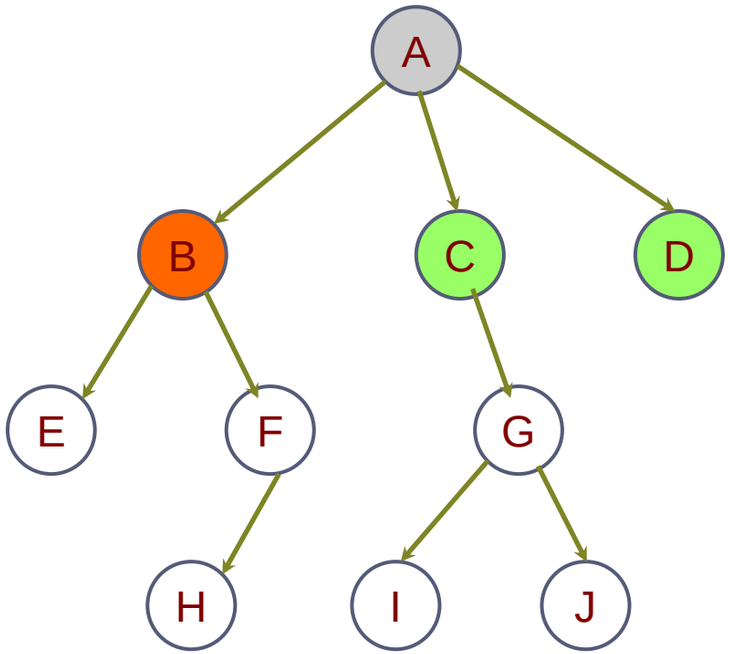
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

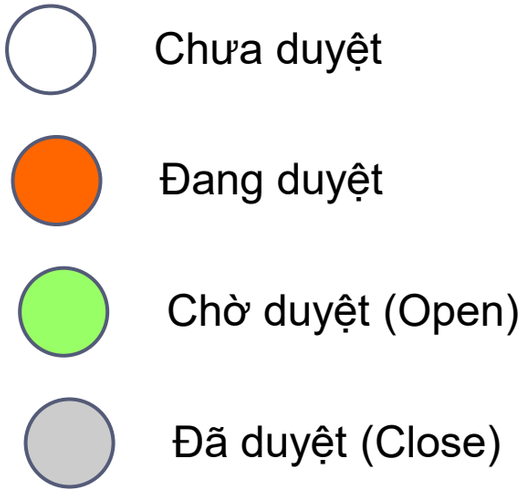




Tìm kiếm theo chiều rộng

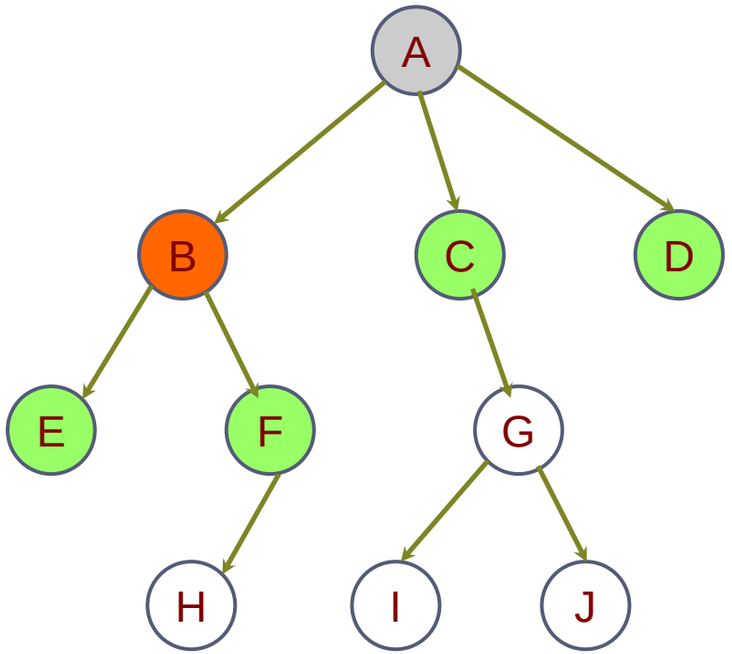
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

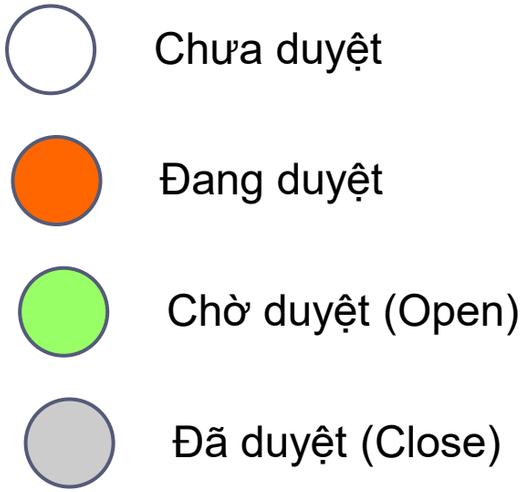




Tìm kiếm theo chiều rộng

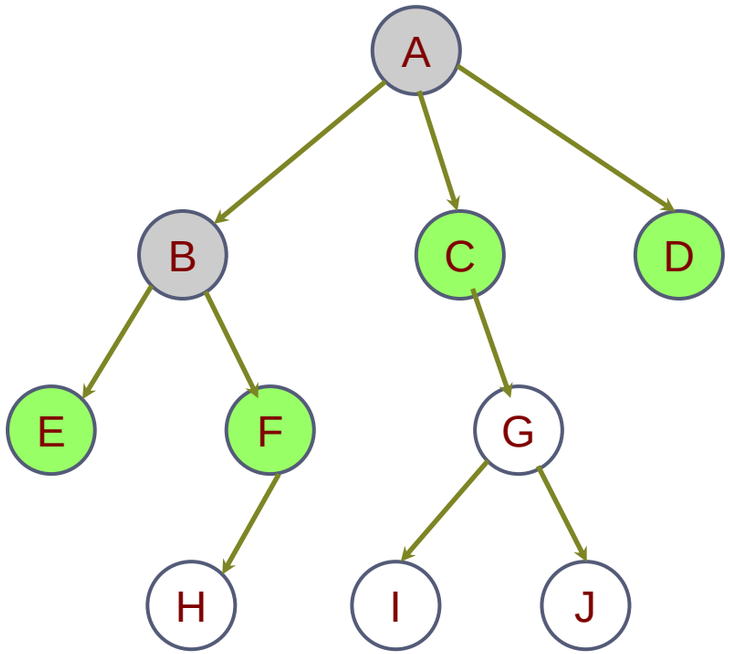
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

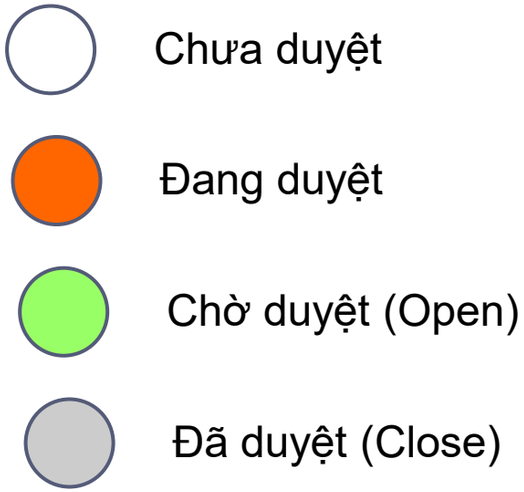




Tìm kiếm theo chiều rộng

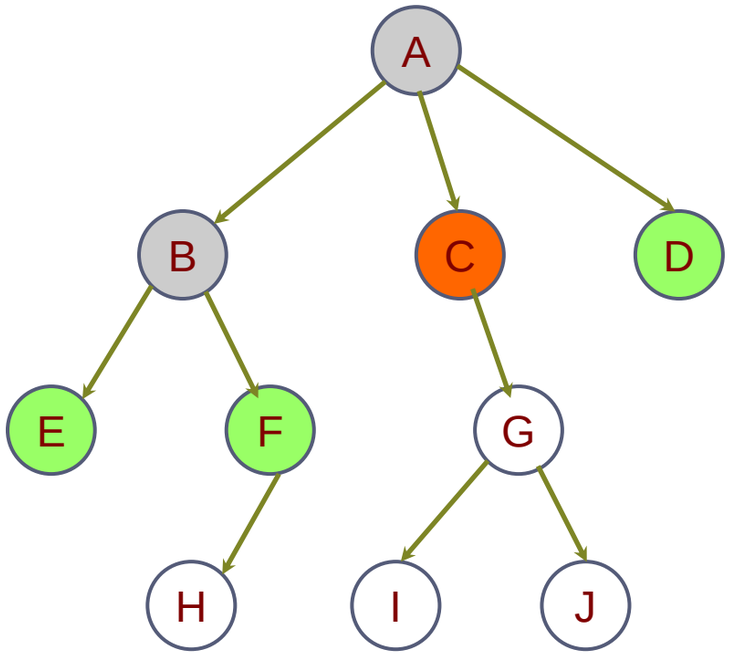
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

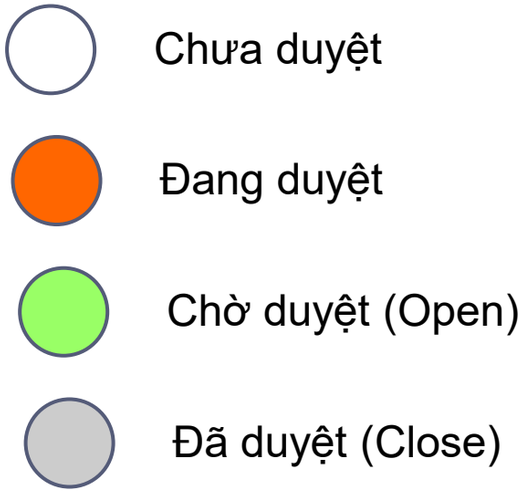




Tìm kiếm theo chiều rộng

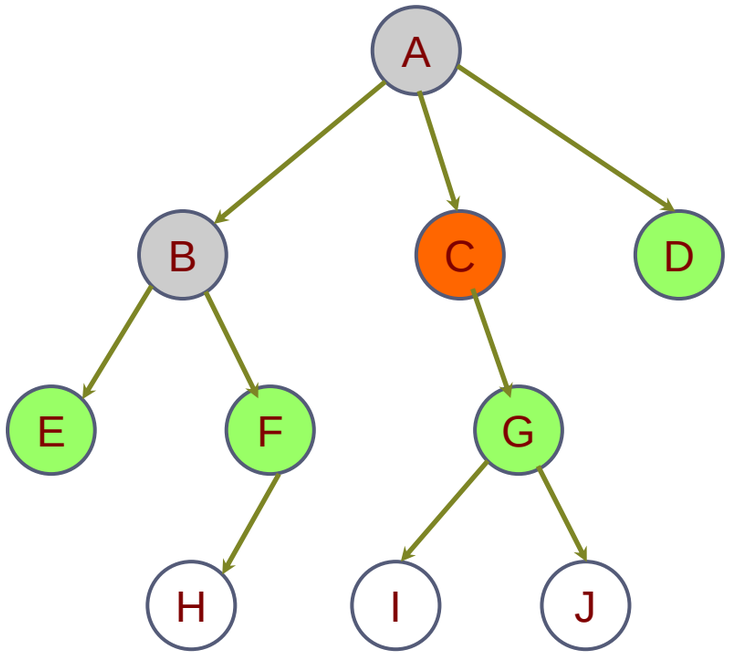
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

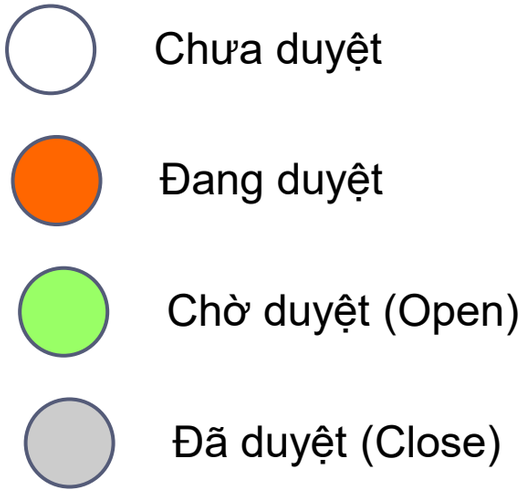




Tìm kiếm theo chiều rộng

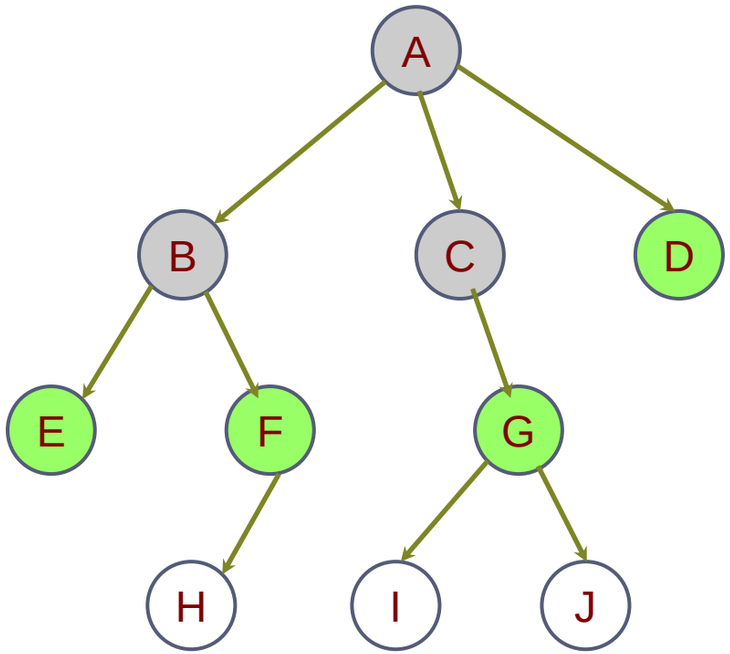
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

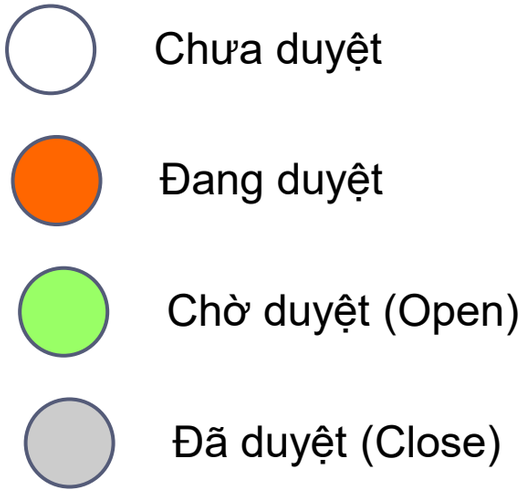




Tìm kiếm theo chiều rộng

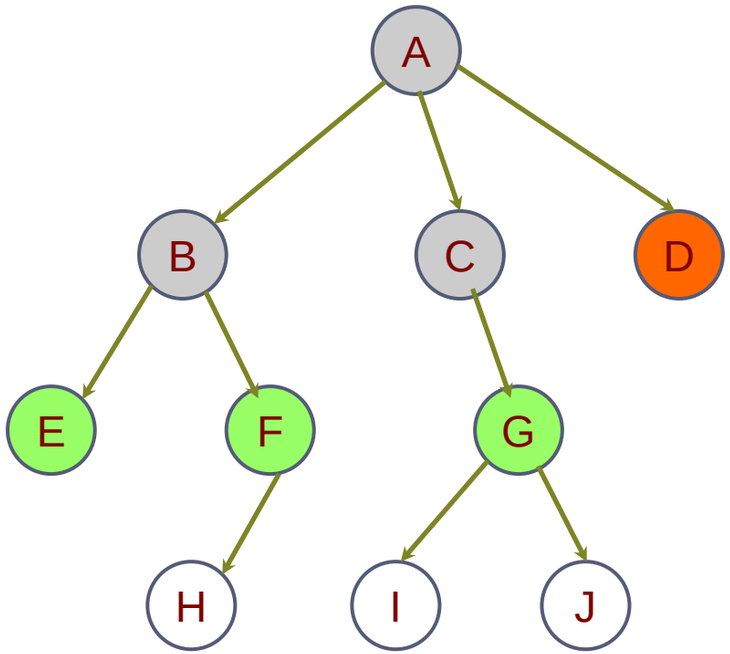
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

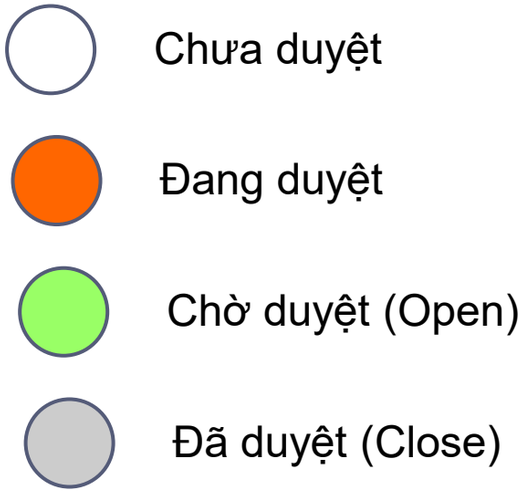




Tìm kiếm theo chiều rộng

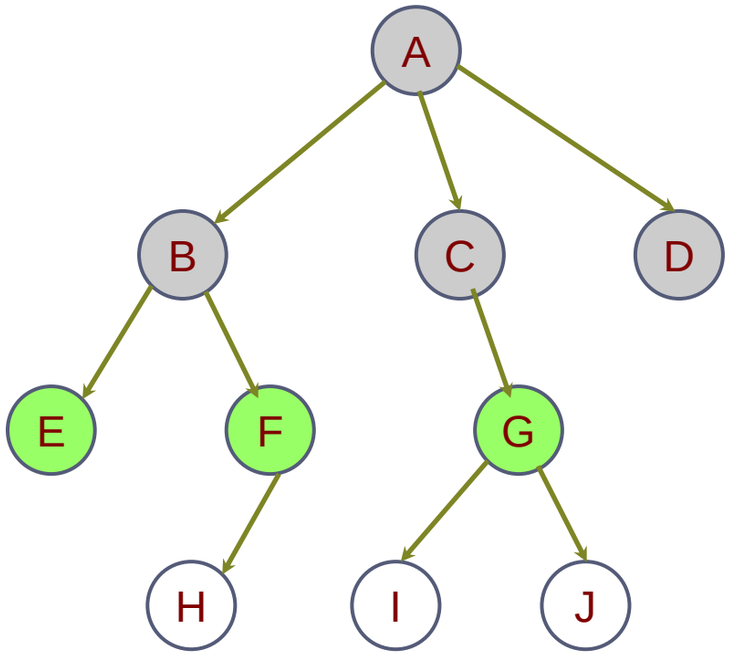
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

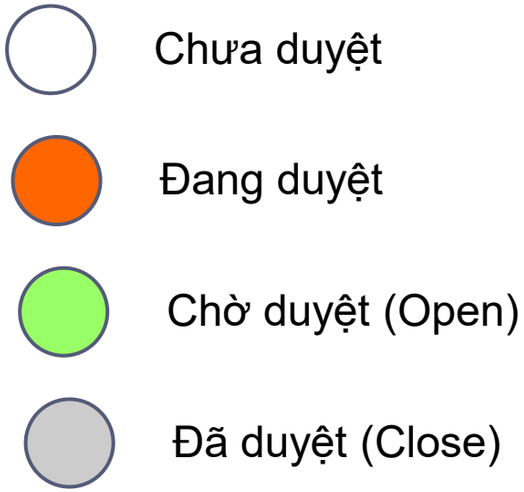




Tìm kiếm theo chiều rộng

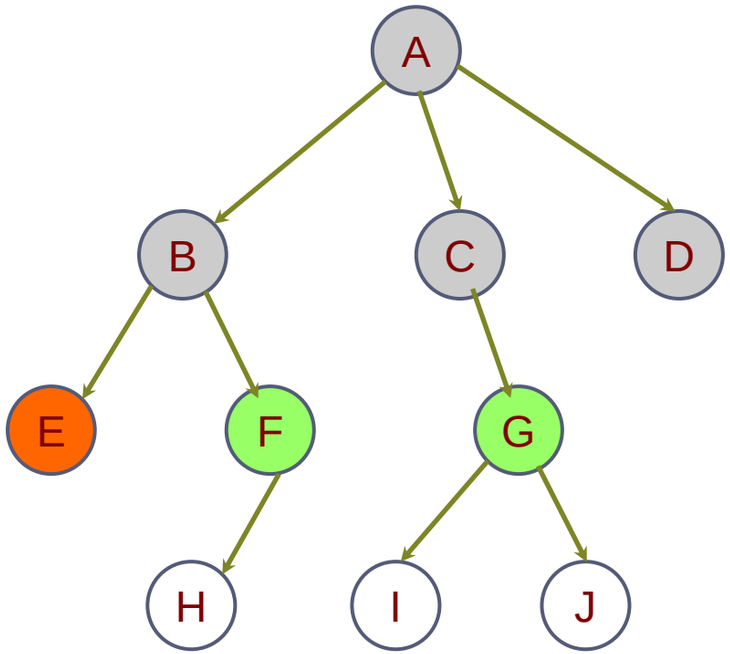
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

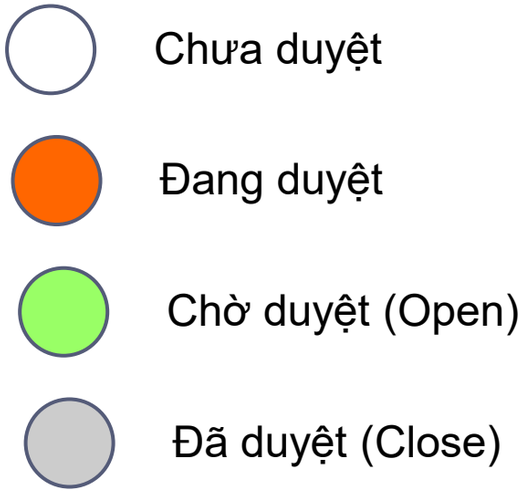




Tìm kiếm theo chiều rộng

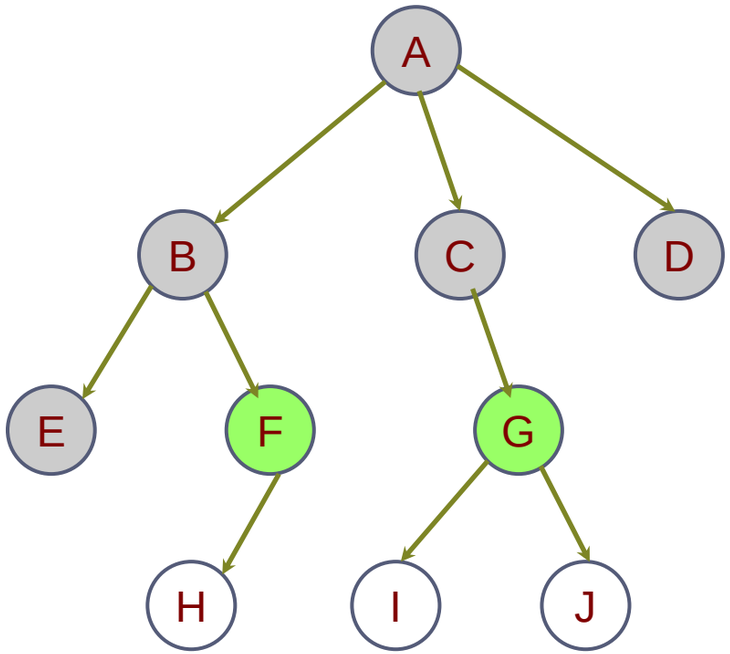
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

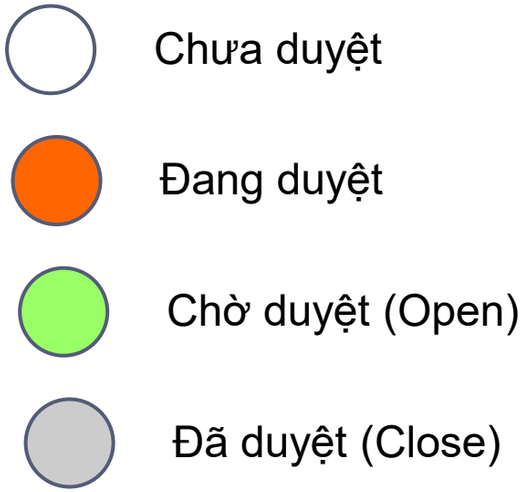




Tìm kiếm theo chiều rộng

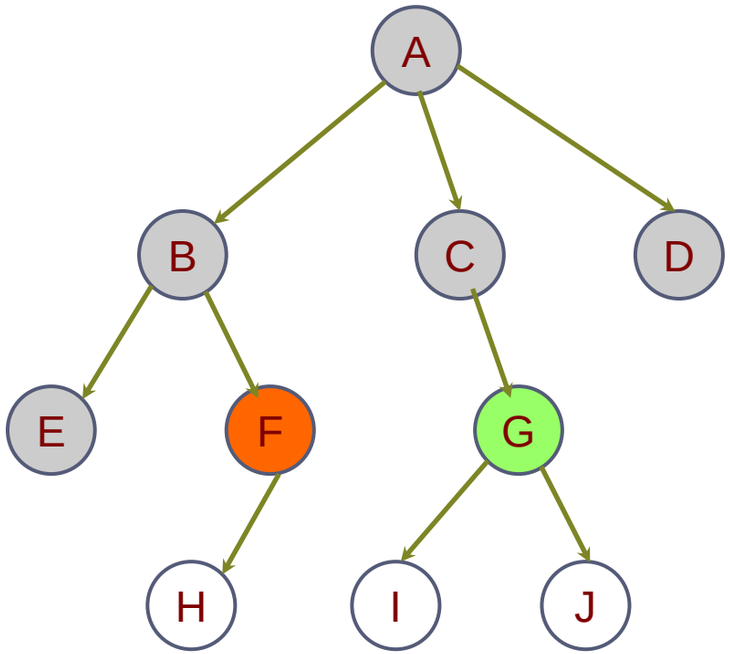
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

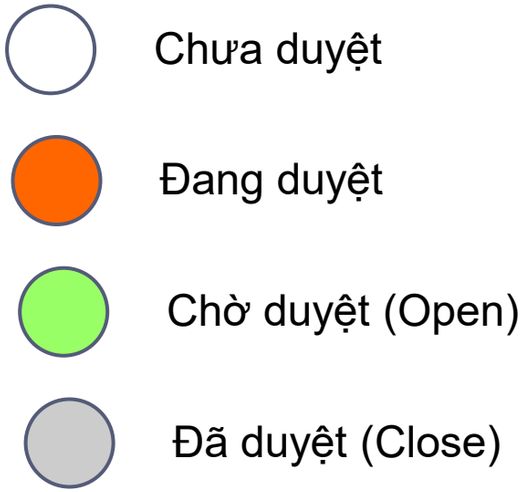




Tìm kiếm theo chiều rộng

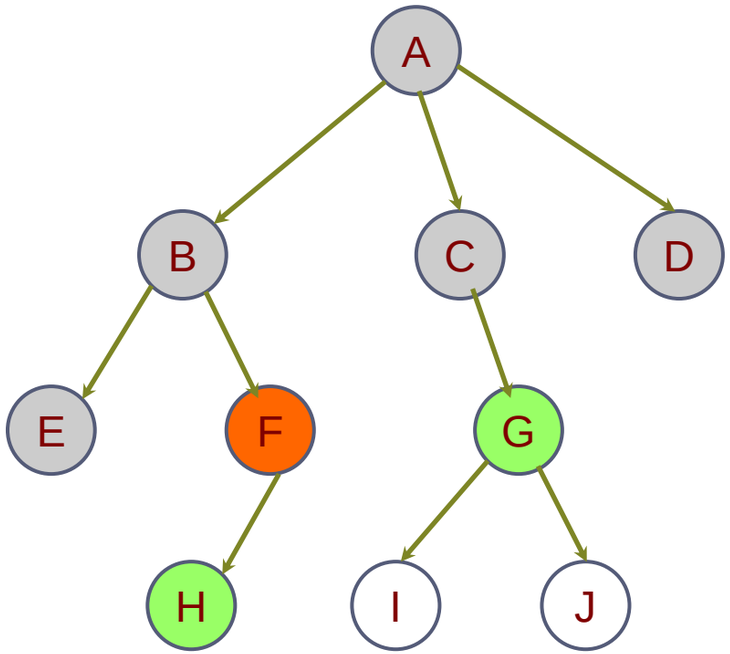
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

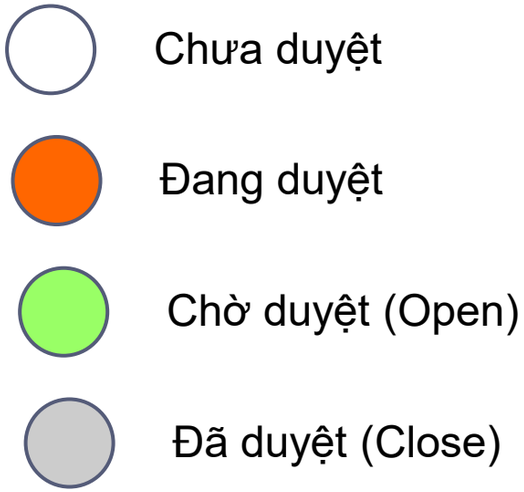




Tìm kiếm theo chiều rộng

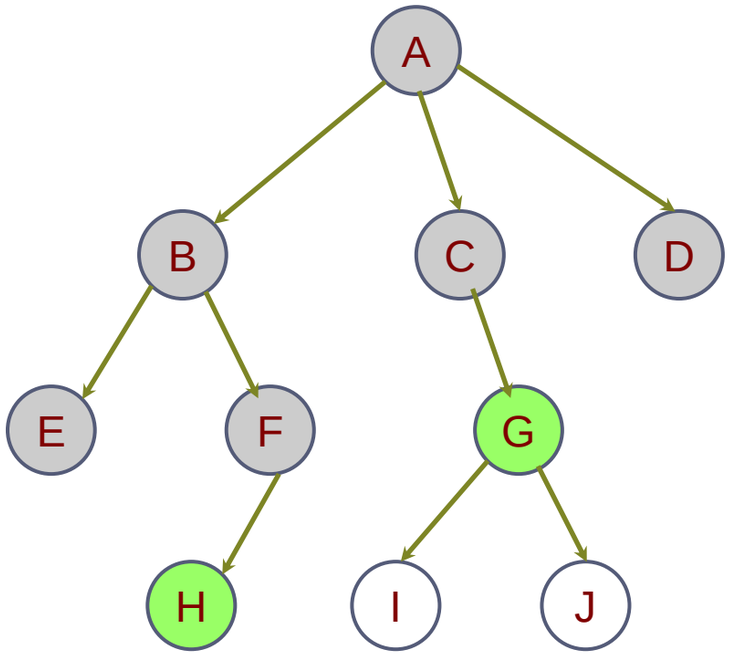
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

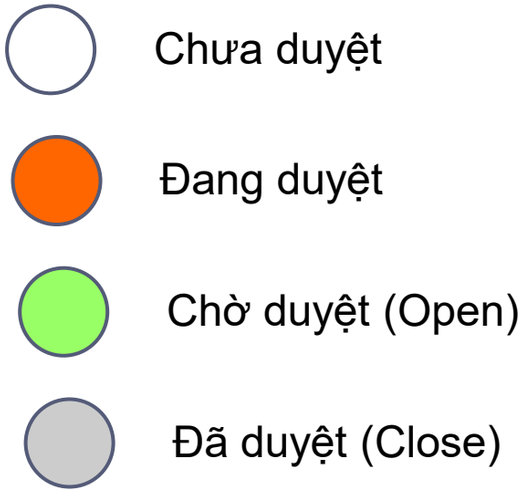




Tìm kiếm theo chiều rộng

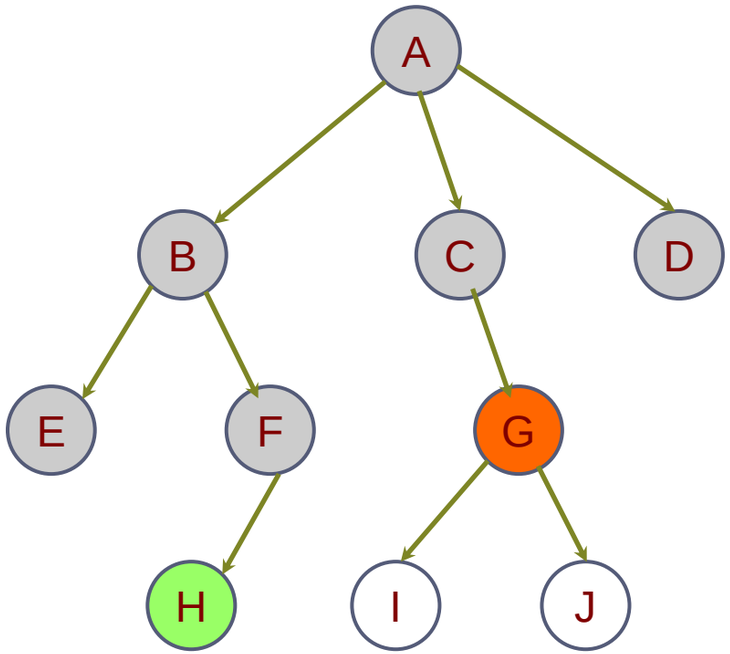
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

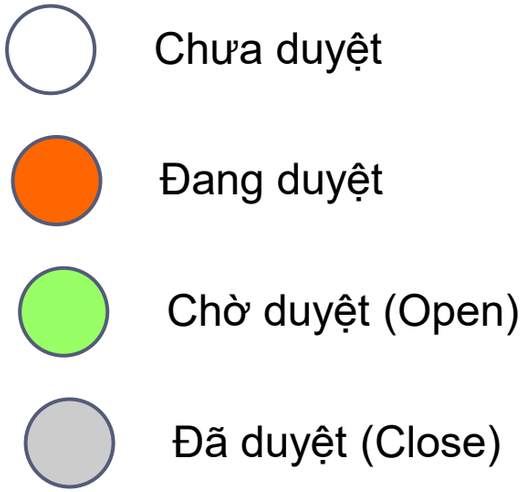




Tìm kiếm theo chiều rộng

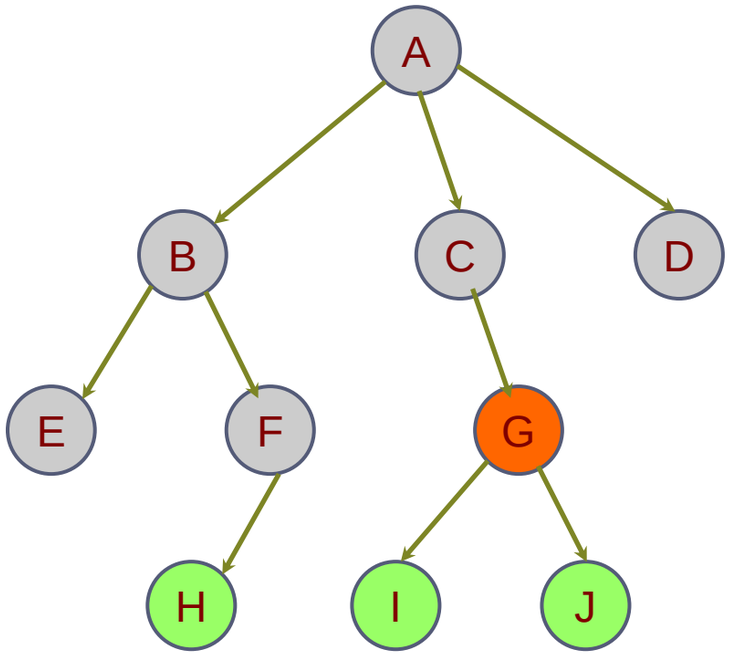
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

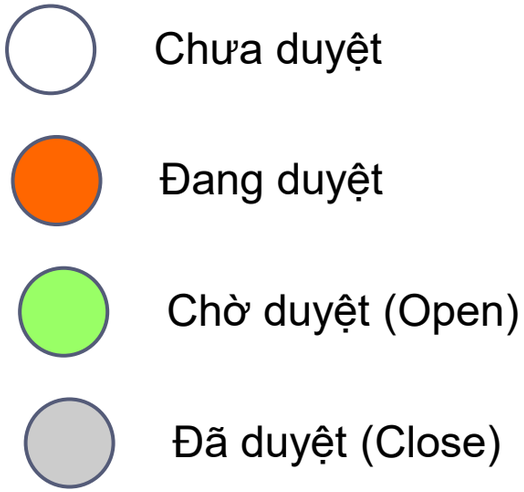




Tìm kiếm theo chiều rộng

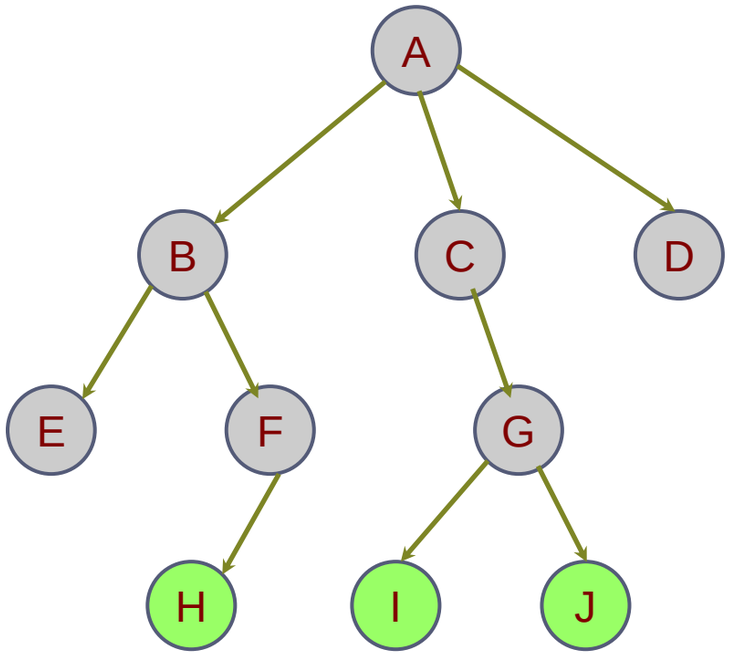
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

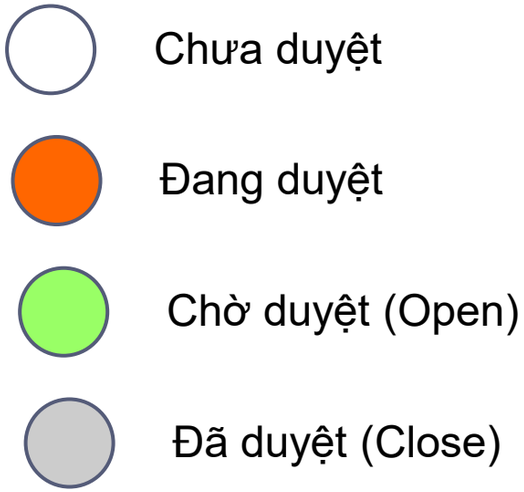




Tìm kiếm theo chiều rộng

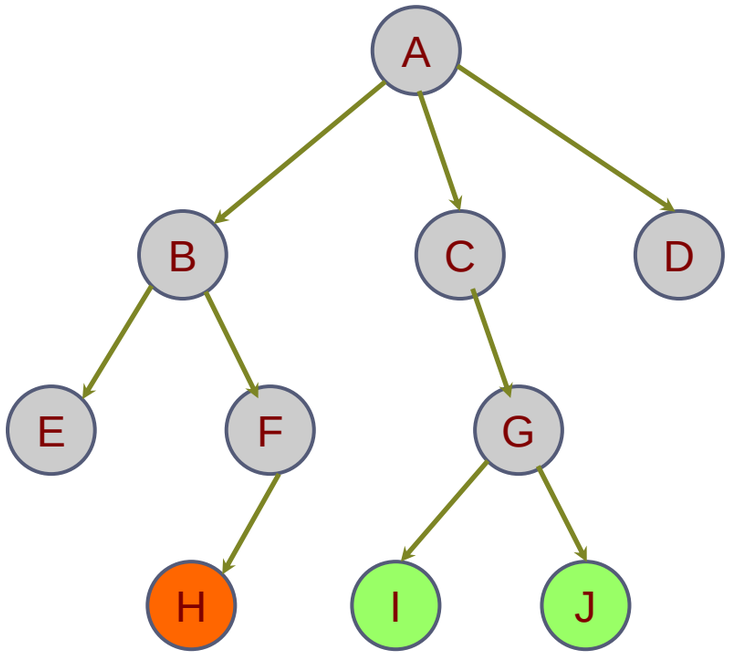
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

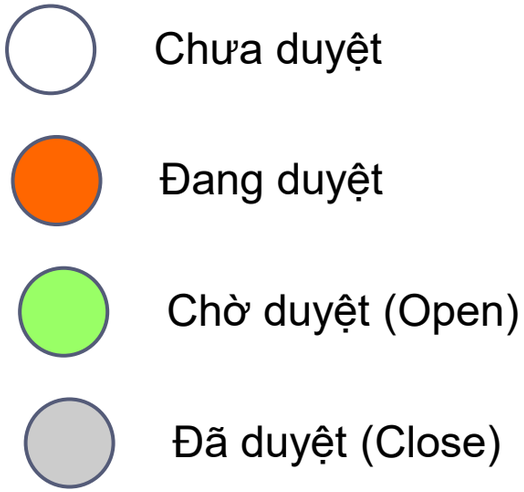




Tìm kiếm theo chiều rộng

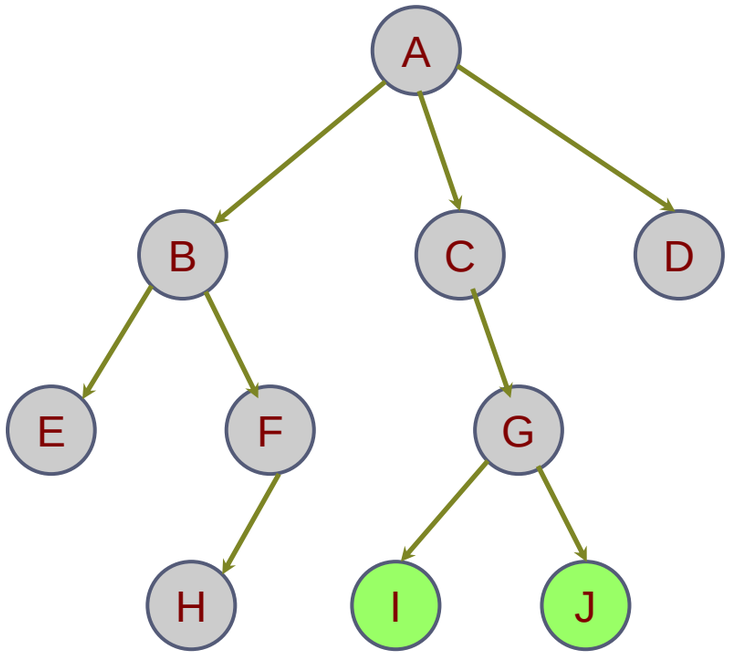
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

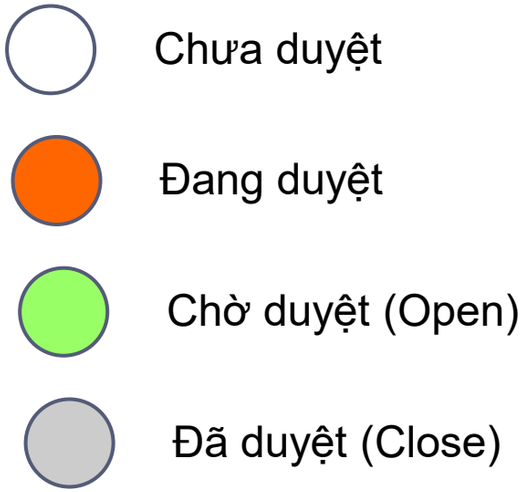




Tìm kiếm theo chiều rộng

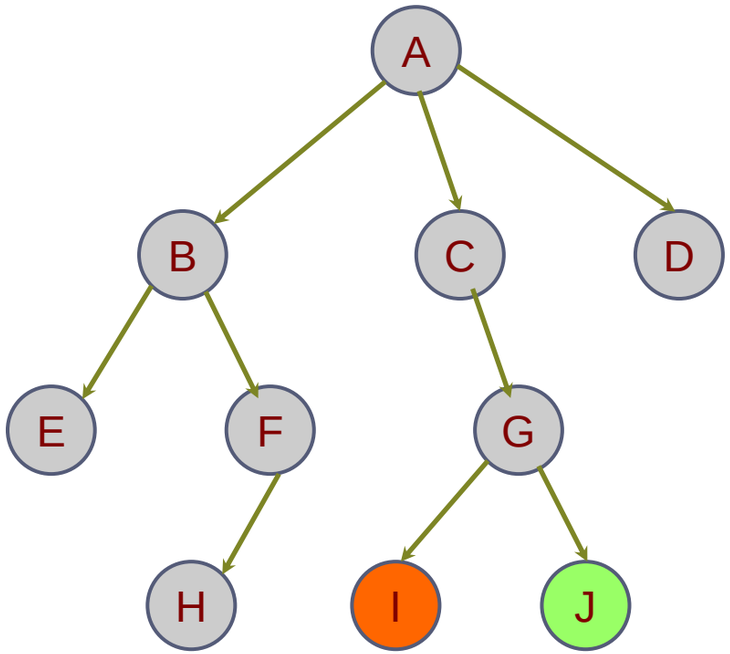
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo

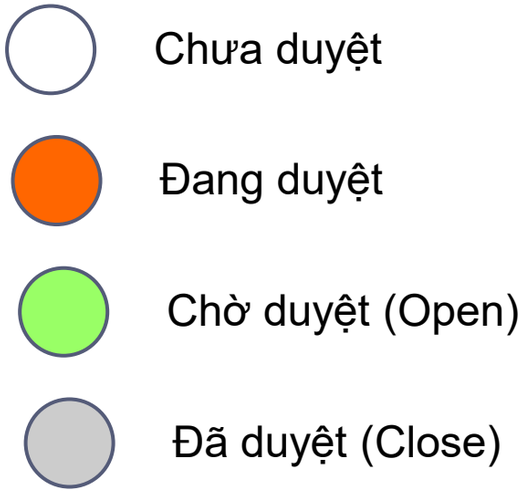




Tìm kiếm theo chiều rộng

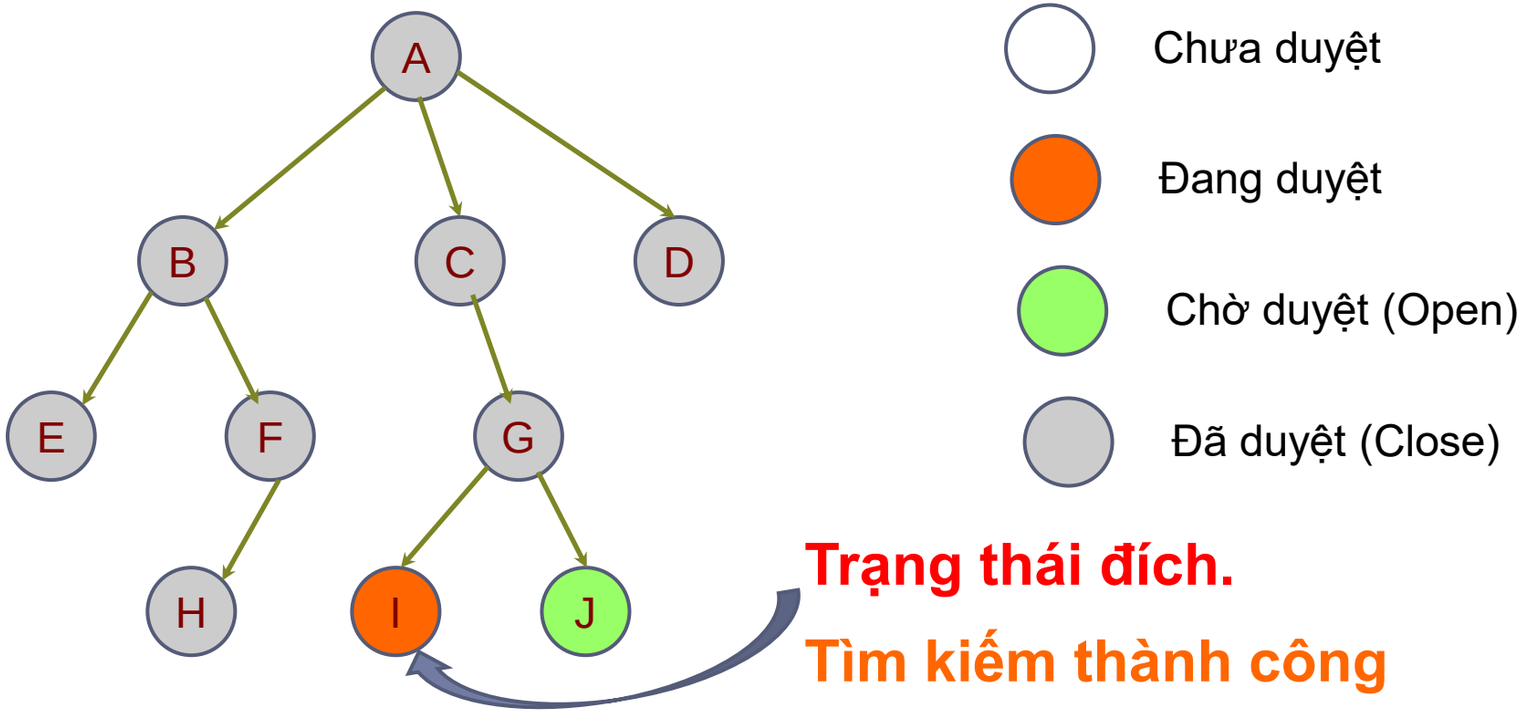
- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo





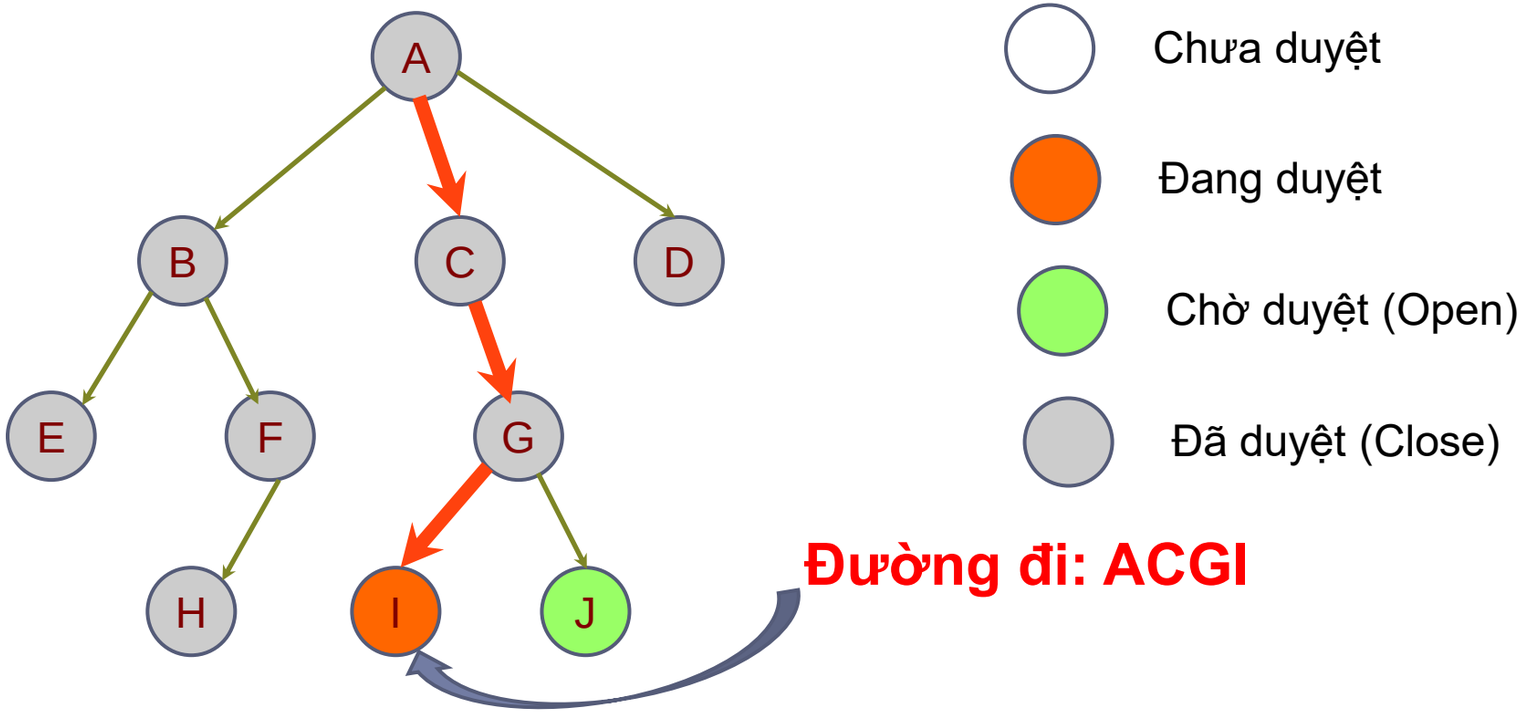
Tìm kiếm theo chiều rộng

- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo



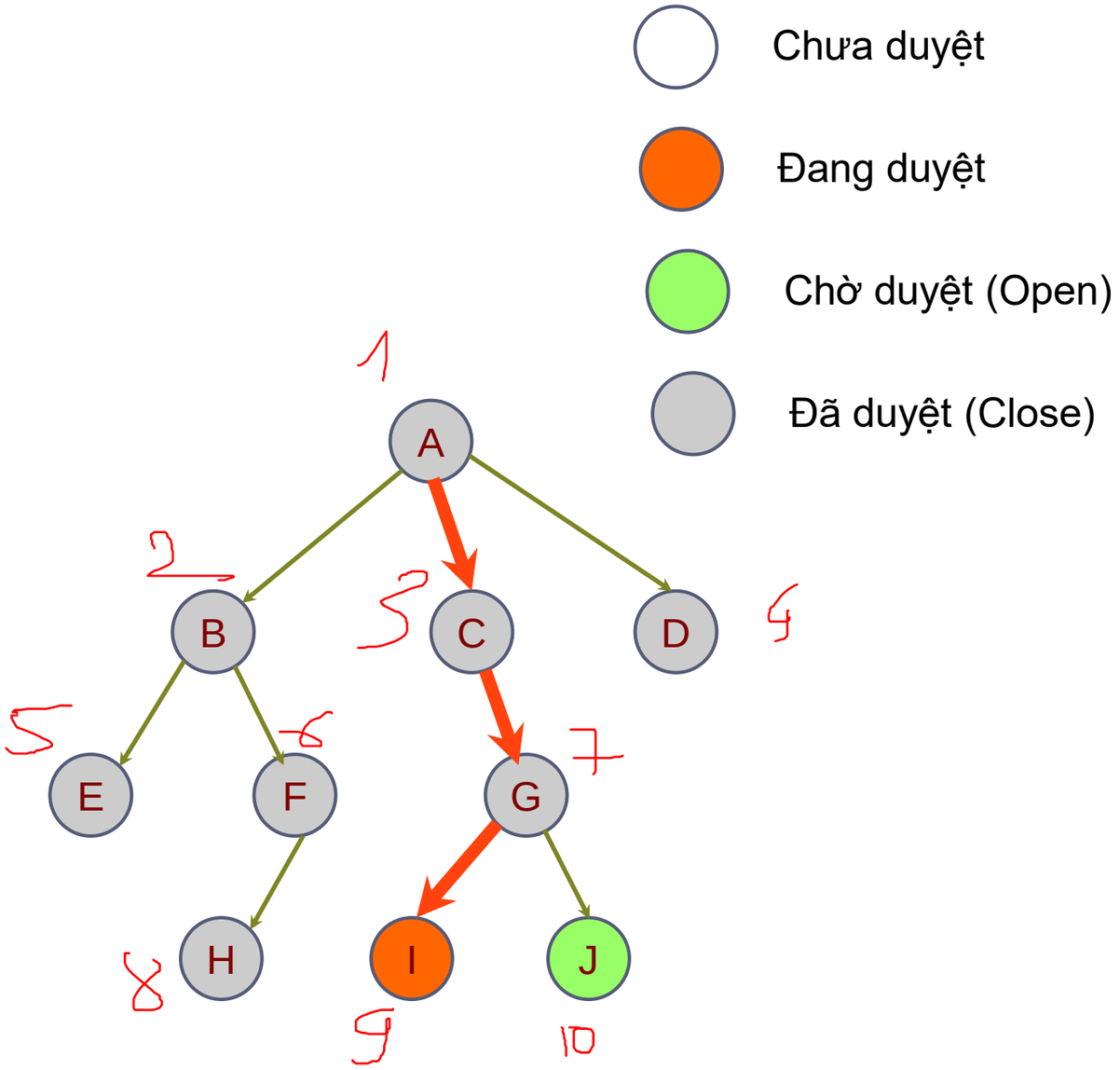
Tìm kiếm theo chiều rộng

- Tập trạng thái chờ duyệt “OPEN”: Queue
- Trạng thái nào được sinh ra trước thì được phát triển trước
- Duyệt hết các nút ở cùng độ sâu rồi mới duyệt tới các nút ở độ sâu tiếp theo



Tìm kiếm theo chiều rộng

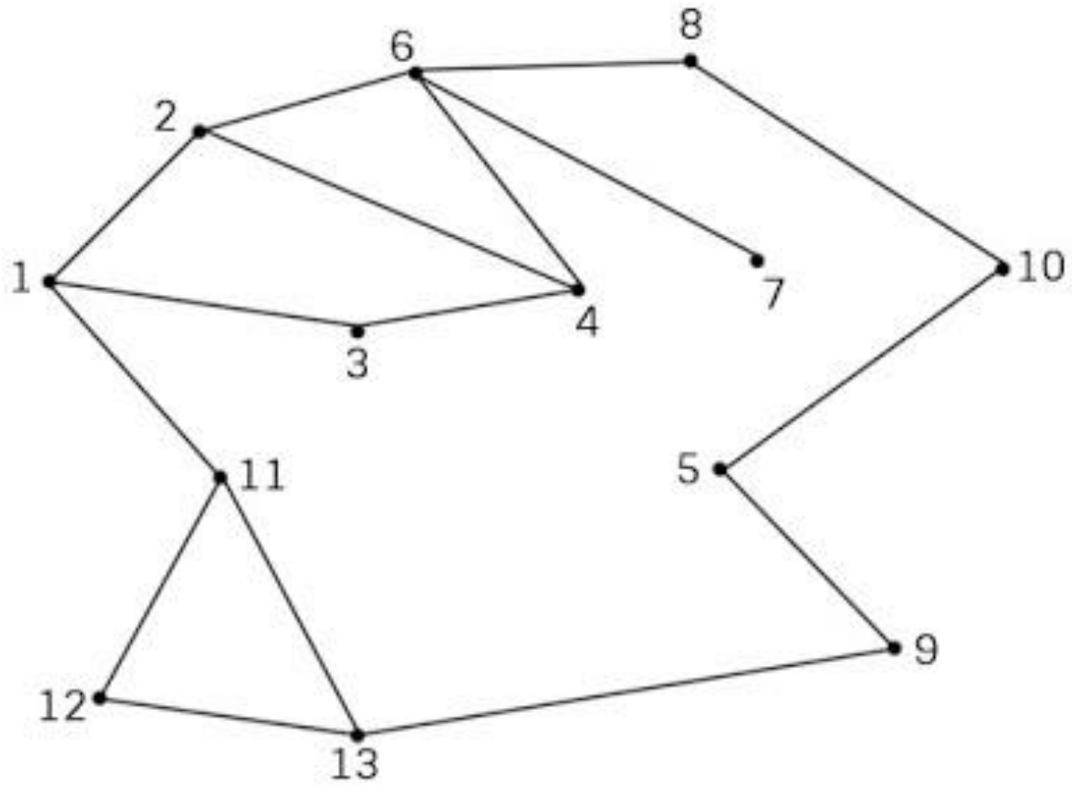
Lần lặp	u	kề(u)	OPEN	CLOSE
0			A	
1	A	B,C,D	B,C,D	A
2	B	E,F	C,D,E,F	A,B
3	C	G	D,E,F,G	A,B,C
4	D		E,F,G	A,B,C,D
5	E		F,G	A,B,C,D,E
6	F	H	G,H	A,B,C,D,E,F
7	G	I,J	H,I,J	A,B,C,D,E,F,G
8	H		I,J	A,B,C,D,E,F,G,H
9	I			A,B,C,D,E,F,G,H,I



Tìm kiếm theo chiều rộng

- Bài tập: Tìm đường đi và trình tự duyệt các đỉnh
  - Trạng thái đầu: 1
  - Trạng thái đích: 12

Lần lặp	u	kề(u)	OPEN
0			1



Tìm kiếm theo chiều rộng

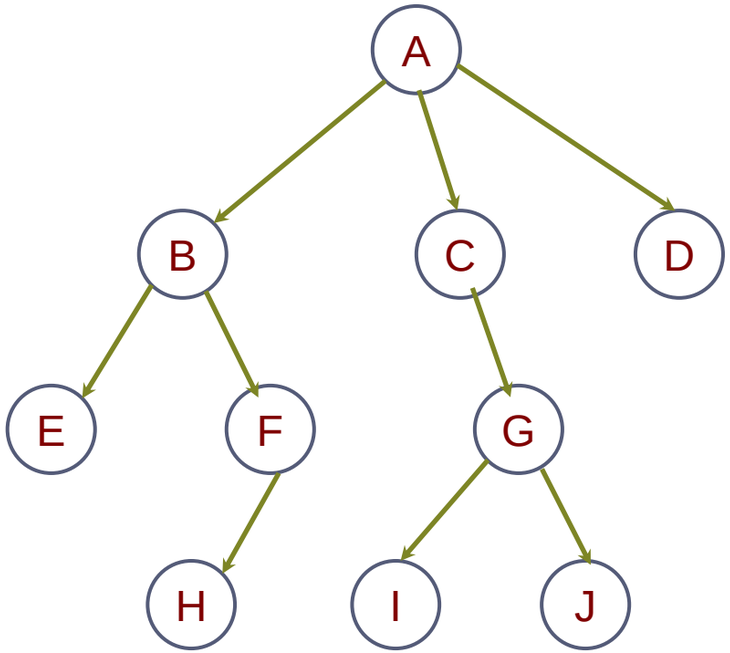
```
Procedure Breadth_First_Search
Begin
  Khởi tạo danh sách OPEN = {trạng thái ban đầu};
  while true do
    if (Open rỗng) then
      {thông báo tìm kiếm thất bại; stop};
    Loại trạng thái u ở đầu danh sách OPEN;
    if u là trạng thái kết thúc then
      {thông báo tìm kiếm thành công; stop};
    Thêm các đỉnh trạng thái kề với u vào cuối
danh sách OPEN;
End
```

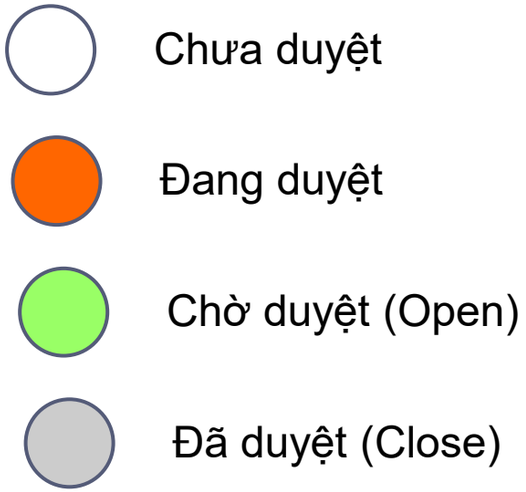
Tìm kiếm theo chiều rộng*

- Giả sử:
  - Cây tìm kiếm có nhân tố nhánh (số nút con tối đa của 1 đỉnh) là b
  - Lời giải của bài toán là đường đi có độ dài d
- Độ phức tạp thời gian:
  - Số nút duyệt $= 1 + b + b^2 + \ldots + b^{d-1} + k = O(b^d)$
  - Trong đó k là số lượng nút ở độ sâu d tính đến nút trạng thái đích
- Độ phức tạp bộ nhớ:
  - Số nút lưu trong danh sách OPEN để chờ duyệt
  - = số đỉnh kề với các đỉnh của cây tìm kiếm ở mức d
  - = số đỉnh của cây tìm kiếm ở mức d+1
  - $= O(b^{d+1})$

Tìm kiếm theo chiều sâu

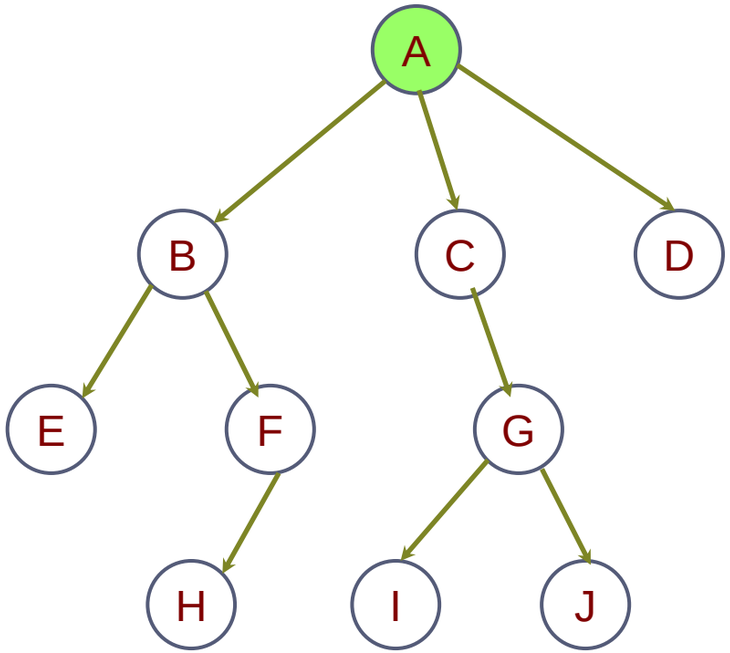
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

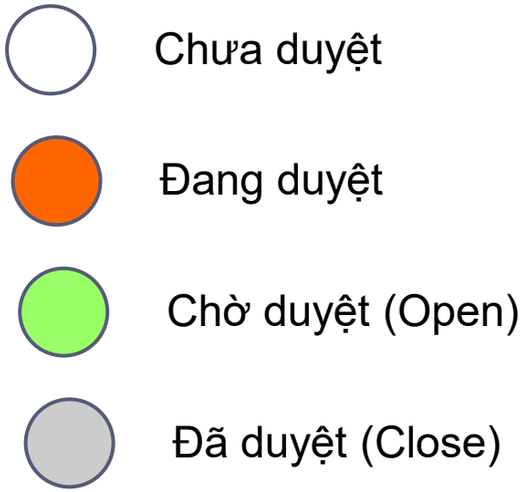




Tìm kiếm theo chiều sâu

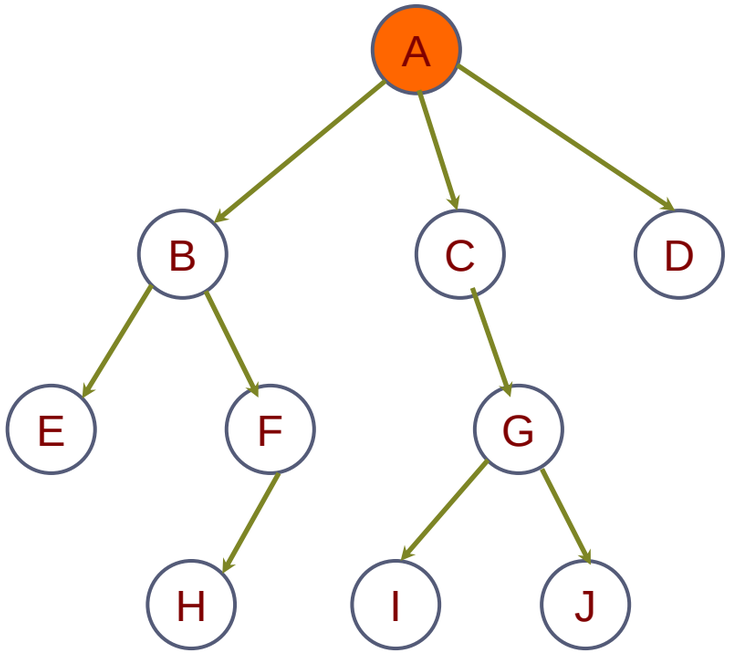
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

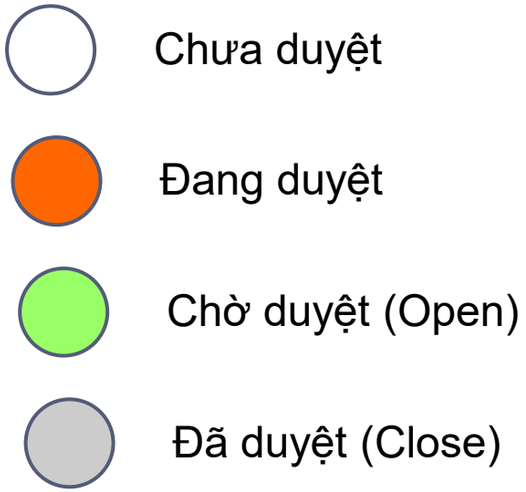




Tìm kiếm theo chiều sâu

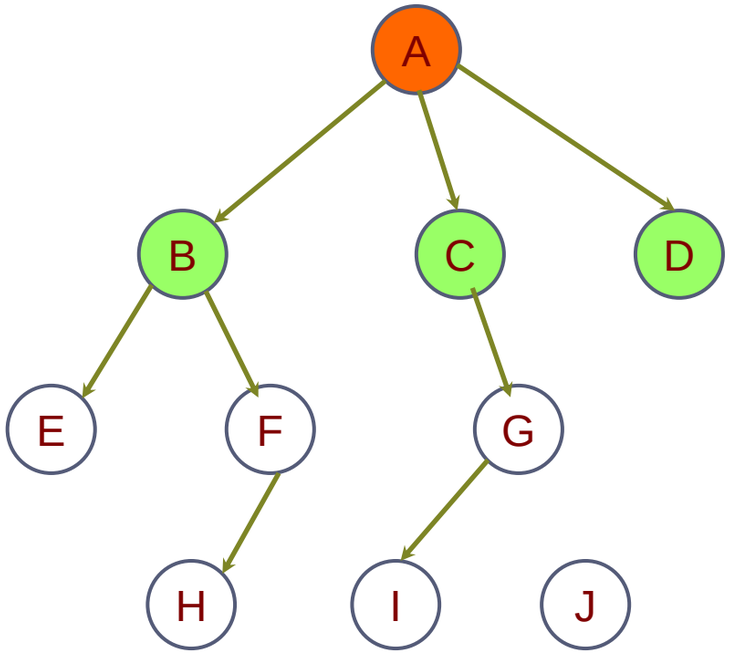
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

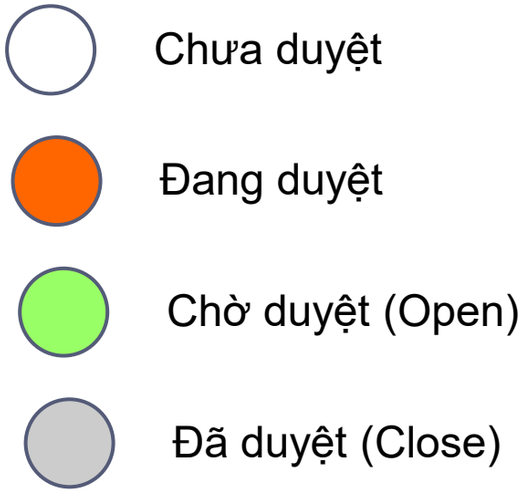




Tìm kiếm theo chiều sâu

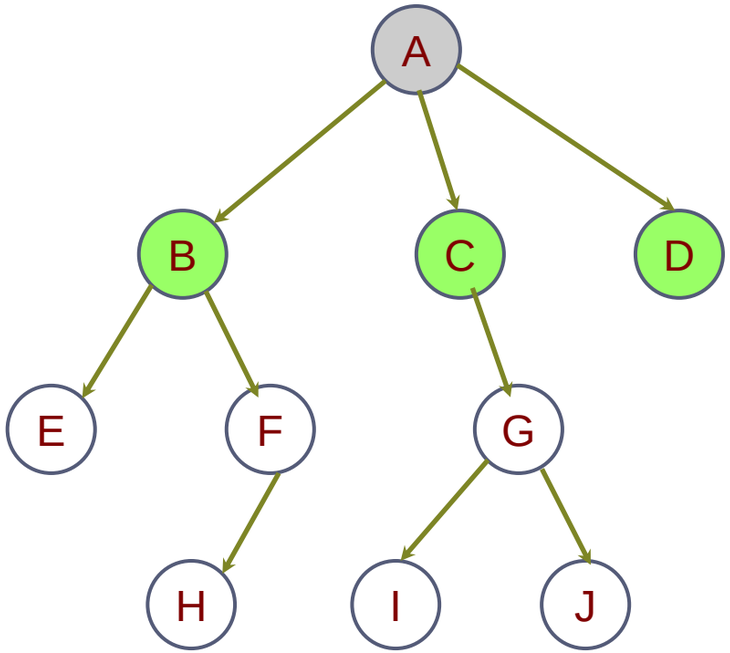
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

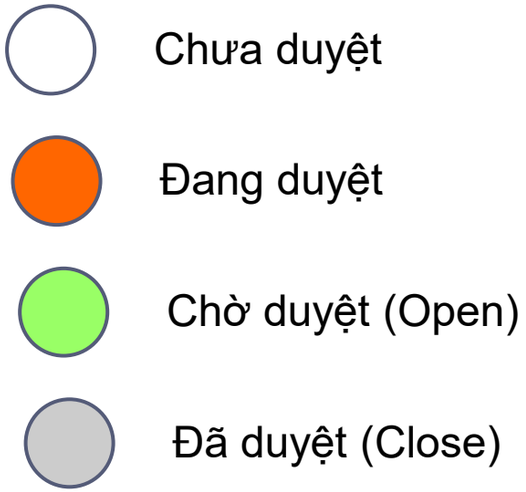




Tìm kiếm theo chiều sâu

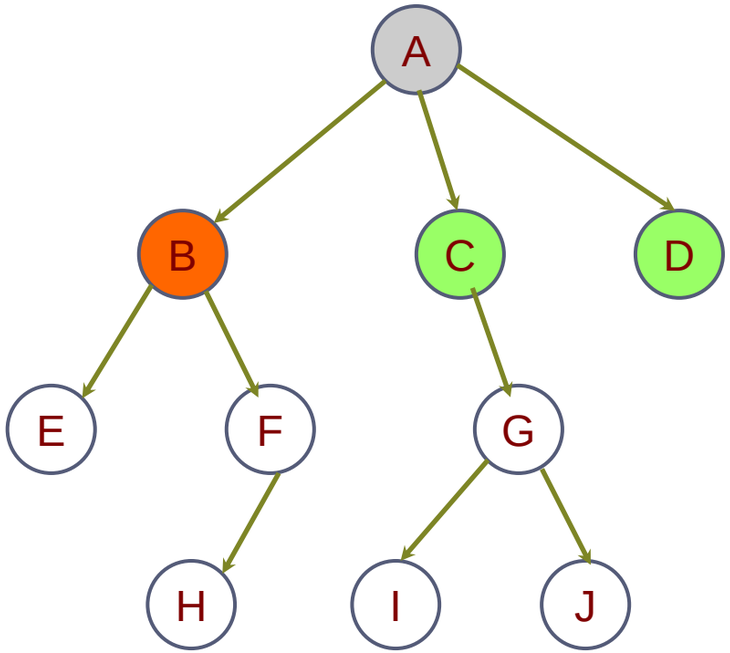
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

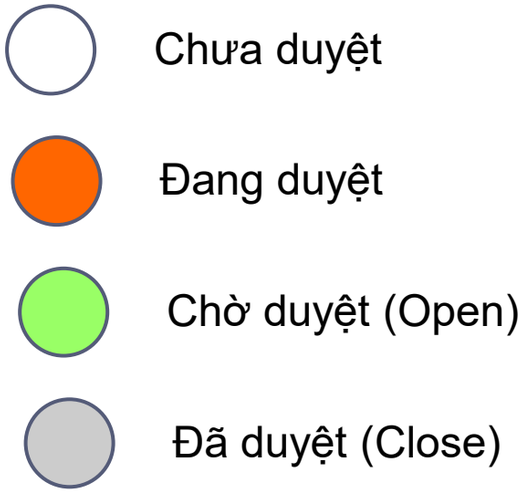




Tìm kiếm theo chiều sâu

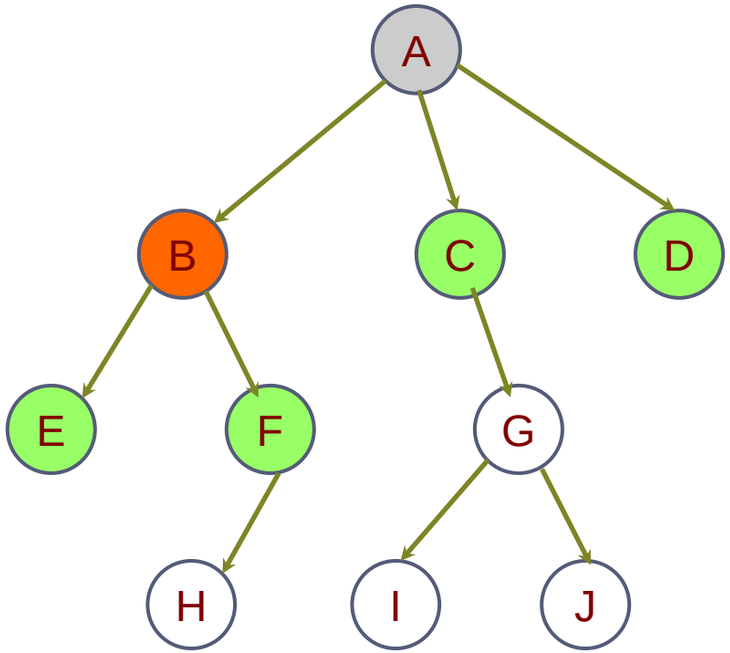
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

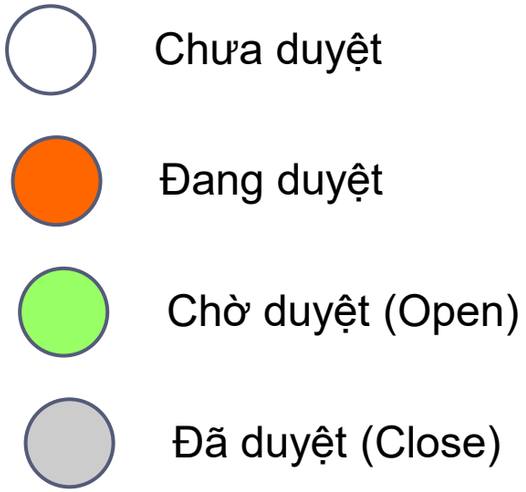




Tìm kiếm theo chiều sâu

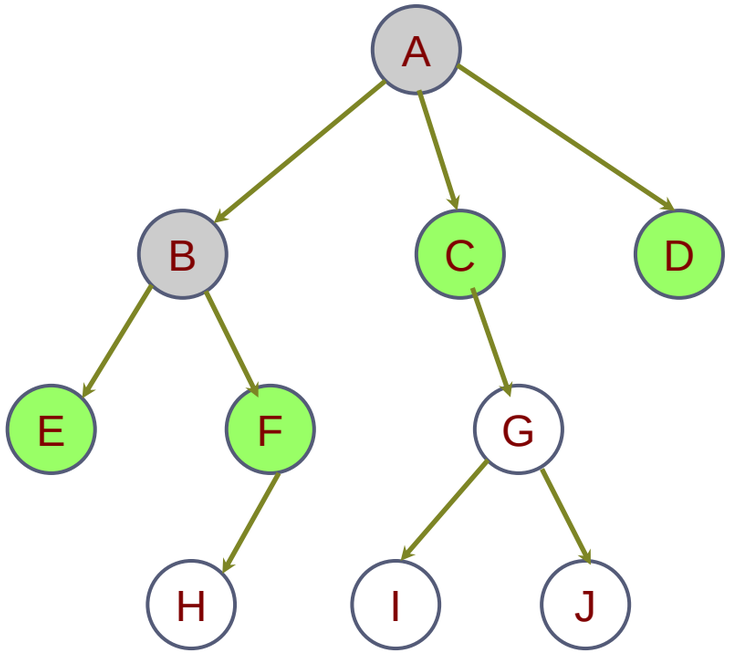
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

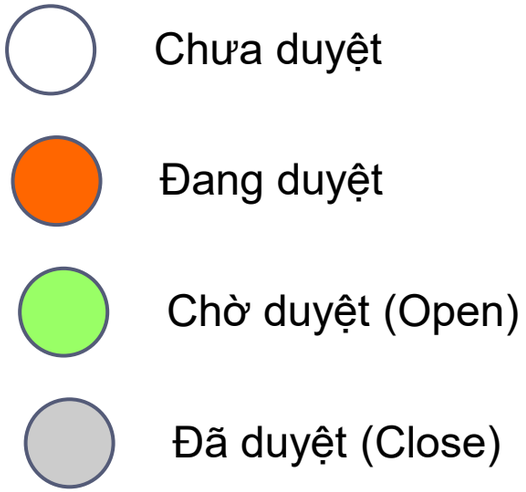




Tìm kiếm theo chiều sâu

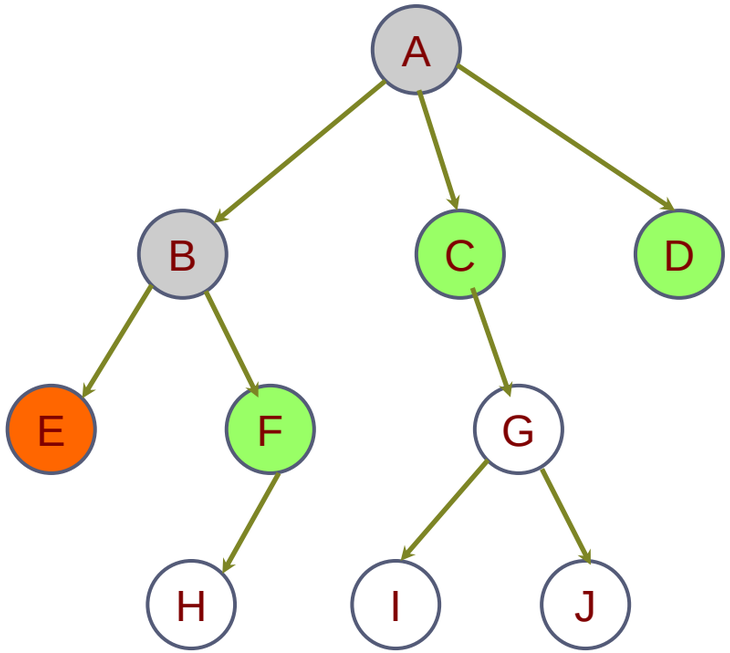
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

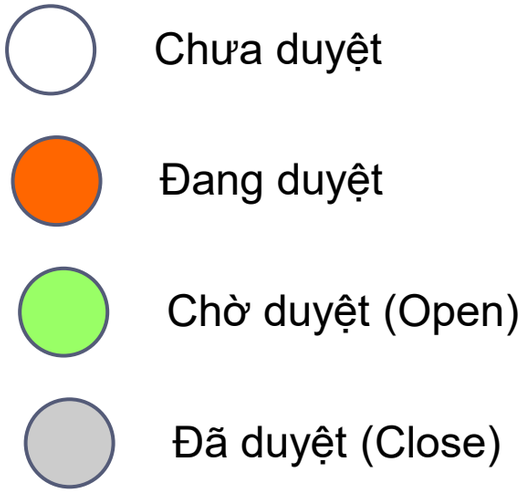




Tìm kiếm theo chiều sâu

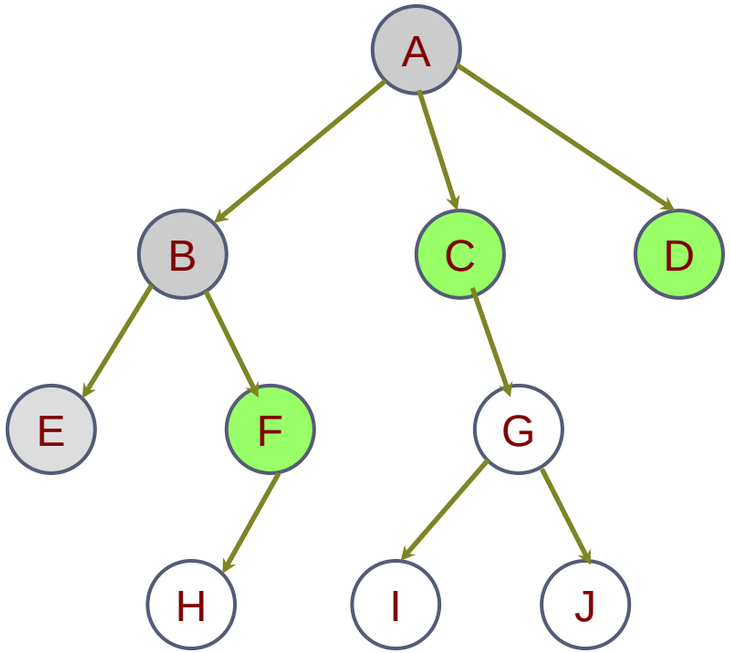
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

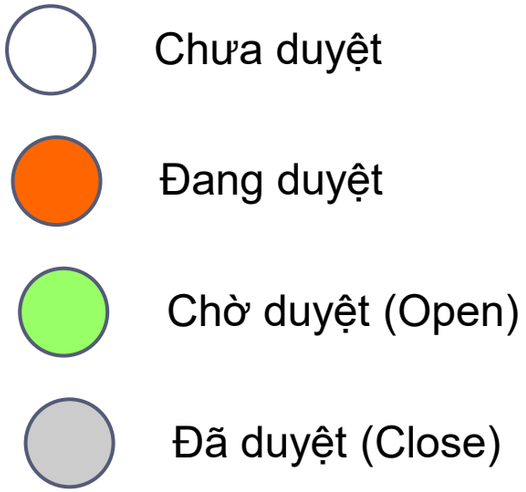




Tìm kiếm theo chiều sâu

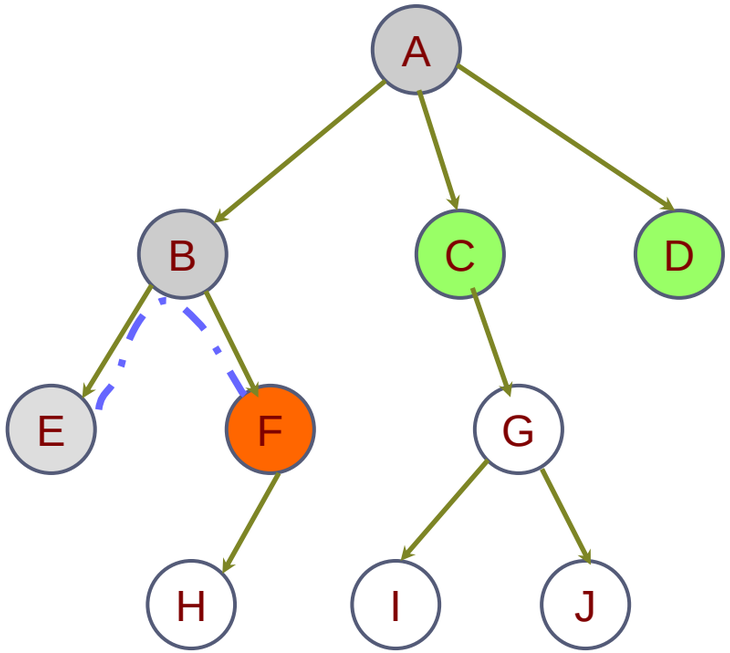
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

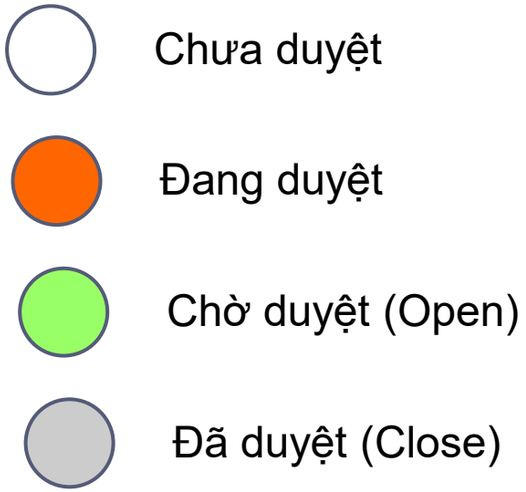




Tìm kiếm theo chiều sâu

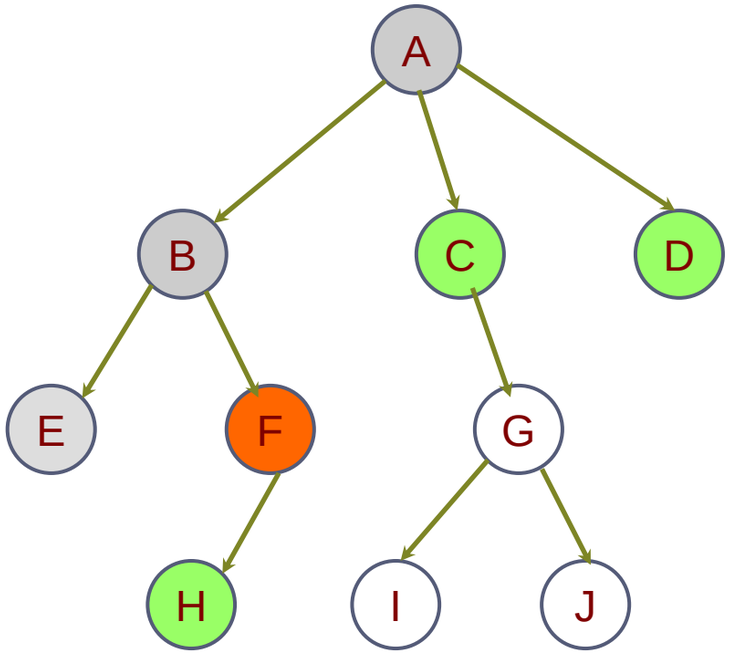
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

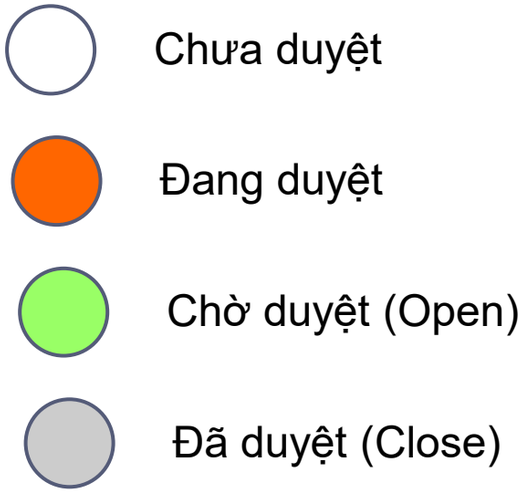




Tìm kiếm theo chiều sâu

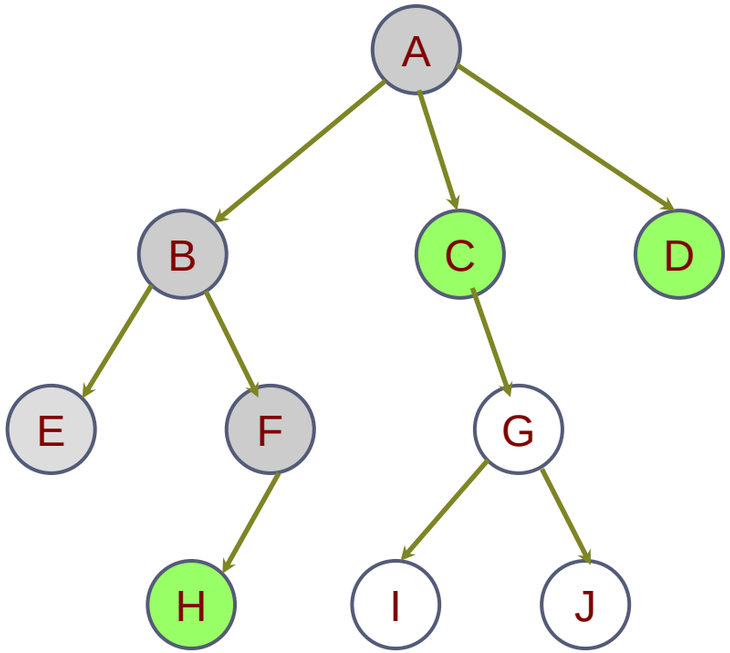
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

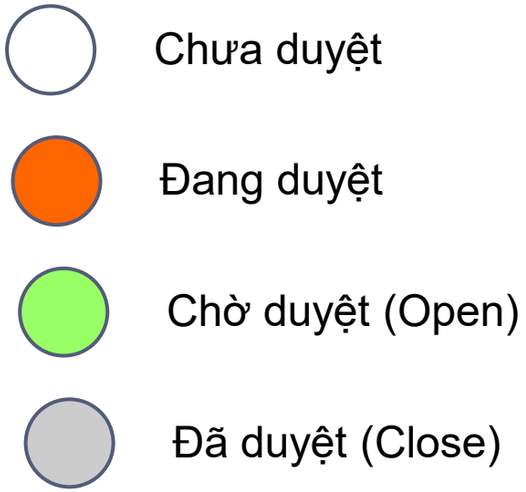




Tìm kiếm theo chiều sâu

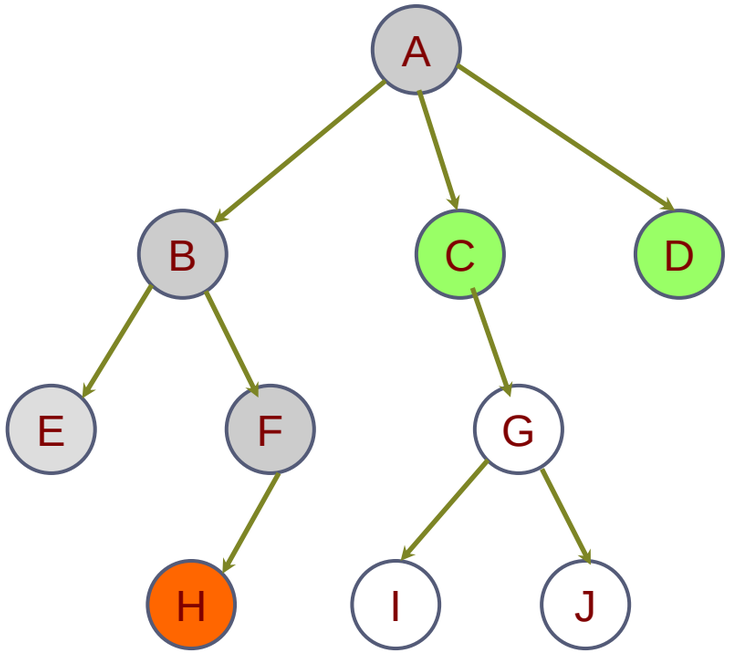
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

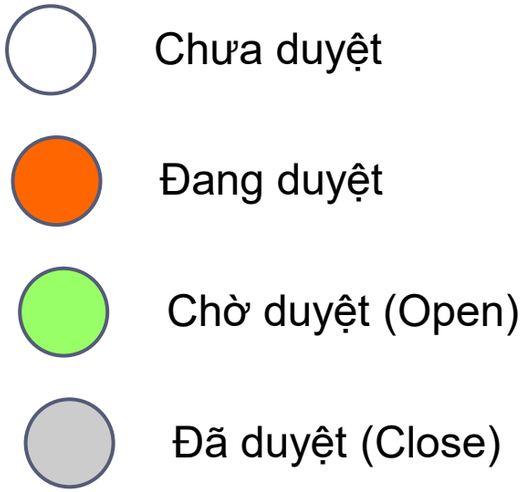




Tìm kiếm theo chiều sâu

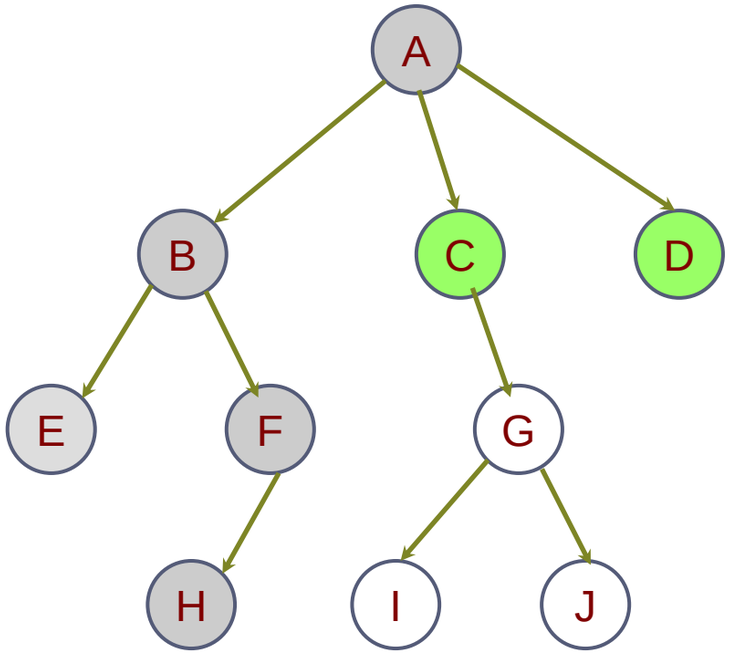
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

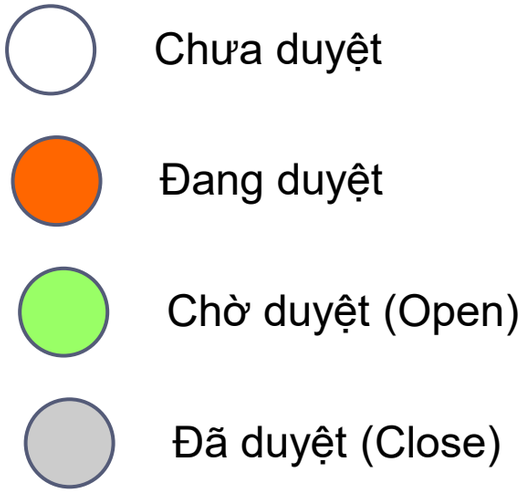




Tìm kiếm theo chiều sâu

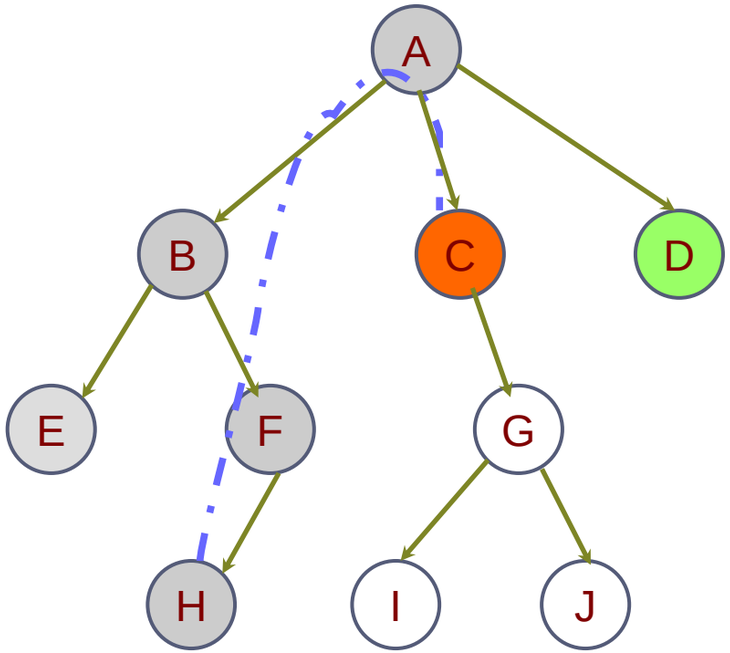
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

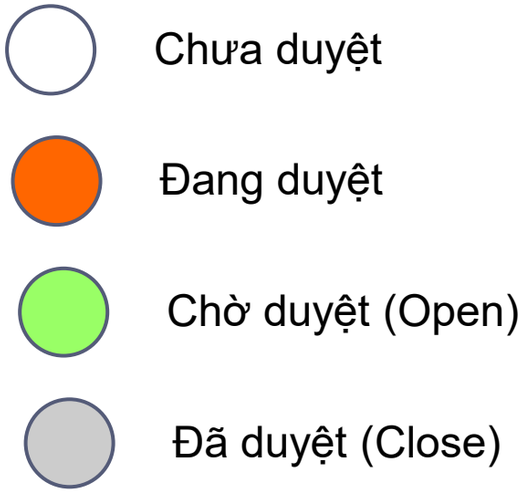




Tìm kiếm theo chiều sâu

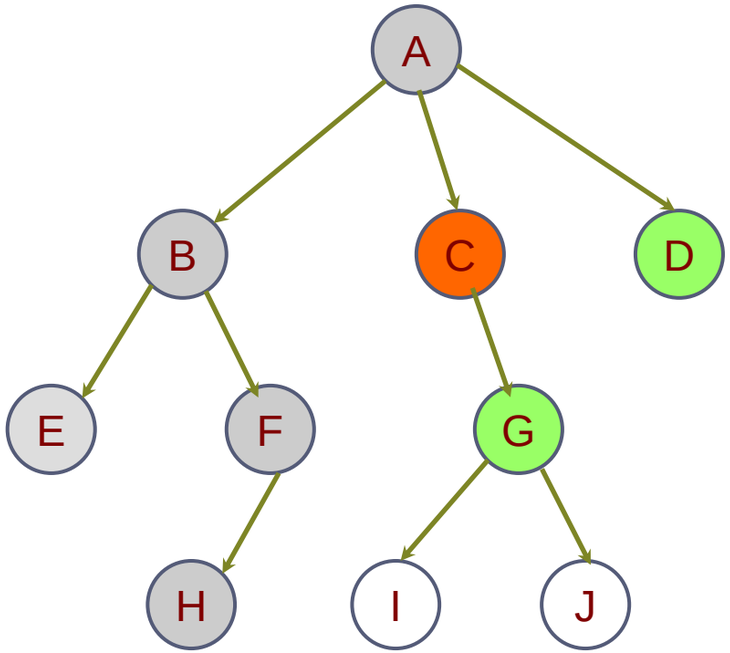
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

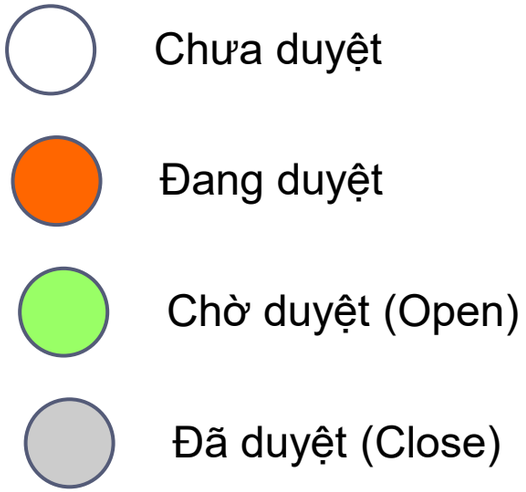




Tìm kiếm theo chiều sâu

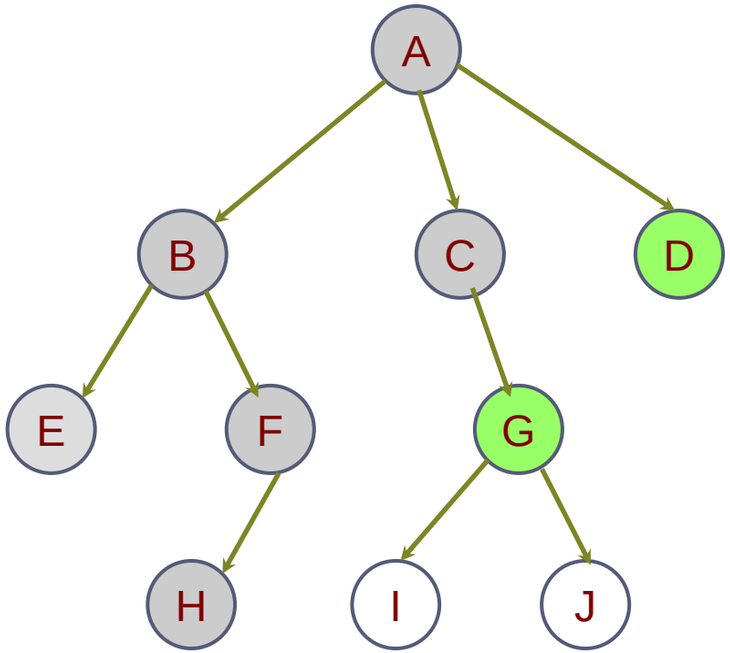
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

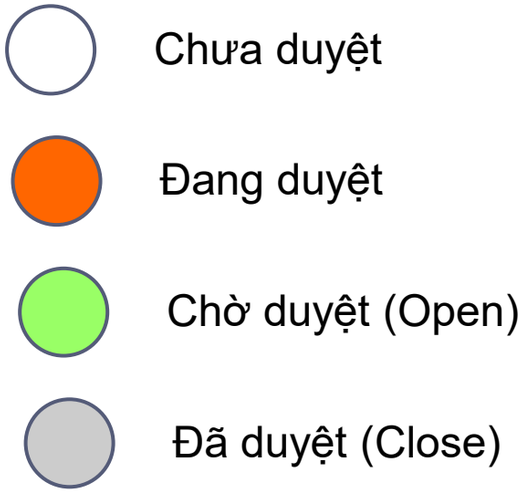




Tìm kiếm theo chiều sâu

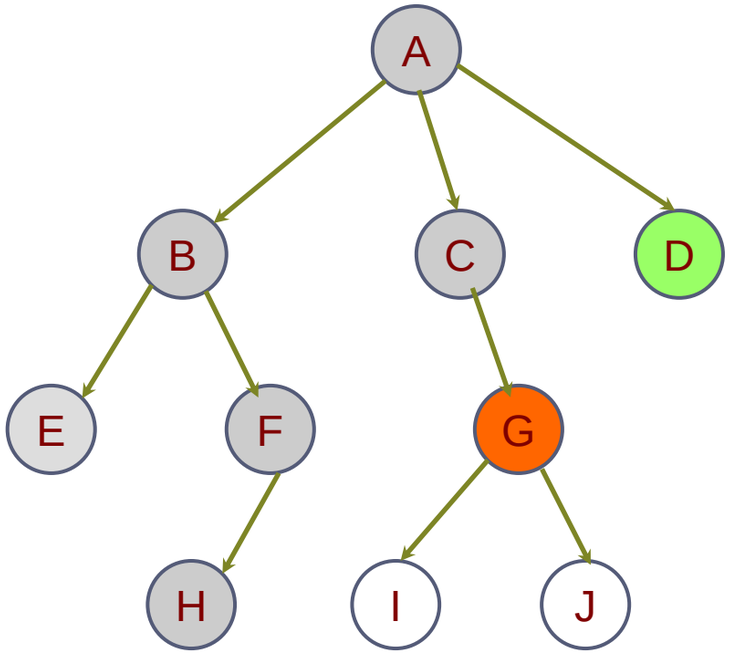
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

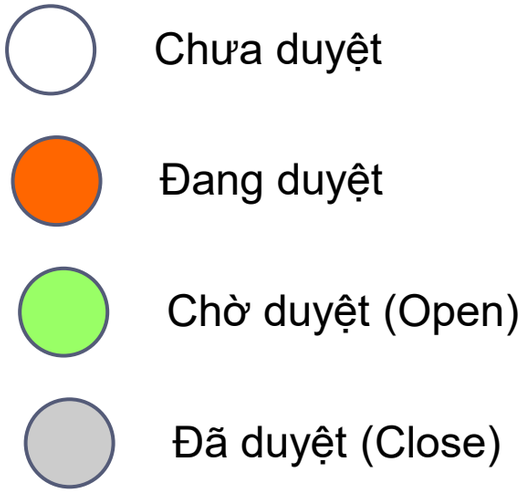




Tìm kiếm theo chiều sâu

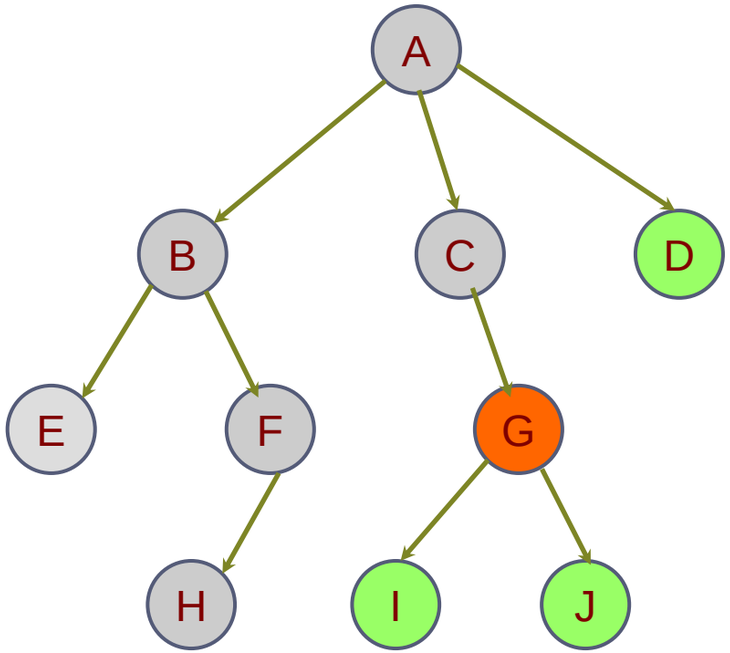
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

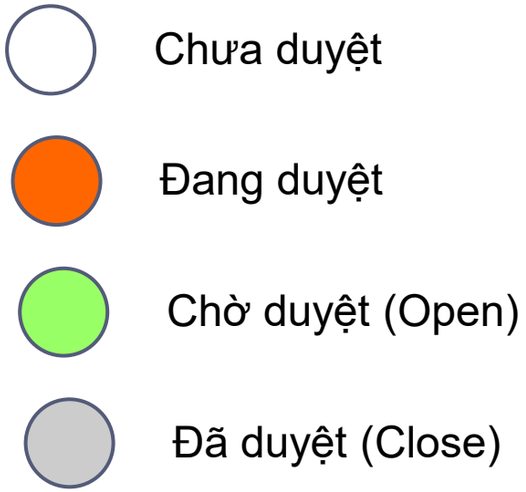




Tìm kiếm theo chiều sâu

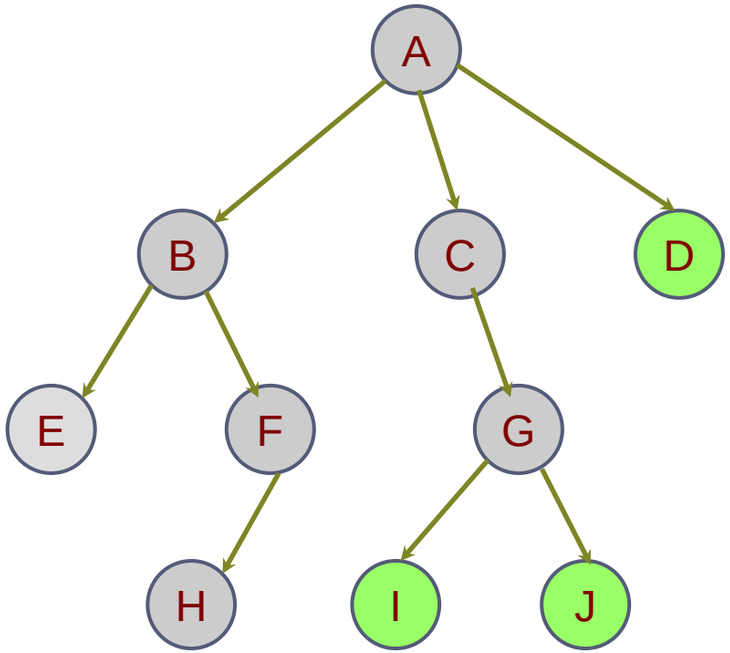
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

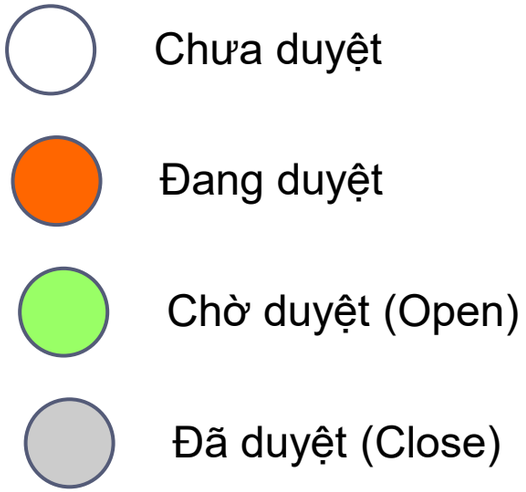




Tìm kiếm theo chiều sâu

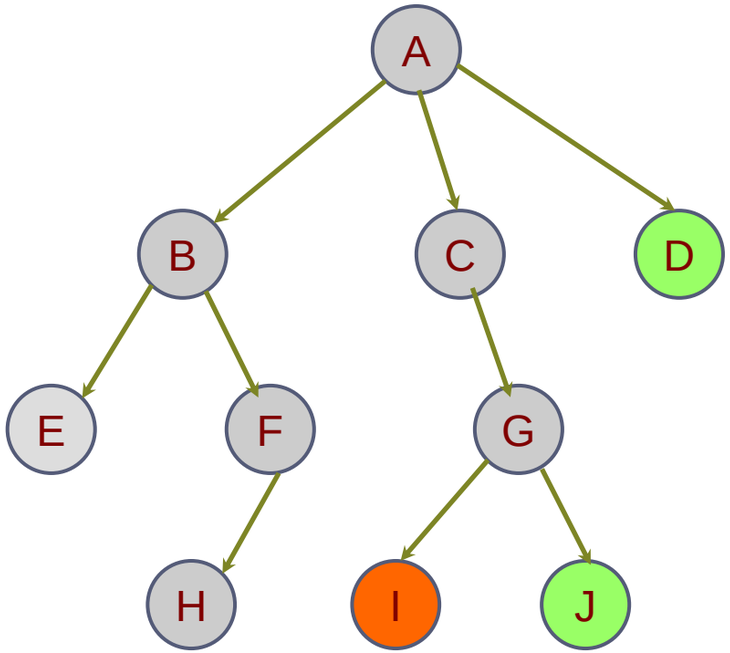
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy

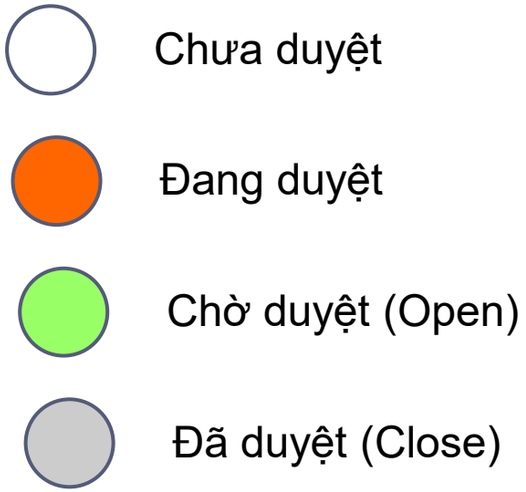




Tìm kiếm theo chiều sâu

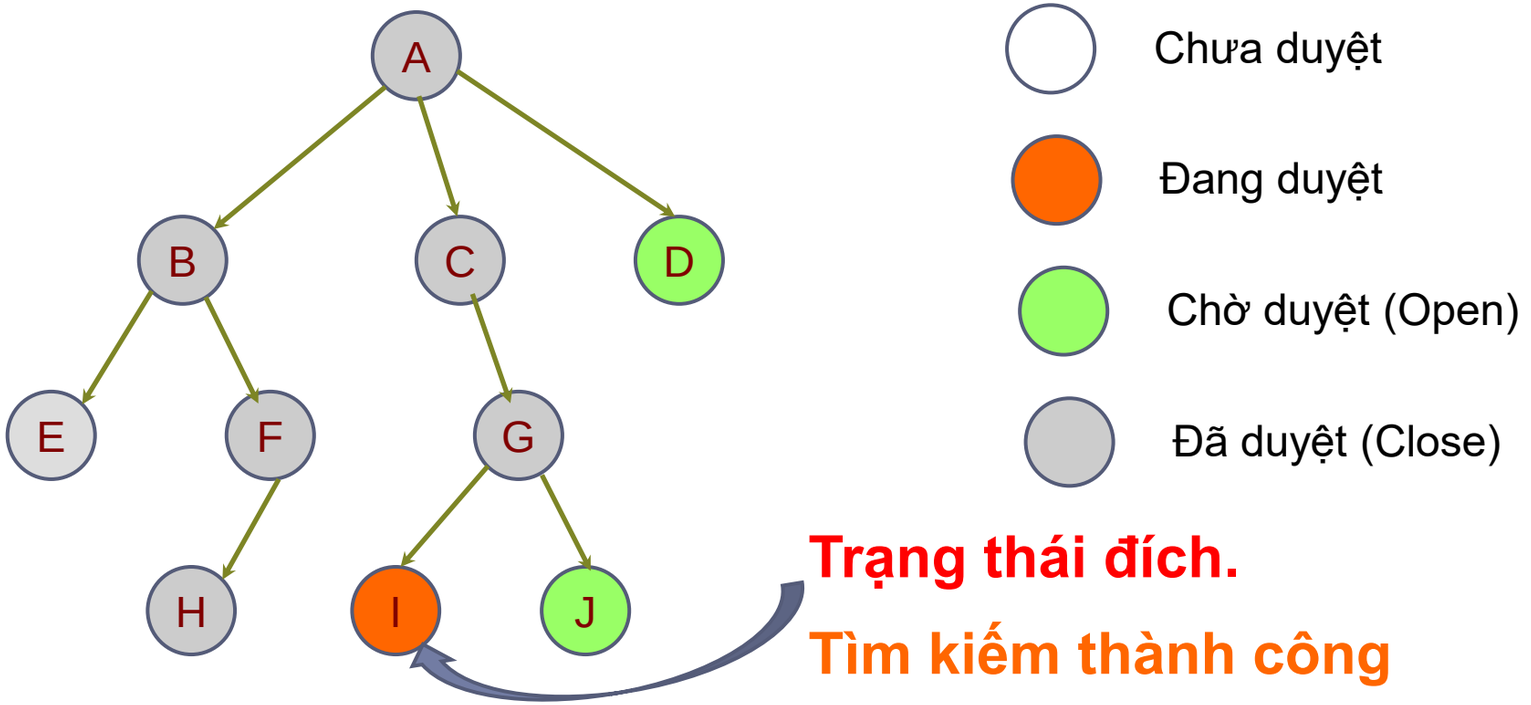
- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy





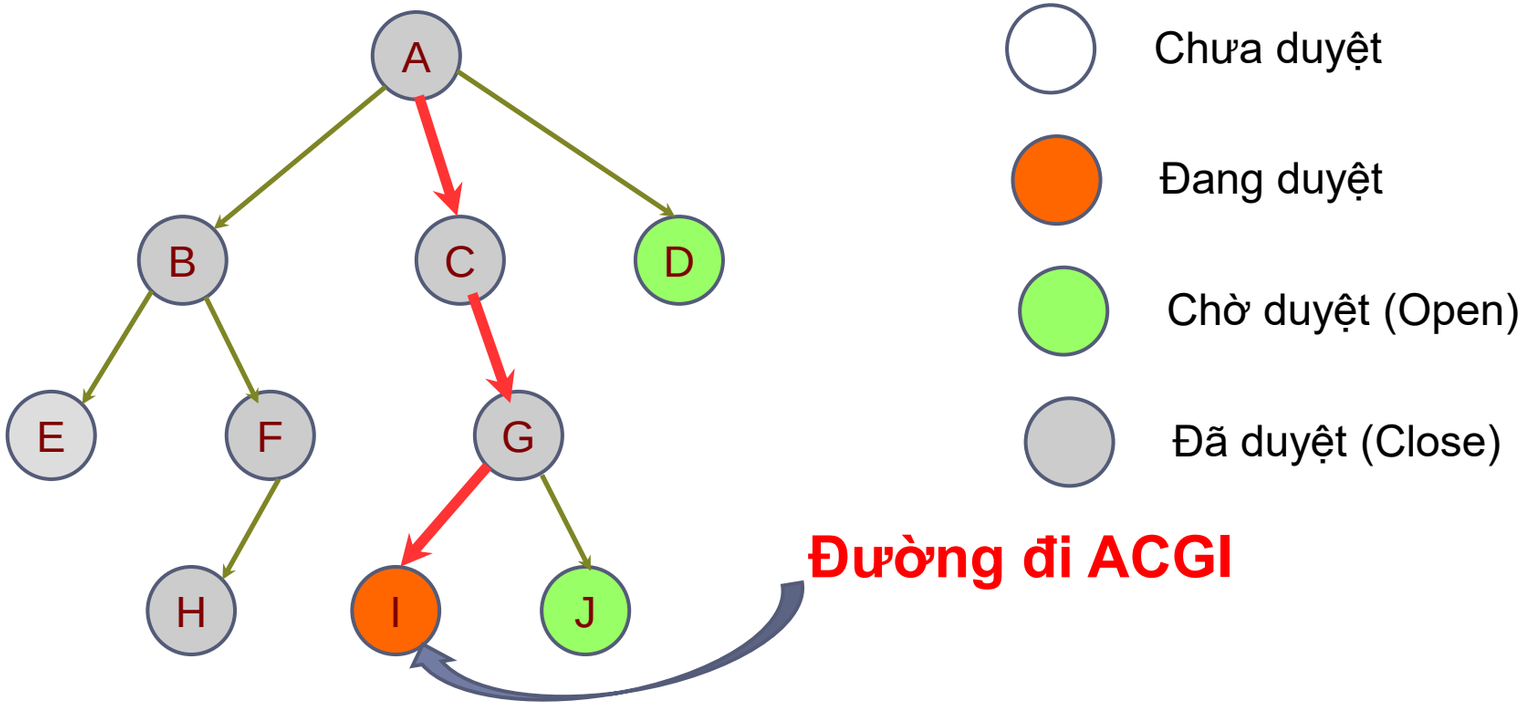
Tìm kiếm theo chiều sâu

- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy



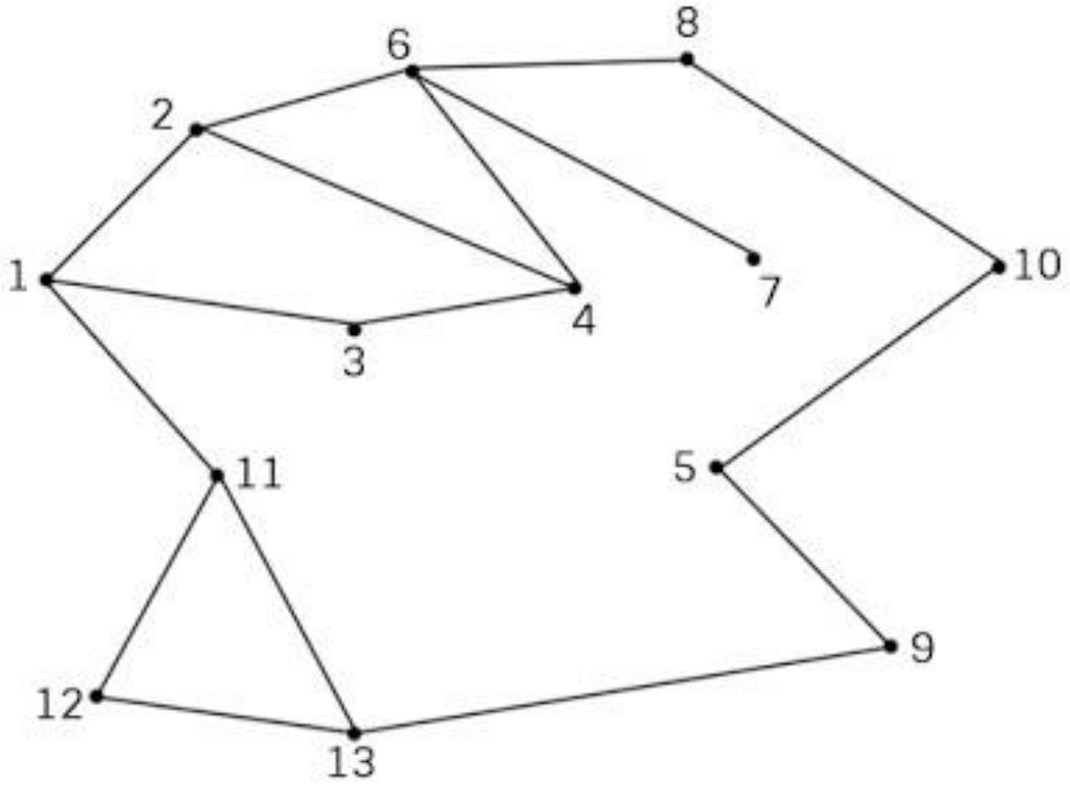
Tìm kiếm theo chiều sâu

- Tập trạng thái chờ duyệt “OPEN”: Stack
- Trạng thái sâu nhất (được đưa vào Open mới nhất) thì được phát triển trước
- Duyệt hết các nút con cháu, … rồi mới đi lên duyệt các nút ở cùng độ sâu nếu chưa tìm thấy



Tìm kiếm theo chiều sâu

- Bài tập: Tìm đường đi và trình tự duyệt các đỉnh
  - Trạng thái đầu: 1
  - Trạng thái đích: 12



Tìm kiếm theo chiều sâu

```
Procedure Depth_First_Search
   Khởi tạo Open = {trạng thái ban đầu};
   while true do
      If (Open rỗng) then
        {thông báo thất bại; stop};
      Loại trạng thái u ở đầu danh sách Open;
      If (u là trạng thái kết thúc) then
         {thông báo thành công; stop};
      Thêm các nút trạng thái kề với u vào đầu danh
      sách Open
End
```

Tìm kiếm theo chiều sâu

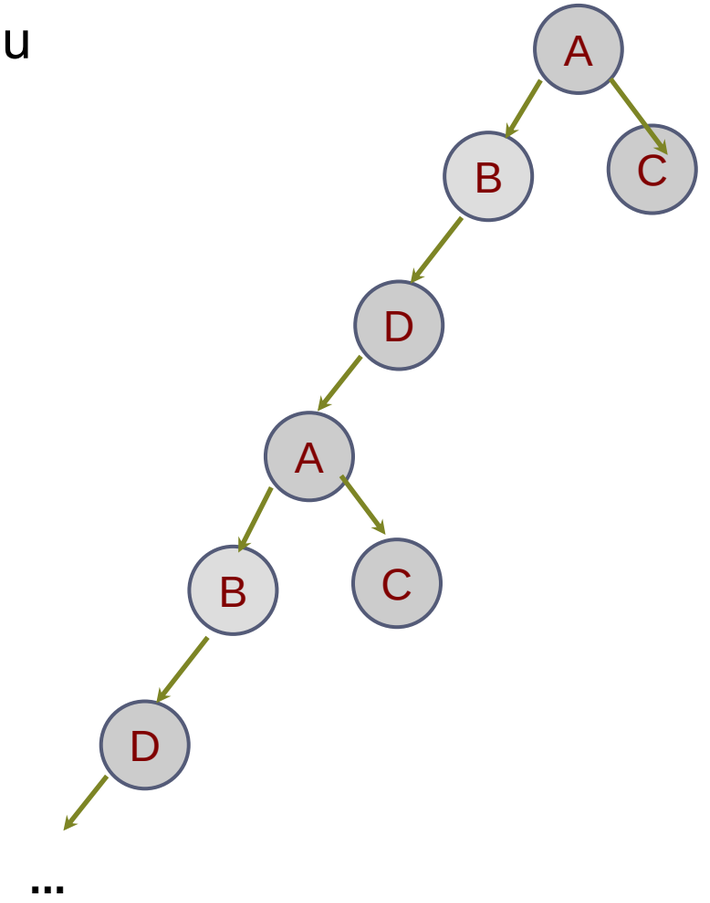
- Độ phức tạp bộ nhớ
  - Chỉ cần lưu các nút con (kề) chưa được phát triển của các đỉnh nằm trên đường đi từ gốc tới đích
  - Số nút cần lưu = b + b + … + b = d.b = O(b.d)
- Độ phức tạp thời gian
  - Trường hợp tốt nhất (nghiệm là đỉnh ngoài cùng bên trái ở mức d):
    - Số nút duyệt = d
  - Trường hợp xấu nhất (nghiệm là đỉnh ngoài cùng bên phải ở mức d)
    - Số nút duyệt = $1 + b + b^2 + \ldots + b^{d-1} + k = O(b^d)$
    - k là vị trí của trạng thái đích ở độ sâu d

So sánh

	Chiều rộng	**Chiều sâu**
Open	FIFO (Queue)	LIFO (Stack)
Hiệu quả	Khi trạng thái đích nằm gần gốc của cây tìm kiếm	Khi trạng thái đích nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác
Độ phức tạp	Tốn nhiều bộ nhớ hơn Có cùng độ phức tạp thời gian về mặt lý thuyết nhưng chậm hơn trong thực tế	
Kết quả	Chắc chắn tìm ra kết quả nếu có	Không tìm ra nghiệm trong không gian lặp hoặc không gian có độ sâu vô hạn

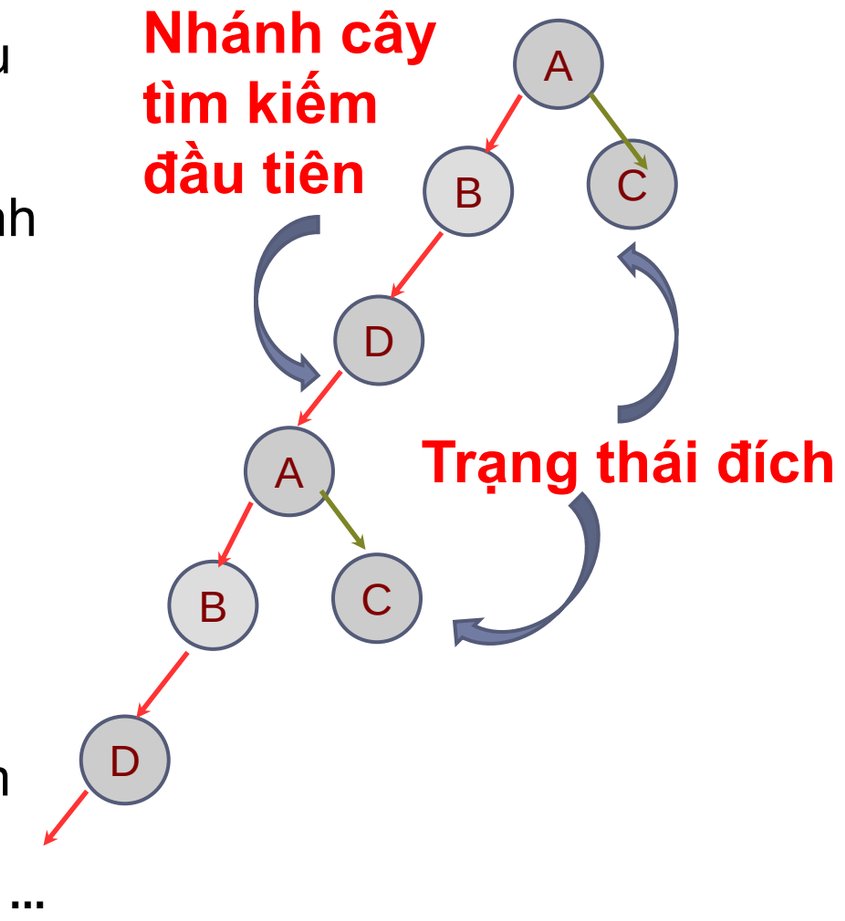
Vấn đề của tìm kiếm theo chiều sâu

- Không gian tìm kiếm có thể có độ sâu vô hạn hoặc lặp trạng thái.



Vấn đề của tìm kiếm theo chiều sâu

- Không gian tìm kiếm có thể có độ sâu vô hạn hoặc lặp trạng thái.
- Khi đó, DFS có thể bị mắc kẹt ở nhánh vô hạn và không tìm ra nghiệm.
- Khắc phục:
  - Chỉ phát triển các nút chưa được phát triển.
    - Thêm biến trạng thái boolean duyệt(u).
  - Hạn chế độ sâu của giải thuật: tìm kiếm theo chiều sâu hạn chế.



Tìm kiếm theo chiều sâu hạn chế

Chỉ tìm kiếm (theo chiều sâu) trong 1 giới hạn độ sâu nào đó. Các nút nằm ở độ sâu sâu hơn không được xét duyệt.

```
Procedure Depth_Limited_Search(l)

begin
    1. Khởi tạo Open = {trạng thái ban đầu u0};
       depth(u0) = 0;
    2. while true do
       2.1 If (Open rỗng) then
           {thông báo thất bại; stop};
       2.2 Loại trạng thái u ở đầu danh sách Open;
       2.3 If (u là trạng thái kết thúc) then
           {thông báo thành công; stop};
       2.4 If (depth(u)<l) then
           for (mỗi trạng thái v kề u) do
              Thêm v vào đầu danh sách Open;
              depth(v) = depth(u) + 1;

end
```

Tìm kiếm sâu lặp

Tìm kiếm sâu lặp (**I**terative **D**eeping **S**earch) còn gọi là tìm kiếm sâu dần, kết hợp được các ưu điểm của tìm kiếm theo bề rộng và tìm kiếm theo độ sau

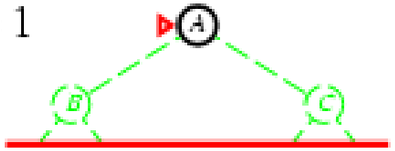
- Là tìm kiếm theo chiều sâu hạn chế với giới hạn độ sâu tăng dần từ 0 tới một mức max.
- Quá trình tìm kiếm:
  - Tìm kiếm theo độ sâu ở mức giới hạn d nào đó (bắt đầu từ d=0)
  - Nếu không tìm thấy nghiệm, tìm kiếm theo độ sâu ở mức giới hạn độ sâu d+1

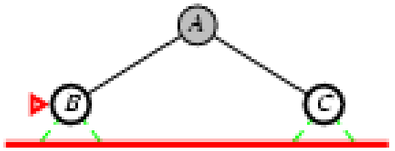
Tìm kiếm sâu lặp

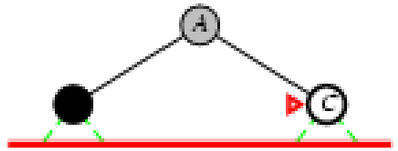
Limit = 0

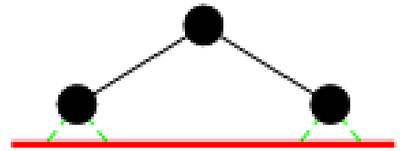
Tìm kiếm sâu lặp

Limit = 1



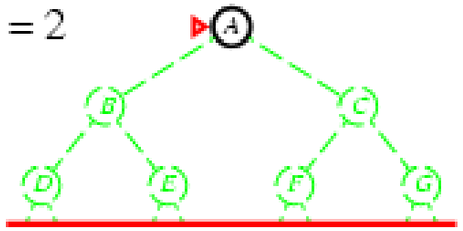


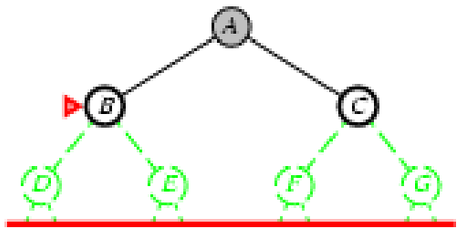


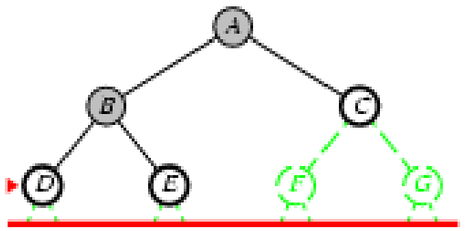


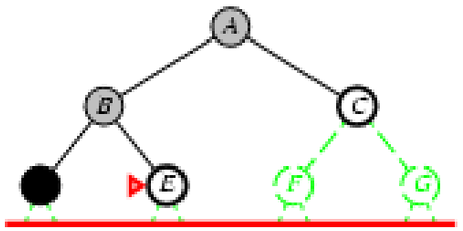
Tìm kiếm sâu lặp

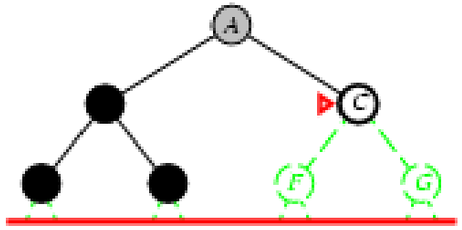
Limit = 2

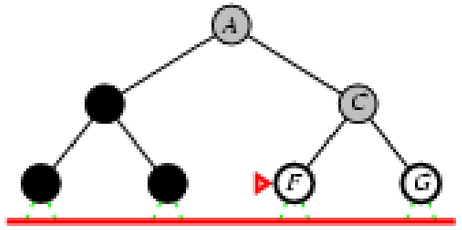


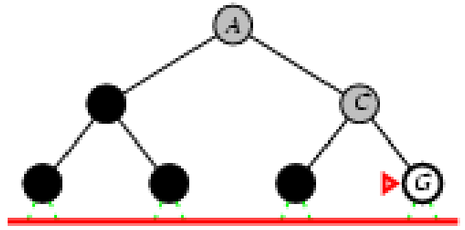


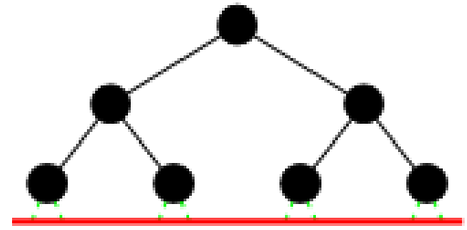




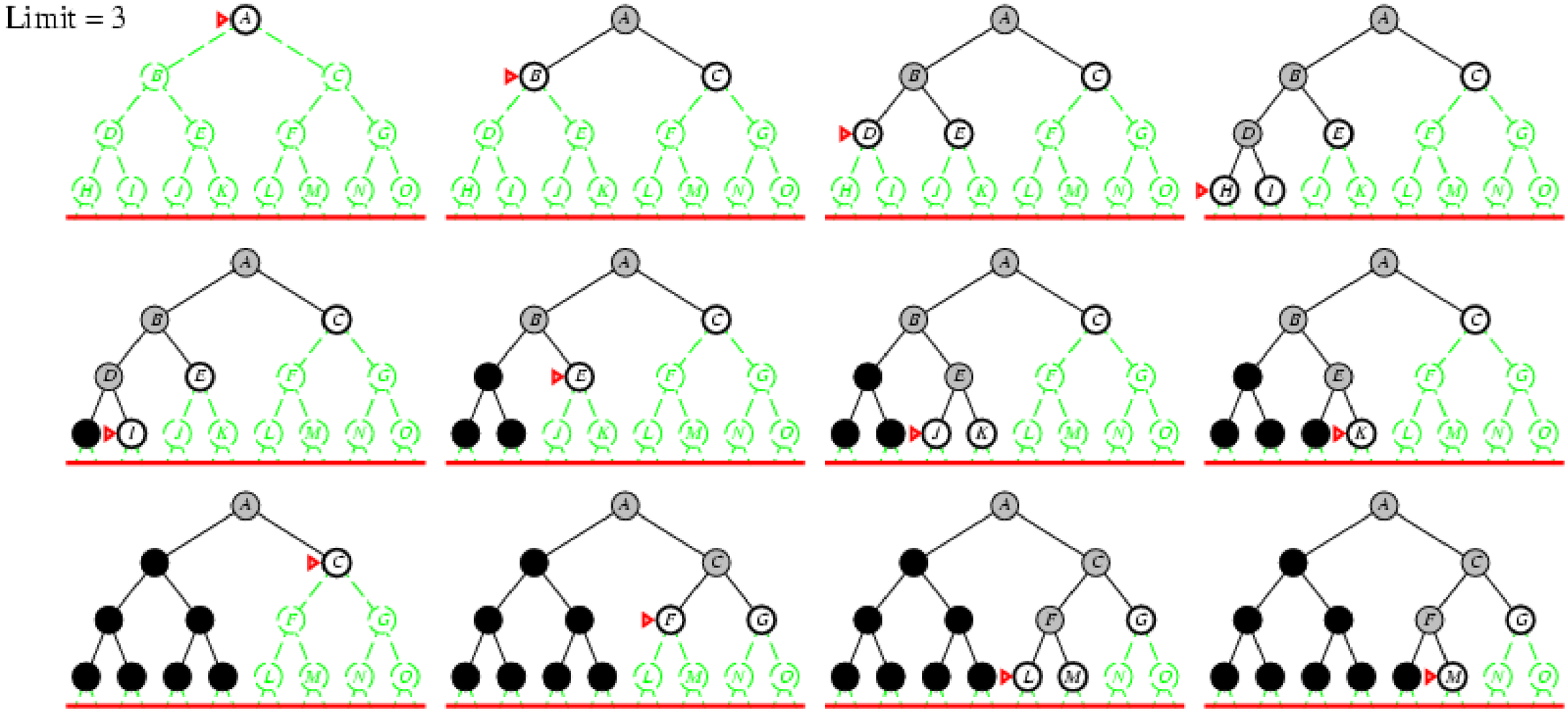








Tìm kiếm sâu lặp



Tìm kiếm sâu lặp

```
Procedure Depth_Deepening_Search(max)
begin
   for (l = 0 to max) do{
      Depth_Limited_Search(l);
      if (thành công) then exit;
   }
end
```

Tìm kiếm sâu lặp

- Nếu nghiệm có độ sâu d và cây có hệ số phân nhánh b thì ta đã thực hiện các thủ tục DLS(0), DLS(1), …, DLS(max)
- Độ phức tạp thời gian
  - Số đỉnh phải phát triển ở mức 0 là 1, lặp max+1 lần
  - Số đỉnh phải phát triển ở mức 1 là b, lặp max lần
  - Số đỉnh phải phát triển ở mức 2 là b^2, lặp d=max-1 lần
  - …
  - Số đỉnh phải phát triển ở mức max là b^{max}, lặp 1 lần
  - Số đỉnh duyệt = $(max+1).1 + max.b + (max-1).b^2 + \dots + 2.b^{max-1} + 1.b^{max} = O(b^{max})$
  - Nếu d<= max thì số đỉnh duyệt là $O(b^d)$

Tìm kiếm sâu lặp

Bài tập: Cho hệ số phân nhánh b = 5, số lượng nút d = 5, tính số đỉnh cần duyệt của tìm kiếm theo chiều sâu (DFS) và tìm kiếm theo chiều sâu lặp (IDS).

So sánh các giải thuật tìm kiếm mù

	Chiều rộng	**Chiều sâu**	**Chiều sâu hạn chế**	**Sâu lặp**
Hoàn thành	Có	Không	Không	Không
Thời gian	$O(b^d)$	$O(b^d)$	$O(b^l)$	$O(b^{max})$
Không gian	$O(b^{d+1})$	O(b.d)	O(b.l)	O(b.max)

- Hoàn thành: là khả năng luôn tìm thấy nghiệm nếu có.
  - Chiều sâu: không tìm thấy nếu không gian lặp trạng thái hoặc có chiều sâu vô hạn
  - Chiều sâu hạn chế: không tìm thấy nếu nghiệm nằm sâu hơn giới hạn
  - Sậu lặp: không tìm thấy nếu nghiệm nằm sâu hơn độ sâu tìm kiếm tối đa

Tìm kiếm với tri thức bổ sung

- Tìm kiếm dựa trên các thông tin đánh giá của các trạng thái
  - **Hàm đánh giá**
- Hiệu quả

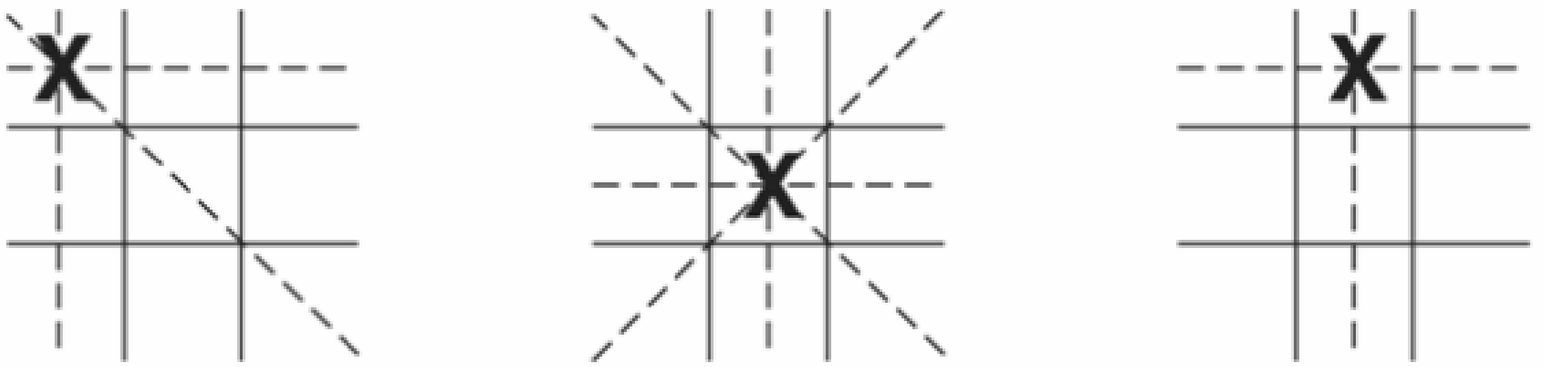
Hàm đánh giá

- Là hàm ước lượng, đánh giá mức độ tốt/xấu, khả năng về đích của mỗi trạng thái
- Dựa trên kinh nghiệm
- Với 1 trạng thái u:
  - g(u) là chi phí đã đi từ xuất phát (thông tin **quá khứ,** đã xác định)
  - h(u) là chi phí còn lại để đi tới đích (thông tin **tương lai** hay **heuristic,** ước lượng). Giá trị này càng nhỏ thì càng tốt
  - g(u) + h(u) là tổng chi phí đi từ trạng thái xuất phát tới đích có qua trạng thái đó (thông tin **chung,** ước lượng). Giá trị này càng nhỏ thì càng tốt

Hàm đánh giá

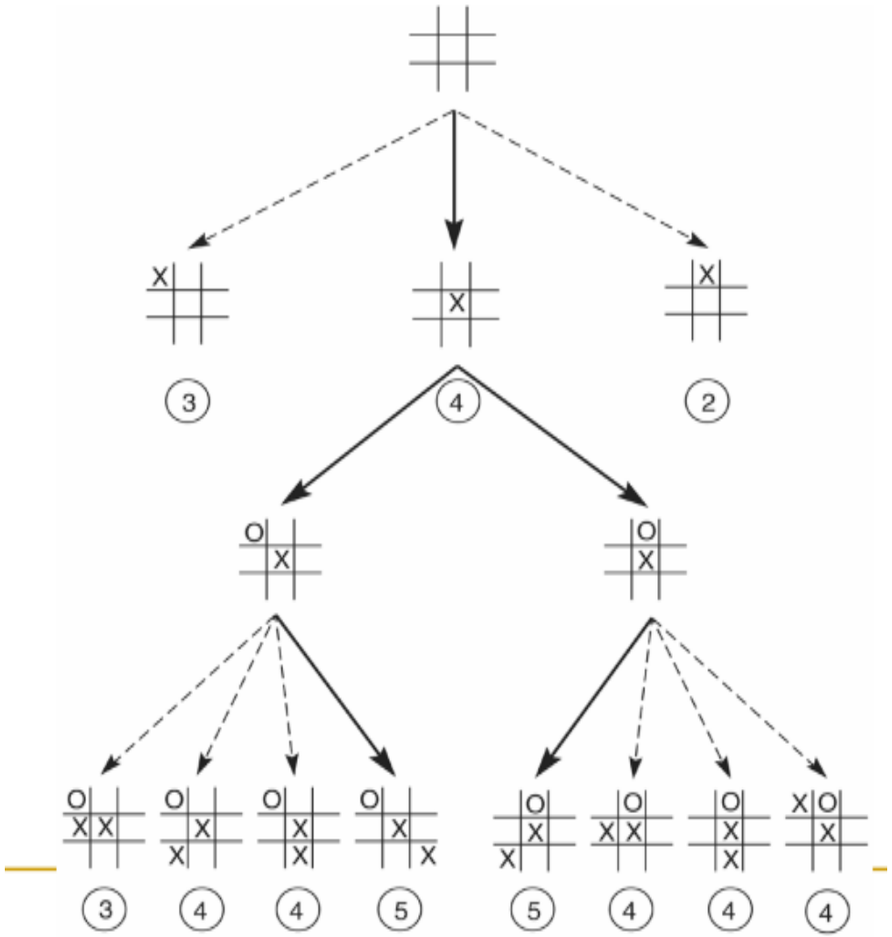
- Có thể chỉ khai thác 1 hoặc cả 2 thông tin đánh giá về quá khứ, tương lai
  - f(u) = g(u) + h(u)
  - f(u) = g(u)
  - f(u) = h(u)
- Các kĩ thuật TK sử dụng hàm đánh giá h(u) gọi chung là TK kinh nghiệm - heuristic search
- Các kĩ thuật TK tối ưu sử dụng hàm đánh giá f(u)=g(u)+h(u)
- Hàm đánh giá càng tốt (sát với thực tế) thì tìm kiếm càng hiệu quả

Phép đo heuristic



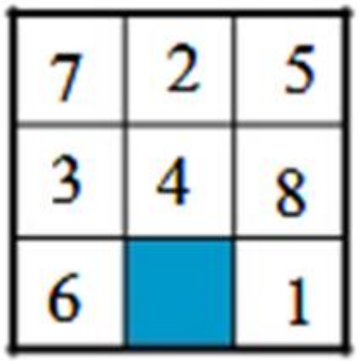
- Heuristic h(u): số đường thẳng đi qua các quân cờ của ta mà không bị chặn bởi đối phương
- Là số nước ăn hay khả năng thắng
- h(u) càng lớn càng tốt

Phép đo heuristic

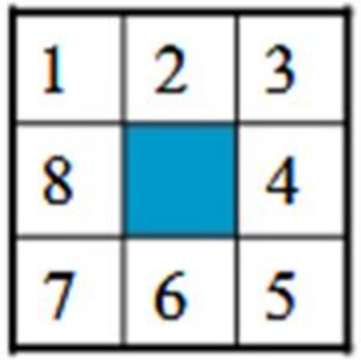


Không gian trạng thái của trò chơi tic-tac-toe

Phép đo heuristic



Trạng thái u

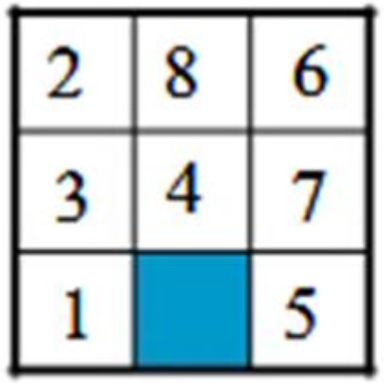


Goal

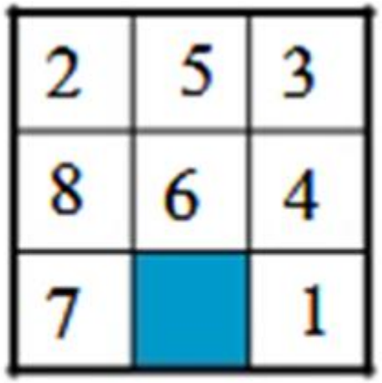
- Cách 1: tổng số ô sai khác so với trạng thái đích
  - h1(u) =7
- Cách 2: tổng số bước dịch chuyển để đưa các ô sai khác về trạng thái đích (tổng khoảng cách Manhattan)
  - h2(u) = 2+0+2+3+1+2+1+4=15
- h(u) càng nhỏ càng tốt

Phép đo heuristic

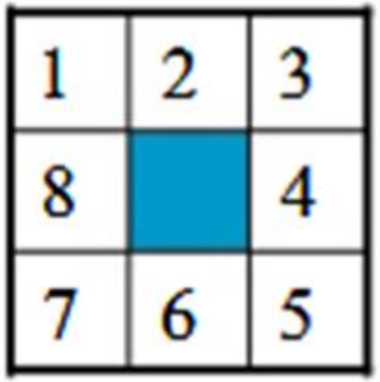
- Tính giá trị heuristic h1, h2 cho các trạng thái sau



h = ?



h = ?



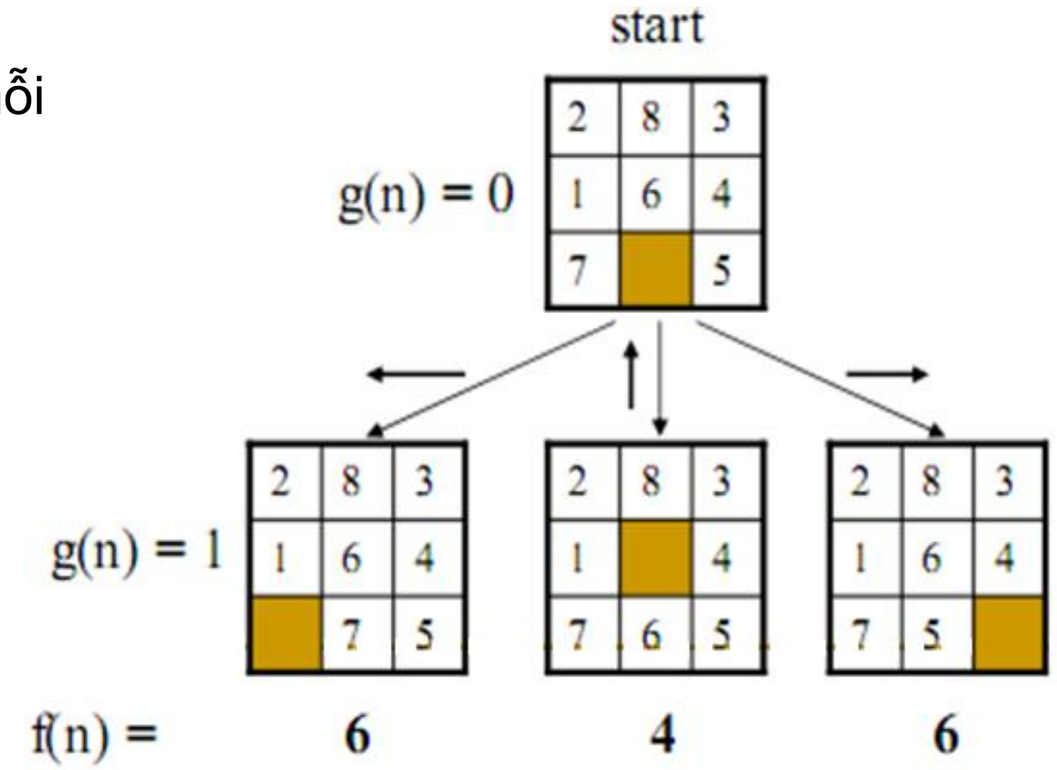
goal

Phép đo heuristic

- Thông thường, h(u) biểu diễn ước lượng chi phí để đi từ u tới đích
- Khi h(u) biểu diễn ước lượng khả năng thắng trong game (như tic-tac-toe) thì ta sử dụng - h(u)
- Một ước lượng h(u) là chấp nhận được nếu nếu với mọi u, $h(u) \leq h^*(u)$, trong đó $h^*(u)$ là chi phí thực sự để đi từ u tới đích
  - Một ước lượng chấp nhận được có xu thế lạc quan, không bao giờ đánh giá quá cao về chi phí thực hiện
- Ví dụ
  - Trong các ví dụ trước: h(u) là chấp nhận được
  - Trong bài toán người du lịch, h(u) có thể là khoảng cách đường chim bay từ u tới đích. Khi đó, h(u) là chấp nhận

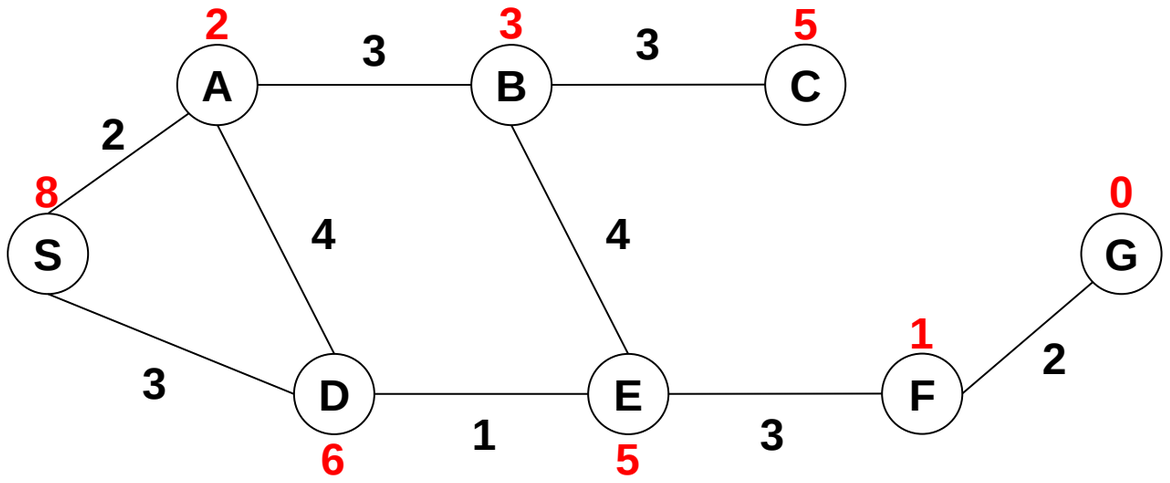
Hàm đánh giá (sửa lại hình)

- **Bài toán tic-tac-toe**
- **Quá khứ**
  - g(u) = khoảng cách thực sự từ trạng thái đầu tới u
    - = số bước đã đi (chi phí cho mỗi bước đi là 1)
- **Tương lai**
  - h1(u) = khoảng cách từ u tới đích
    - = số ô còn sai khác
- **Hàm đánh giá**
  - f(u) = g(u) + h1(u)



Hàm đánh giá

- Bài toán người du lịch: tìm đường đi ngắn nhất từ 1 thành phố ban đầu đến 1 thành phố đích
- Hàm đánh giá
  - Quá khứ: g(u) = khoảng cách đã đi từ thành phố xuất phát
  - Tương lai: h(u) = khoảng cách theo đường chim bay từ 1 thành phố đến thành phố đích



Tìm kiếm với hàm đánh giá

TK kinh nghiệm (TK heuristic): sử dụng hàm đánh giá f(u) = h(u)

- TK tốt nhất đầu tiên = TK bề rộng + h(u)
- TK leo đồi = TK chiều sâu + h(u)

Tìm kiếm tối ưu: sử dụng hàm đánh giá f(u) = g(u)+h(u)

- TK A* = TK tốt nhất đầu tiên + f(u)
- TK nhánh cận = TK leo đồi + f(u)
- ...

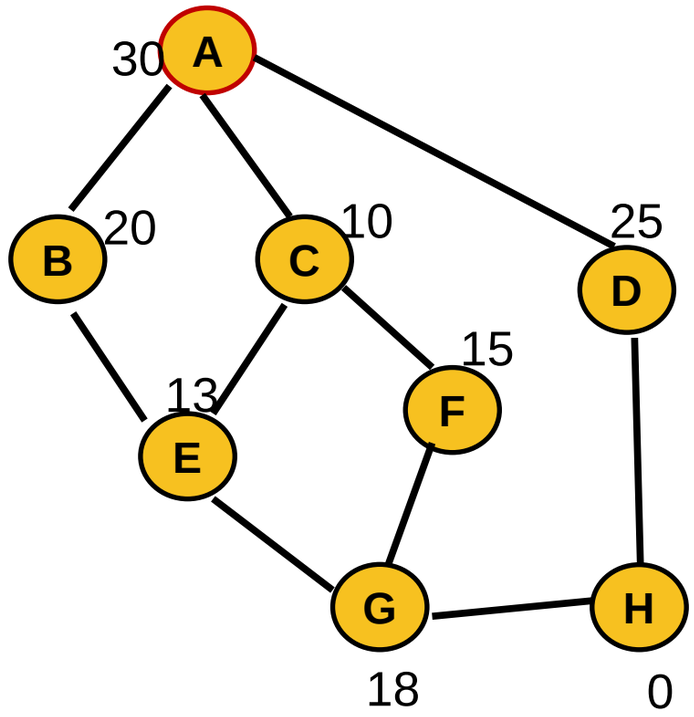
Tìm kiếm kinh nghiệm

Tìm kiếm tốt nhất đầu tiên

- Là TK theo bề rộng được định hướng bởi hàm đánh giá h(u)
- OPEN: tập các đỉnh chờ phát triển được sắp xếp theo thứ tự tăng dần của hàm đánh giá
- Tại mỗi bước của tìm kiếm:
  - Chọn đỉnh u trong OPEN có giá trị hàm đánh giá nhỏ nhất để duyệt
  - Đưa các đỉnh kề của u vào OPEN sao cho OPEN được sắp xếp theo thứ tự **tăng dần của hàm đánh giá**

Tìm kiếm tốt nhất đầu tiên

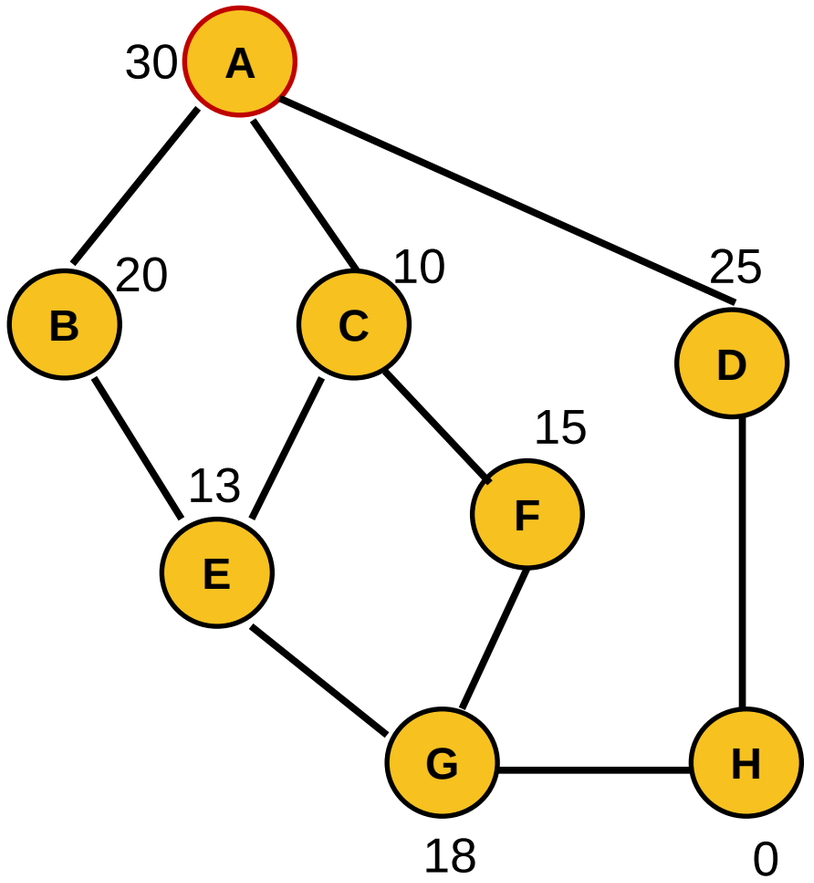
Ví dụ: tìm đường đi ngắn nhất từ A → H



Giá trị gắn với mỗi nút là đánh giá heuristic h(u)

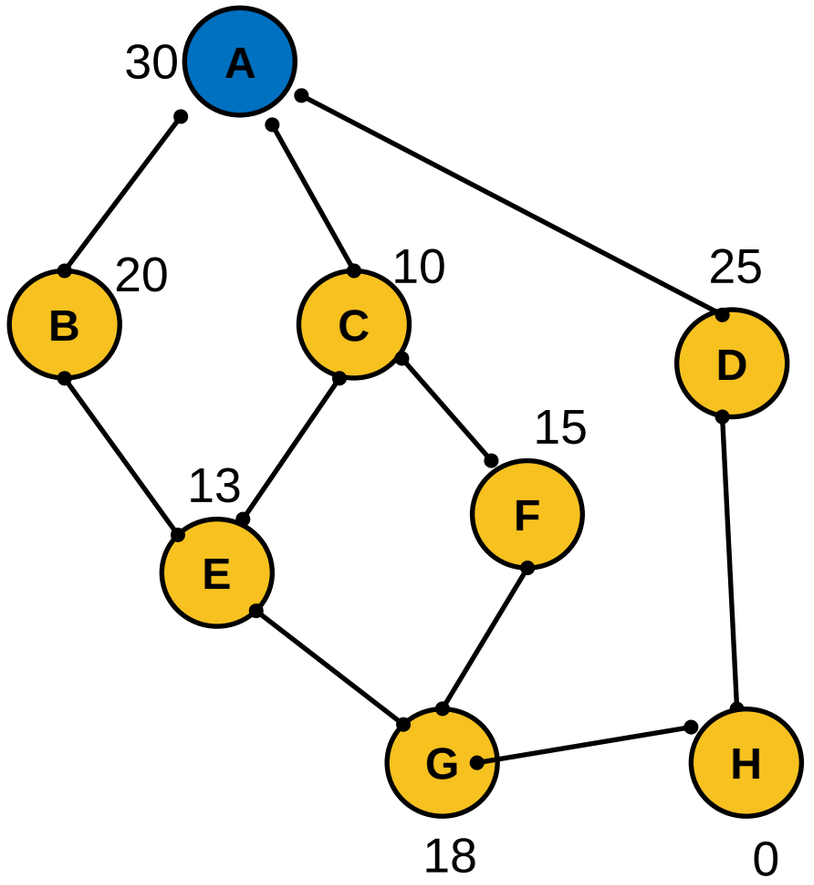
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}



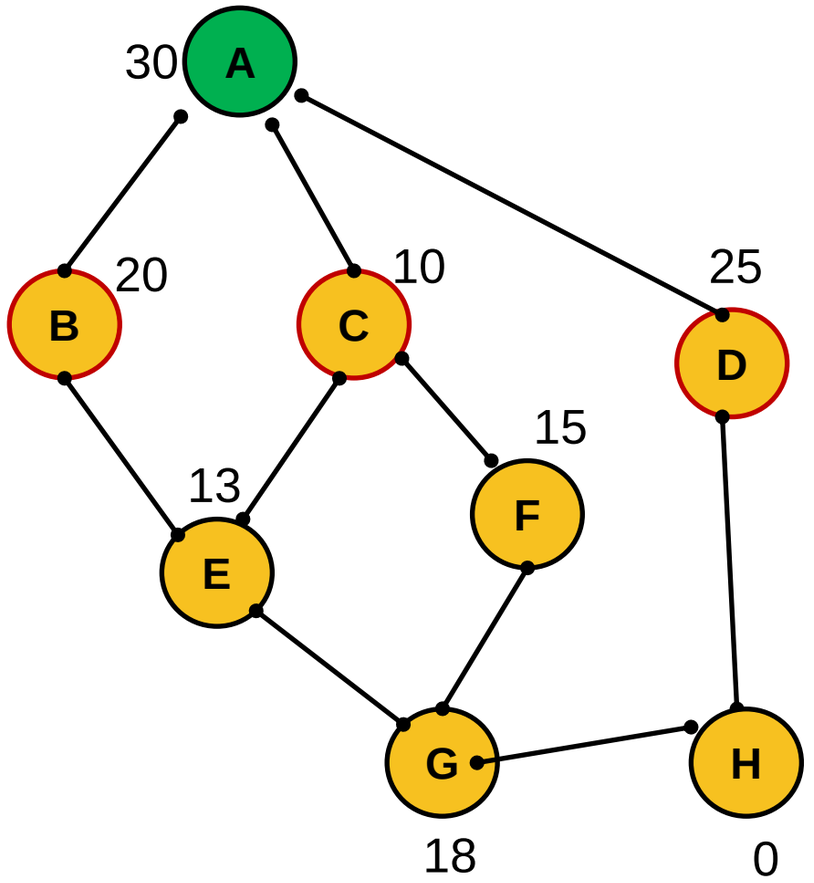
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			~~A~~[30]
I	A		



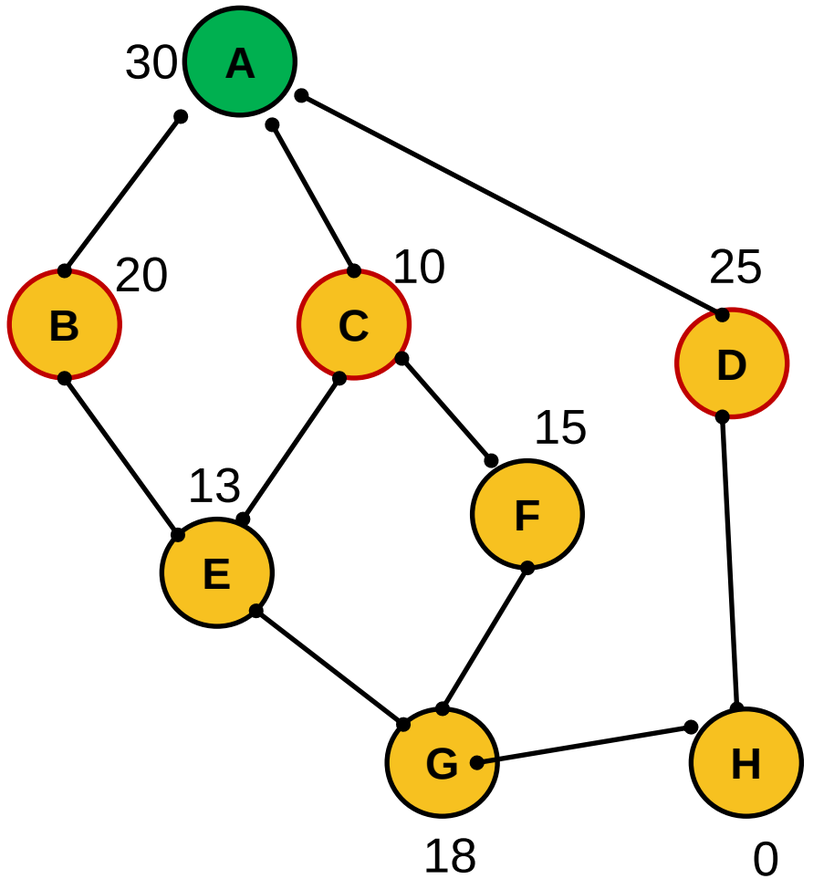
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
I	A	B^{20}, C^{10}, D^{25}	



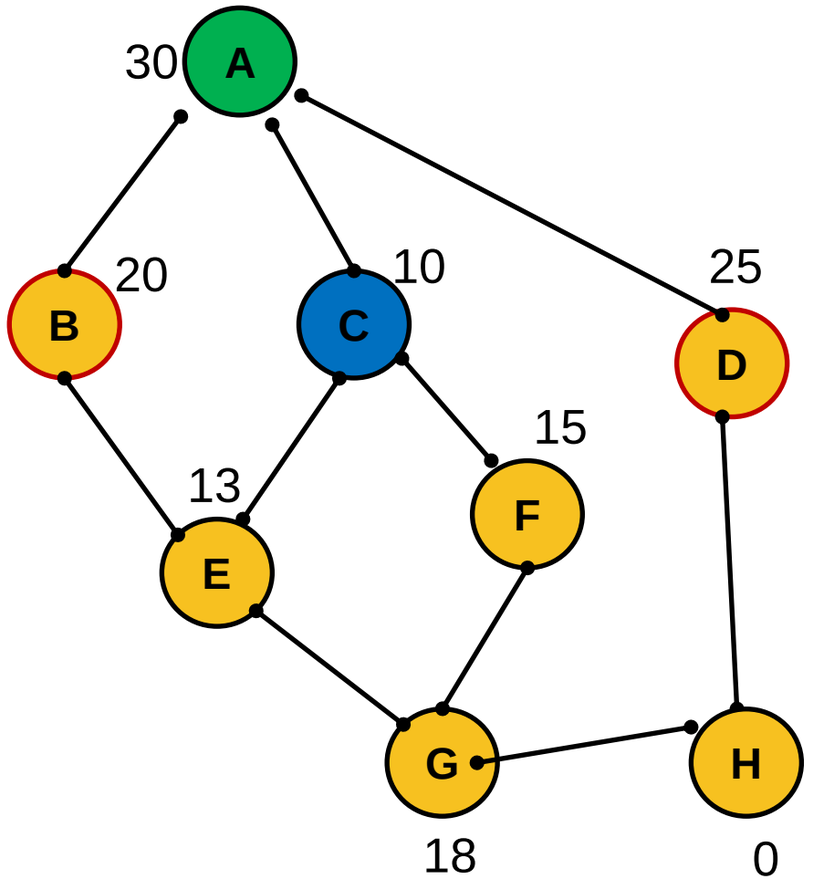
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
I	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}



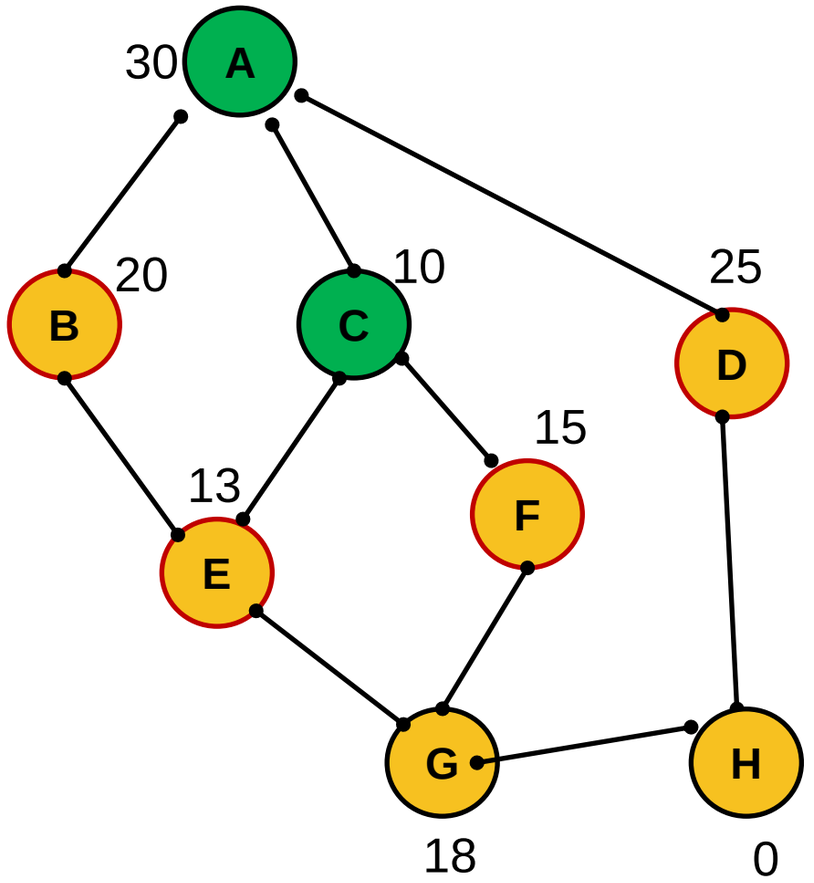
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C		B^{20}, D^{25}



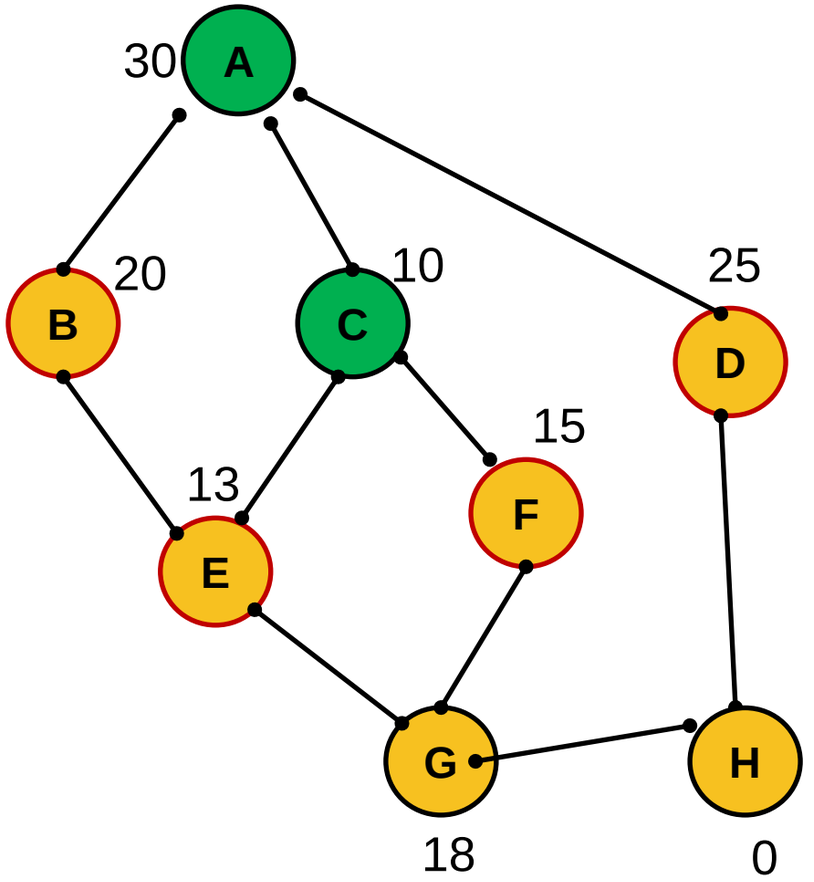
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$B^{20}, D^{25},$



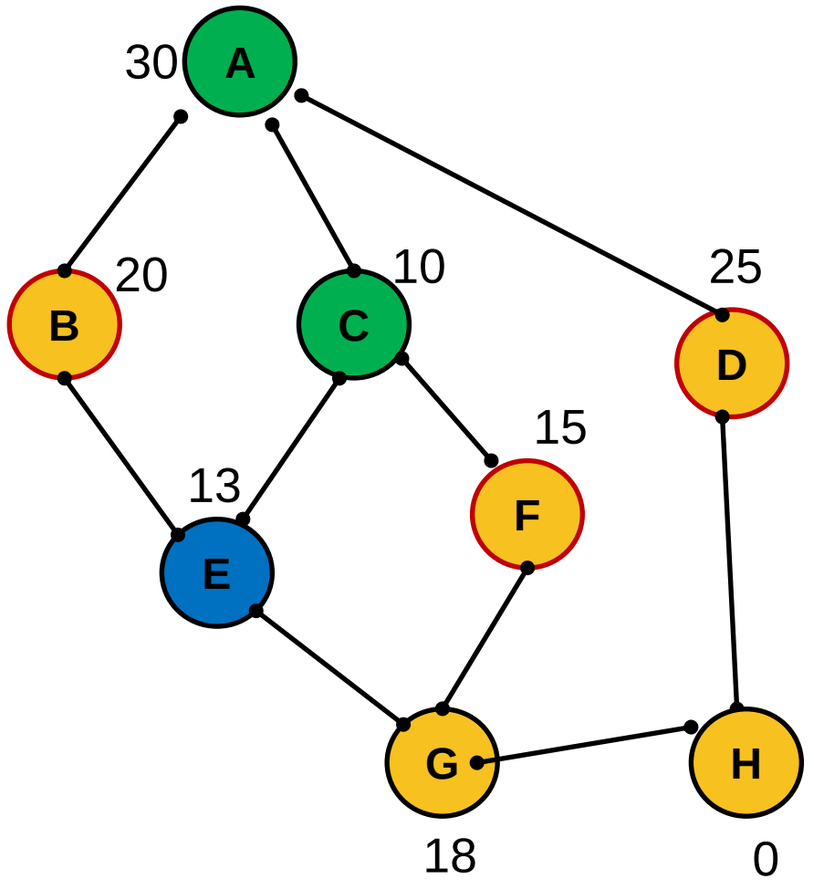
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25,}$



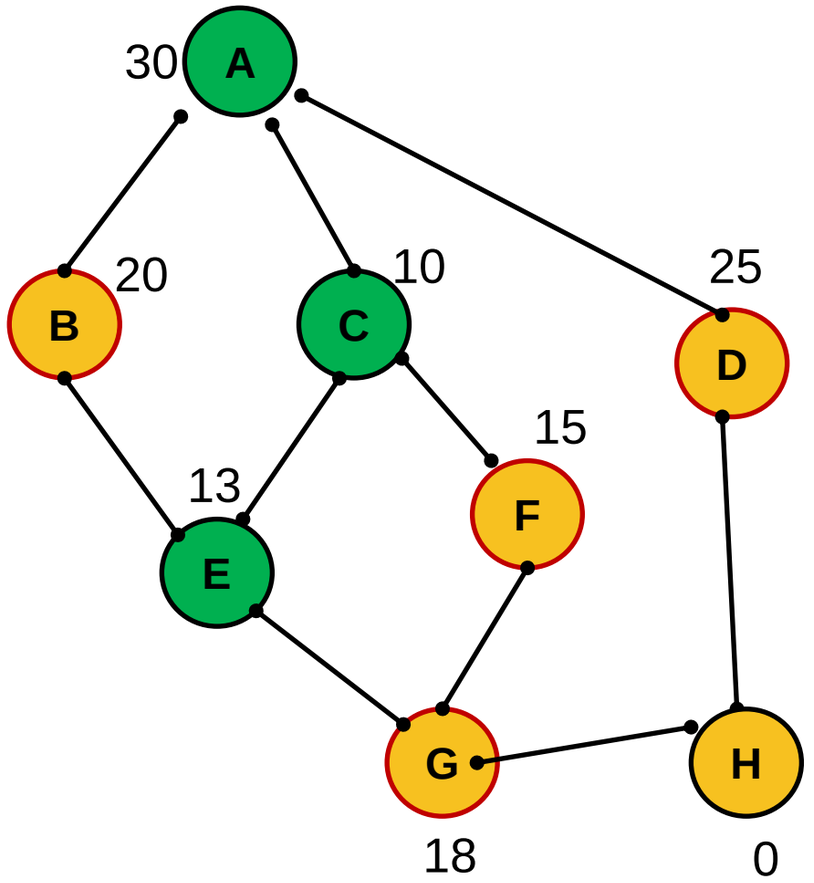
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$\sout{E^{13}}, F^{15}, B^{20}, D^{25}$,
3	E	G^{18}, B^{20}	F^{15}, B^{20}, D^{25}



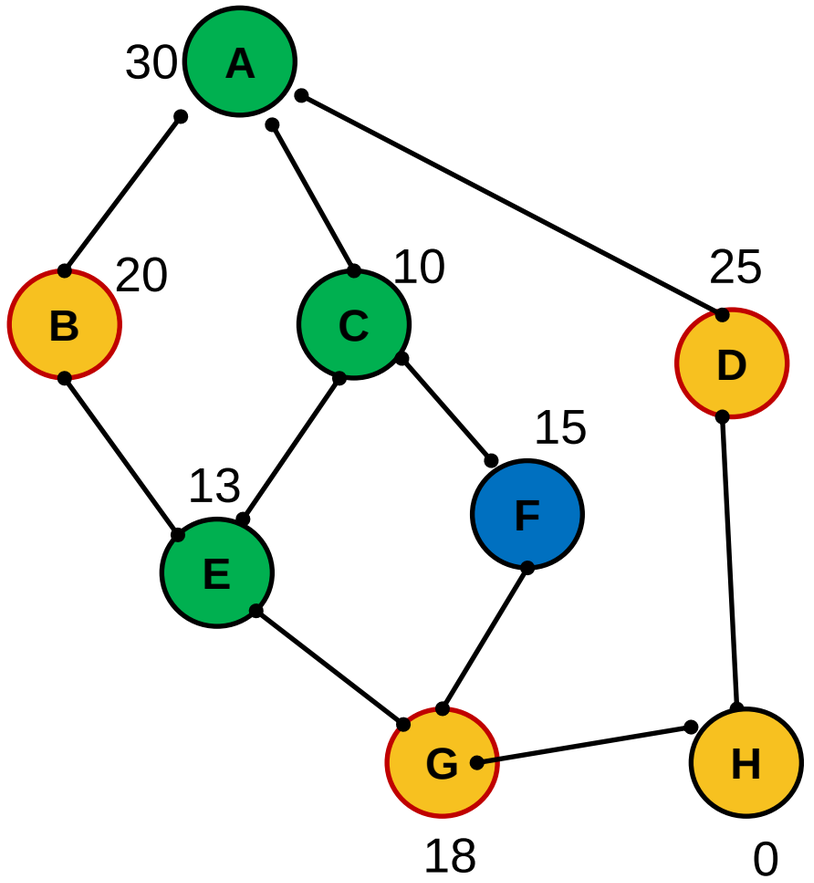
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25,}$
3	E	$G^{18,} B^{20}$	$F^{15}, G^{18}, B^{20}, D^{25}$



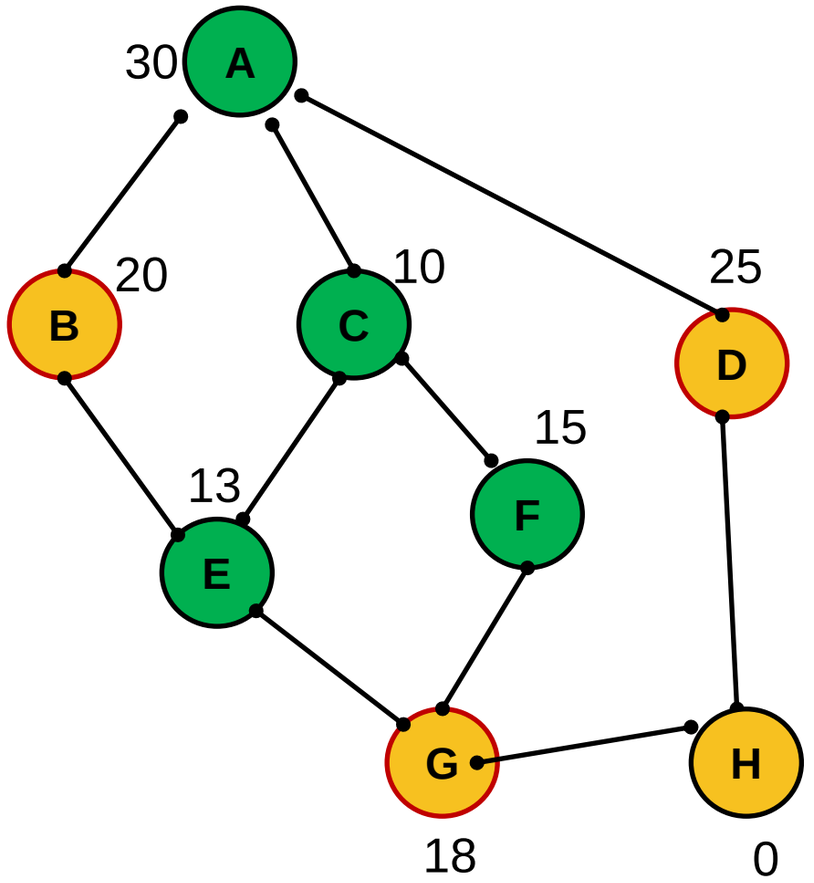
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25,}$
3	E	$G^{18,} B^{20}$	~~F^{15}~~, G^{18}, B^{20}, D^{25}
4	F		G^{18}, B^{20}, D^{25}



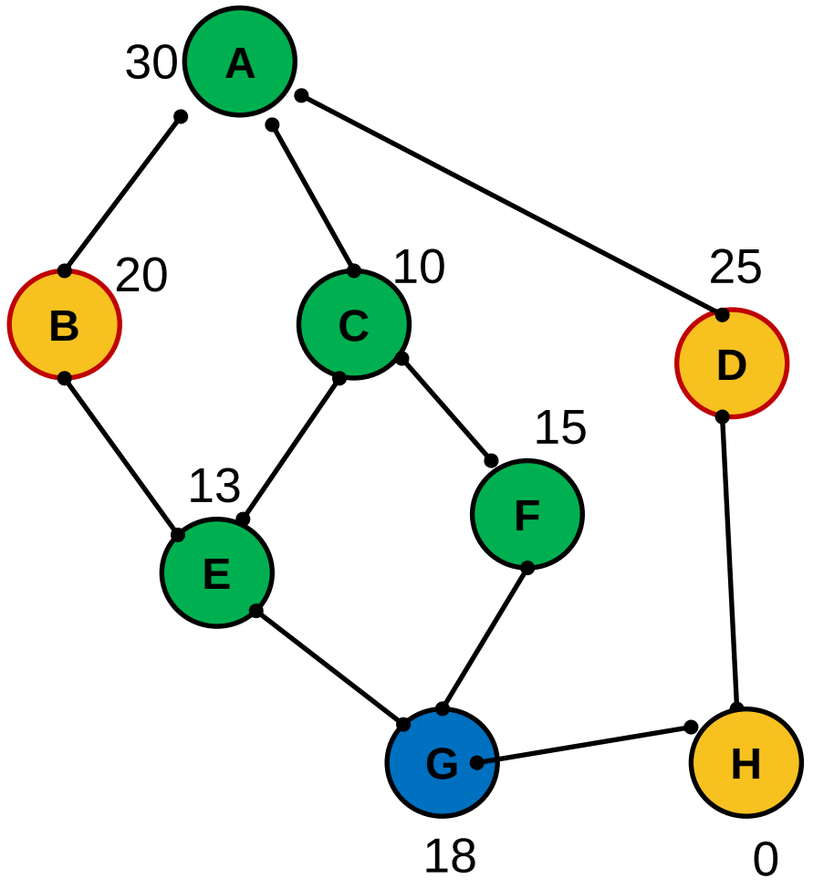
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25,}$
3	E	$G^{18,} B^{20}$	~~F^{15}~~, G^{18}, B^{20}, D^{25}
4	F		G^{18}, B^{20}, D^{25}



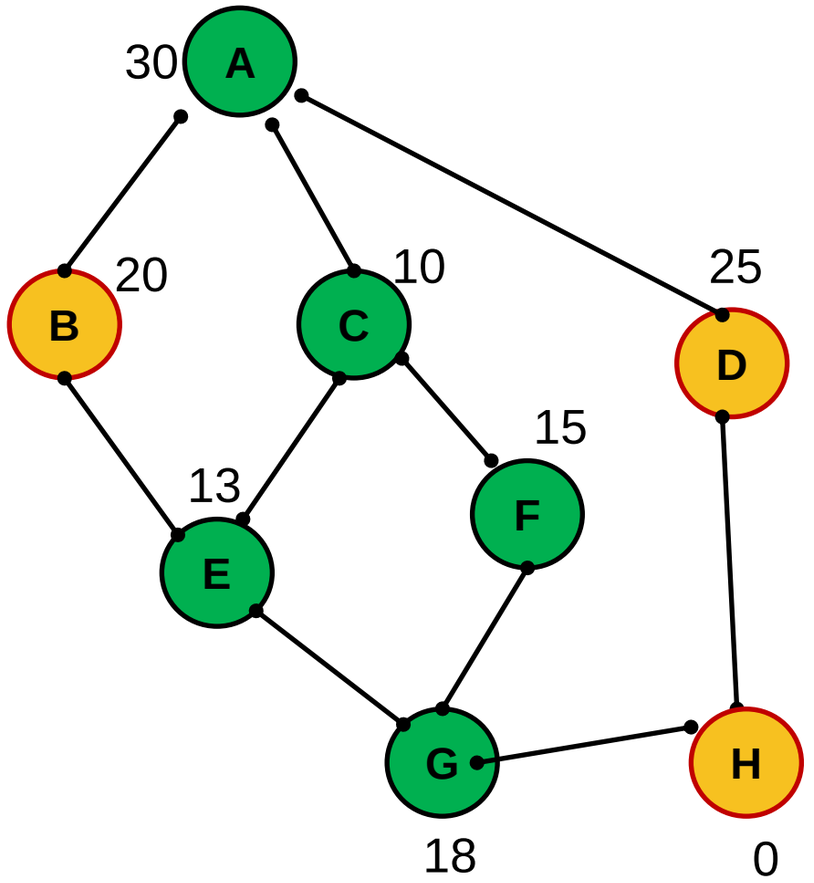
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18},	~~G^{18}~~,B^{20},D^{25}
5	G		B^{20},D^{25}



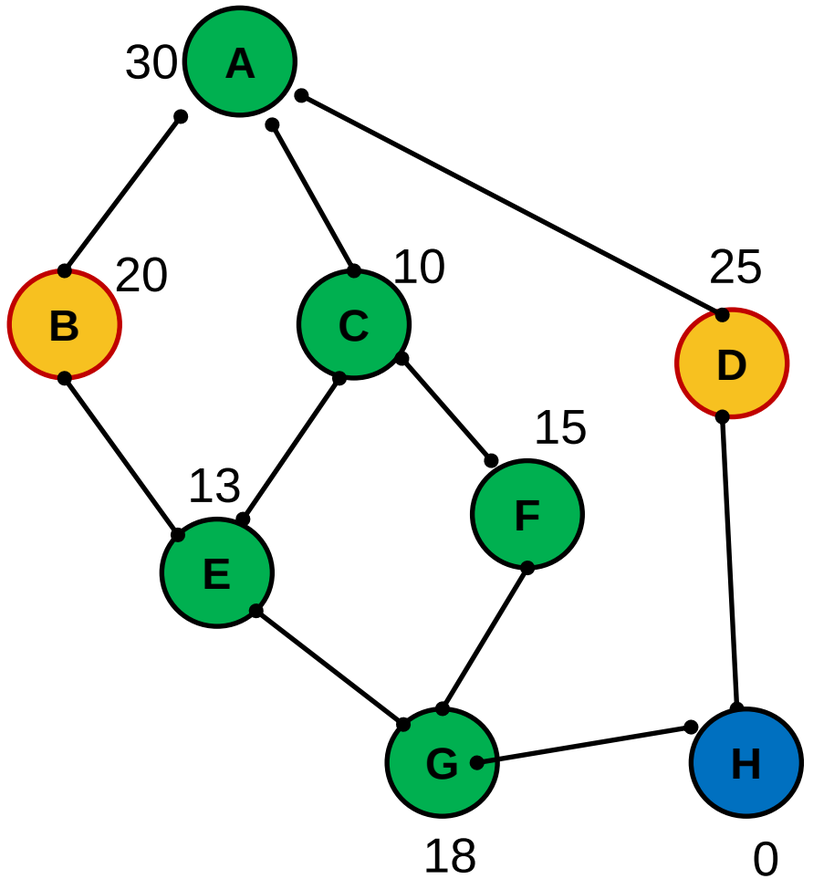
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20},C^{10},D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18}	~~G^{18}~~,B^{20},D^{25}
5	G	H^{0}	H^{0} ,B^{20},D^{25}



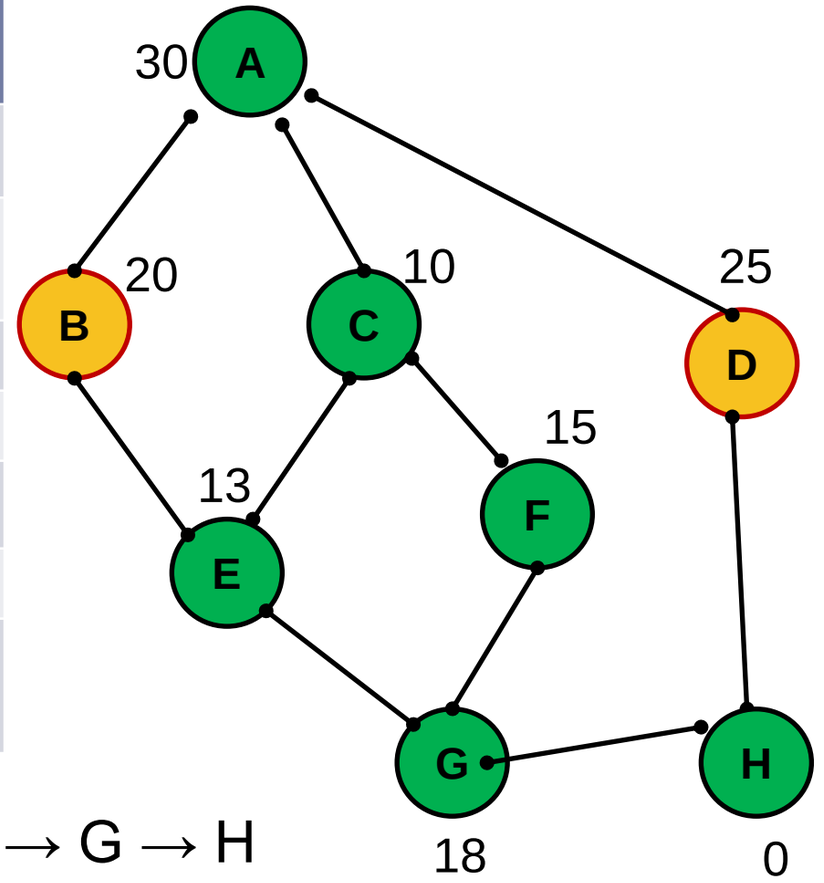
Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
I	A	B^{20},C^{10}, D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18}	~~G^{18}~~,B^{20},D^{25}
5	G	H^{0}	~~H^{0}~~ ,B^{20},D^{25}
6	H ≡ đích		B^{20},D^{25}



Tìm kiếm tốt nhất đầu tiên

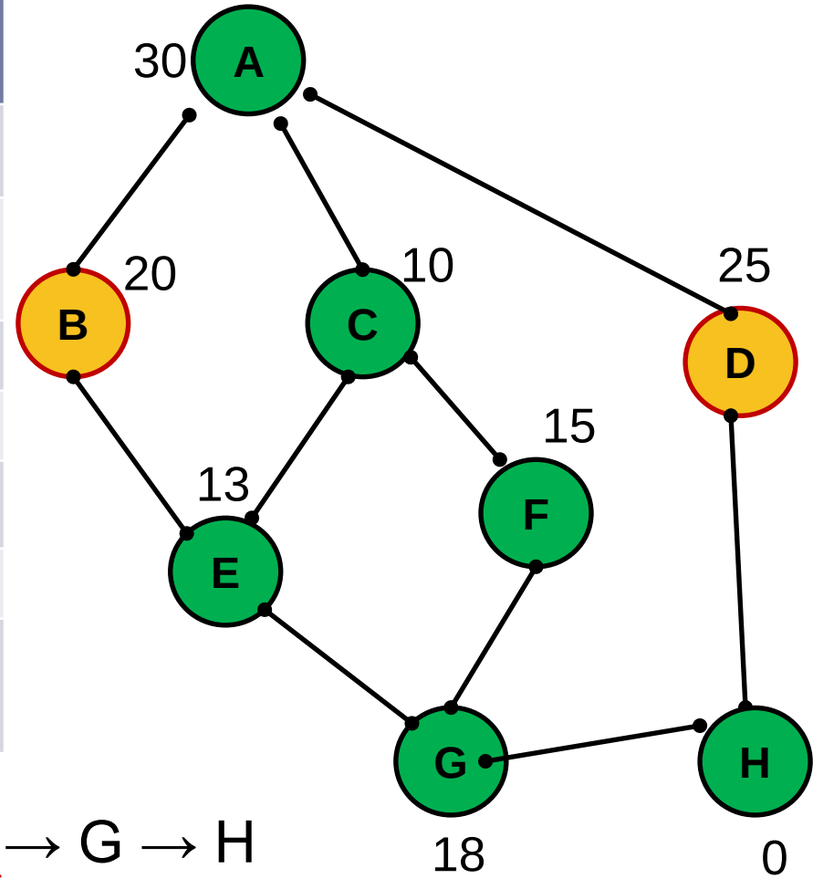
Bước lặp	u	kề(u)	OPEN
0			A^{30}
I	A	B^{20},C^{10}, D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18}	~~G^{18}~~,B^{20},D^{25}
5	G	H^{0}	~~H^{0}~~ ,B^{20},D^{25}
6	H ≡ đích		B^{20},D^{25}



Quá trình duyệt : A → C → E →F → G → H

Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
I	A	B^{20},C^{10}, D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18}	~~G^{18}~~,B^{20},D^{25}
5	G	H^{0}	~~H^{0}~~ ,B^{20},D^{25}
6	H ≡ đích		B^{20},D^{25}

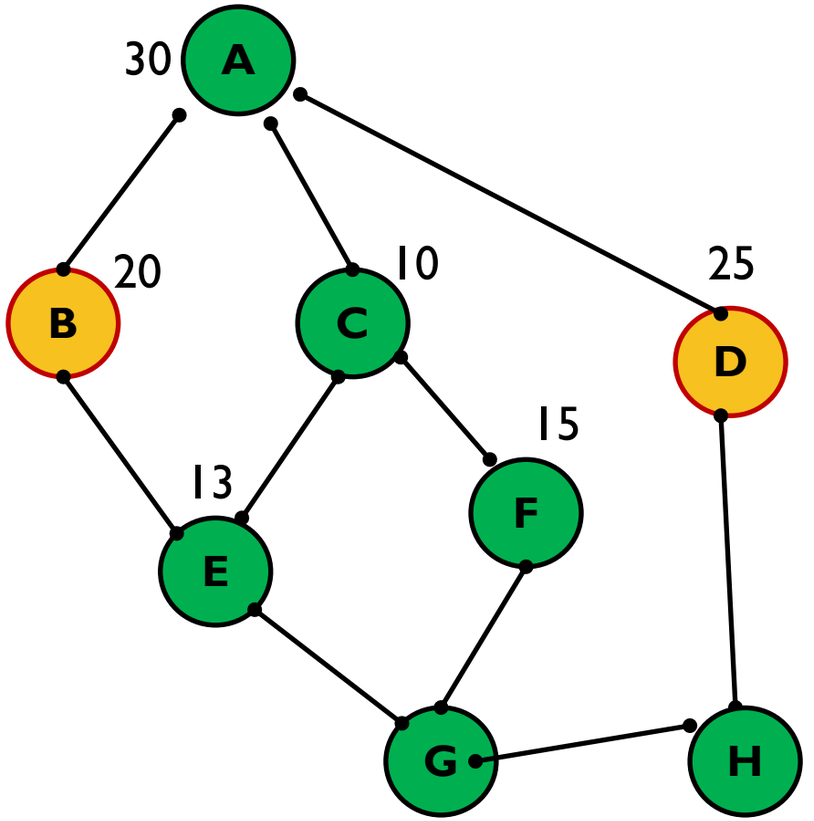


Quá trình duyệt : A → C → E → ~~F~~ → G → H

parent

Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20}, C^{10}, D^{25}	C^{10}, B^{20}, D^{25}
2	C	E^{13}, F^{15}	$E^{13}, F^{15}, B^{20}, D^{25}$
3	E	G^{18}, B^{20}	$F^{15}, G^{18}, B^{20}, D^{25}$
4	F	G^{18}	~~G^{18}~~$, B^{20}, D^{25}$
5	G	H^{0}	~~H^{0}~~ $, B^{20}, D^{25}$
7	H ≡ đích	rỗng	B^{20}, D^{25}



Quá trình duyệt : A → C → E → F → G → H

Đường đi : A → C → E → G → H hoặc A → C → F → G → H

Phương pháp: lần ngược từ đích về nút xuất phát theo các nút cha

Tìm kiếm tốt nhất đầu tiên

Bước lặp	u	kề(u)	OPEN
0			A^{30}
1	A	B^{20},C^{10}, D^{25}	C^{10}, B^{20},D^{25}
2	C	E^{13},F^{15}	E^{13},F^{15},B^{20},D^{25},
3	E	G^{18}, B^{20}	F^{15}, G^{18},B^{20},D^{25}
4	F	G^{18}	~~G^{18}~~,B^{20},D^{25}
5	G	H^{0}	~~H^{0}~~ ,B^{20},D^{25}
7	H ≡ đích	rỗng	B^{20},D^{25}

Loại bỏ nút dư thừa để tìm ra vết đường đi: từ bảng tìm kiếm, lần ngược từ đích về nút xuất phát theo quan hệ cha (cột u) - con (cột kề(u)):

- Cha của H là G (bước 5)
- Cha của G là F (bước 4) và/hoặc E (bước 3)
- Cha của F và E đều là C (bước 2)
- Cha của C là A (bước 1)

Phương pháp: lần ngược từ đích về nút xuất phát theo các nút cha

Tìm kiếm tốt nhất đầu tiên

```
Procedure  Best_First_Search
begin
Khởi tạo danh sách OPEN = {trạng thái ban đầu};
while true do
   if (OPEN rỗng) then
      {thông báo tìm kiếm thất bại; stop};
   Loại trạng thái u ở đầu danh sách OPEN;
   if u là trạng thái kết thúc then
      {thông báo tìm kiếm thành công; stop};
   Chèn các đỉnh kề của u vào OPEN sao cho OPEN được
sắp xếp theo thứ tự tăng dần của hàm đánh giá
end
```

Tìm kiếm beam

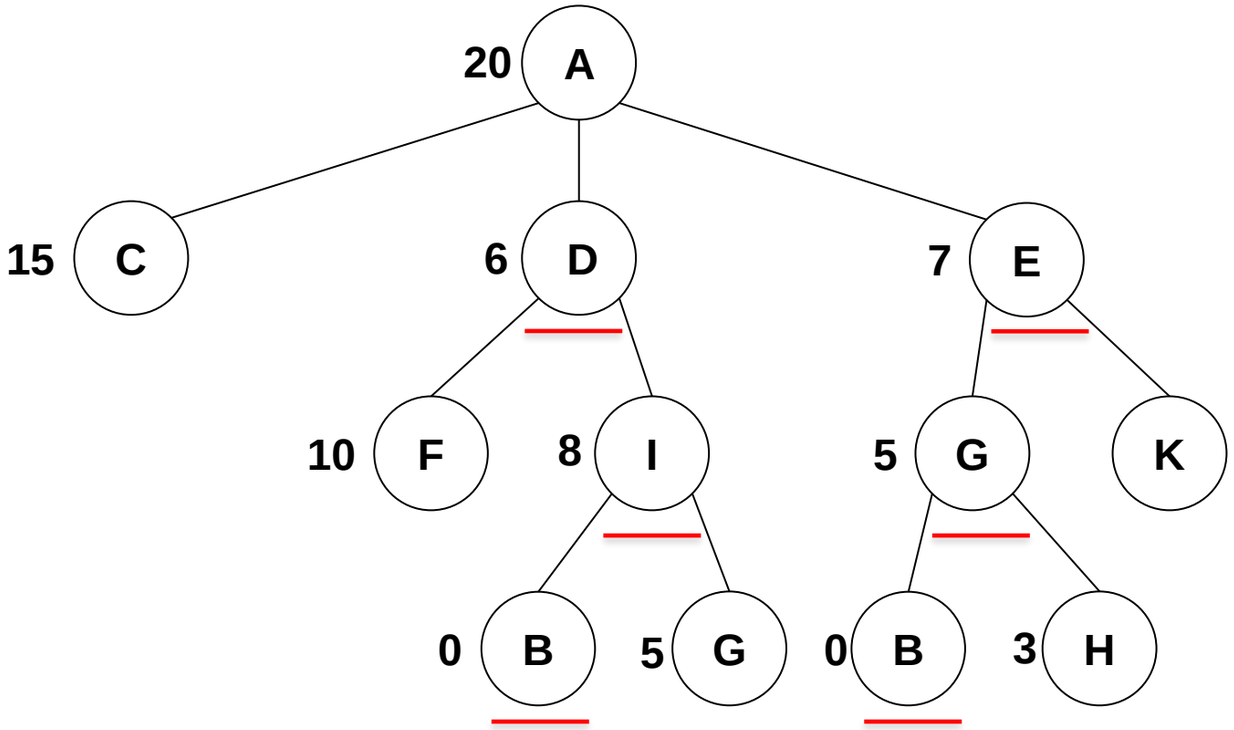
- Giống tìm kiếm tốt nhất đầu tiên

- Hạn chế chỉ phát triển k đỉnh tốt nhất ở mỗi độ sâu

Tốt nhất đầu tiên BFS	Beam
Duyệt u, phát triển tất cả các đỉnh kề v của u (đưa tất cả vào OPEN)	Duyệt u, phát triển 1 số đỉnh kề v của u sao cho tổng số đỉnh ở cùng độ sâu trong OPEN là k (đưa những đỉnh n vào OPEN)

Tìm kiếm beam

Ví dụ: tìm đường đi từ A → B, k = 2



Tìm kiếm leo đồi

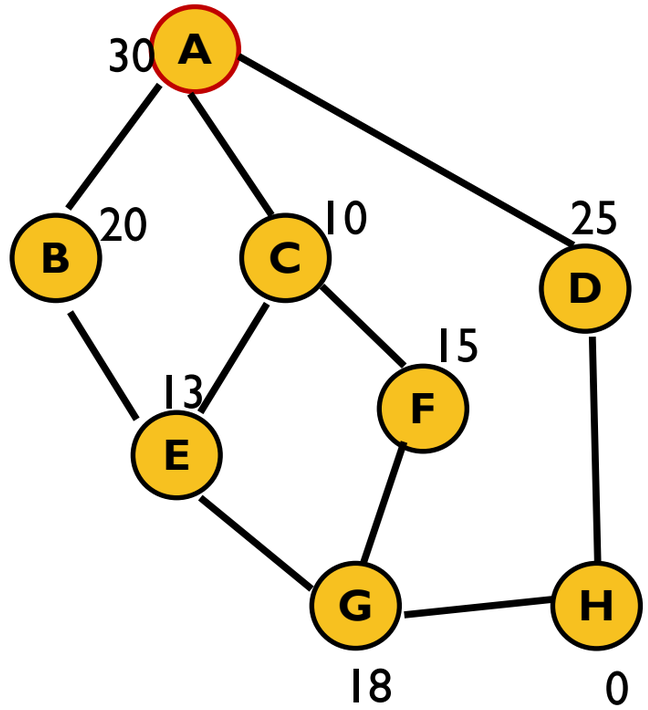
- Tìm kiếm theo độ sâu dưới định hướng của hàm đánh giá h(u)
- Tại mỗi bước của tìm kiếm
  - Chọn đỉnh u ở đầu danh sách OPEN để duyệt
  - Sau khi duyệt đỉnh u :
    - Sắp xếp danh sách các đỉnh kề của u theo thứ tự tăng dần của hàm đánh giá
    - Chèn danh sách kề này vào đầu OPEN

Tìm kiếm leo đồi

```
Procedure Hill_Climbing_Search
begin
   Khởi tạo Open = {trạng thái ban đầu};
   while true do
      If (Open rỗng) then
         {thông báo thất bại; stop};
      Loại trạng thái u ở đầu danh sách Open;
      If (u là trạng thái kết thúc) then
         {thông báo thành công; stop};
      for (mỗi v kề u) do thêm v vào danh sách L;
      Sắp xếp danh sách L theo thứ tự tăng dần của hàm
      đánh giá;
      Chèn L vào đầu OPEN;
end
```

Tìm kiếm leo đồi

Ví dụ: tìm đường đi ngắn nhất từ A→H bằng tìm kiếm leo đồi?



Tìm kiếm tối ưu

Tìm kiếm tối ưu

Trong thực tế, ta thường không chỉ quan tâm đến việc tìm ra nghiệm mà còn phải quan tâm đến việc nghiệm đó có tối ưu hay không.

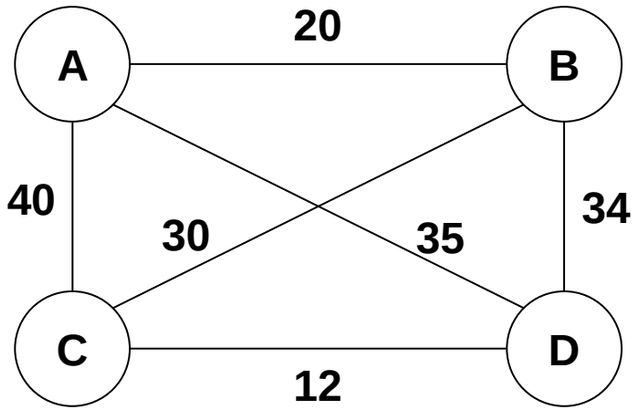
- Tìm đường đi ngắn nhất : tính tới chi phí một hành trình
- Trò chơi 8 số yêu cầu thời gian ngắn nhất để đưa về trạng thái đích: tính cả số bước đã đi

Trong các chiến lược tìm kiếm mù và tìm kiếm kinh nghiệm, chúng ta chưa quan tâm đến độ dài hay chi phí của đường đi nghiệm.

Tìm kiếm tối ưu

Bài toán Người du lịch:

Tìm đường đi ngắn nhất cho du lịch xuất phát từ một thành phố, đi qua lần lượt tất cả các thành phố duy nhất một lần và quay về thành phố ban đầu với chi phí rẻ nhất



Tìm kiếm tối ưu

Tìm kiếm kinh nghiệm	Tìm kiếm tối ưu
Hàm đánh giá : f(u) = h(u)	Hàm đánh giá đầy đủ : f(u) = h(u) + g(u)
Hàm đánh giá định hướng việc tìm kiếm (chọn đỉnh, rẽ nhánh)	Tối ưu hóa hàm đánh giá Hàm đánh giá → min(max)

Tìm kiếm tối ưu

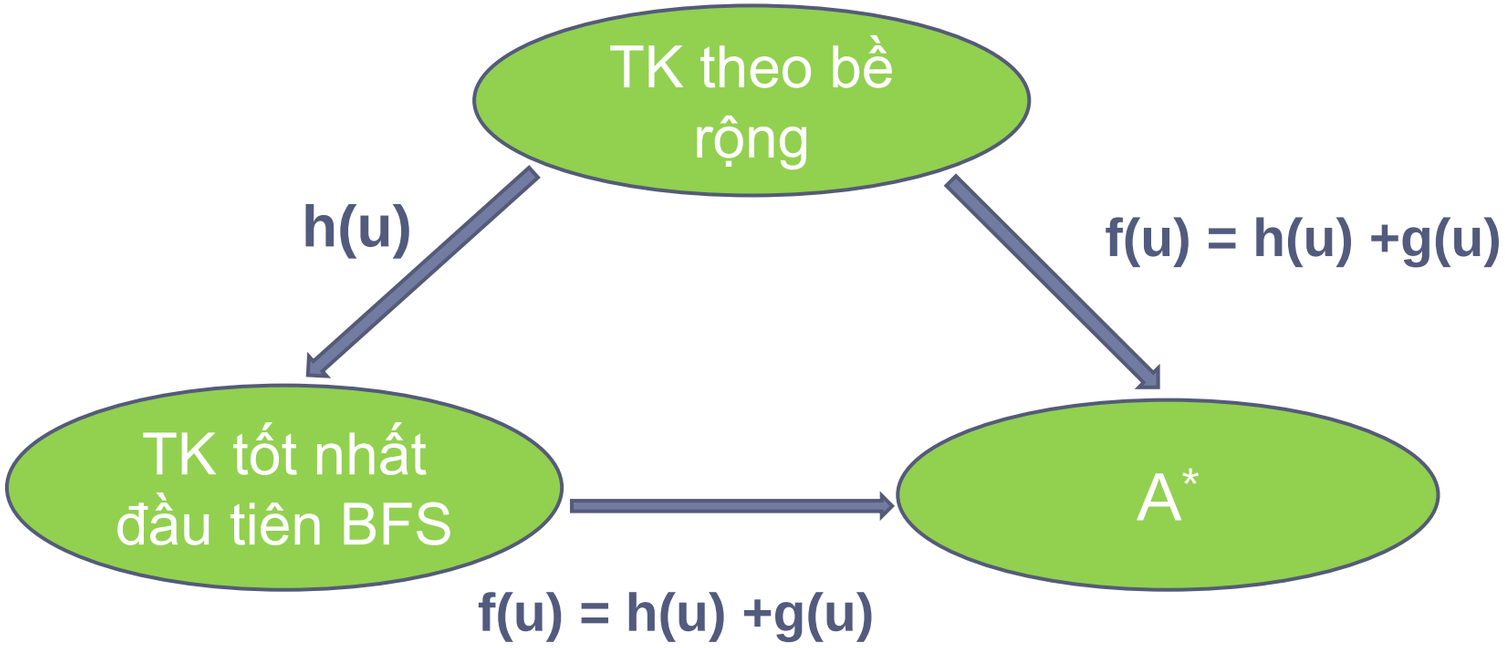
- Các kĩ thuật tìm đường đi ngắn nhất : A^*, nhánh cận

- Các kĩ thuật tìm kiếm cục bộ: TK leo đồi, gradient, mô phỏng luyện kim

- Tìm kiếm bắt chước sự tiến hóa : thuật toán di truyền

Thuật toán A*

Là thuật toán tìm kiếm tốt nhất đầu tiên với hàm đánh giá h(u)+g(u) thay vì h(u)

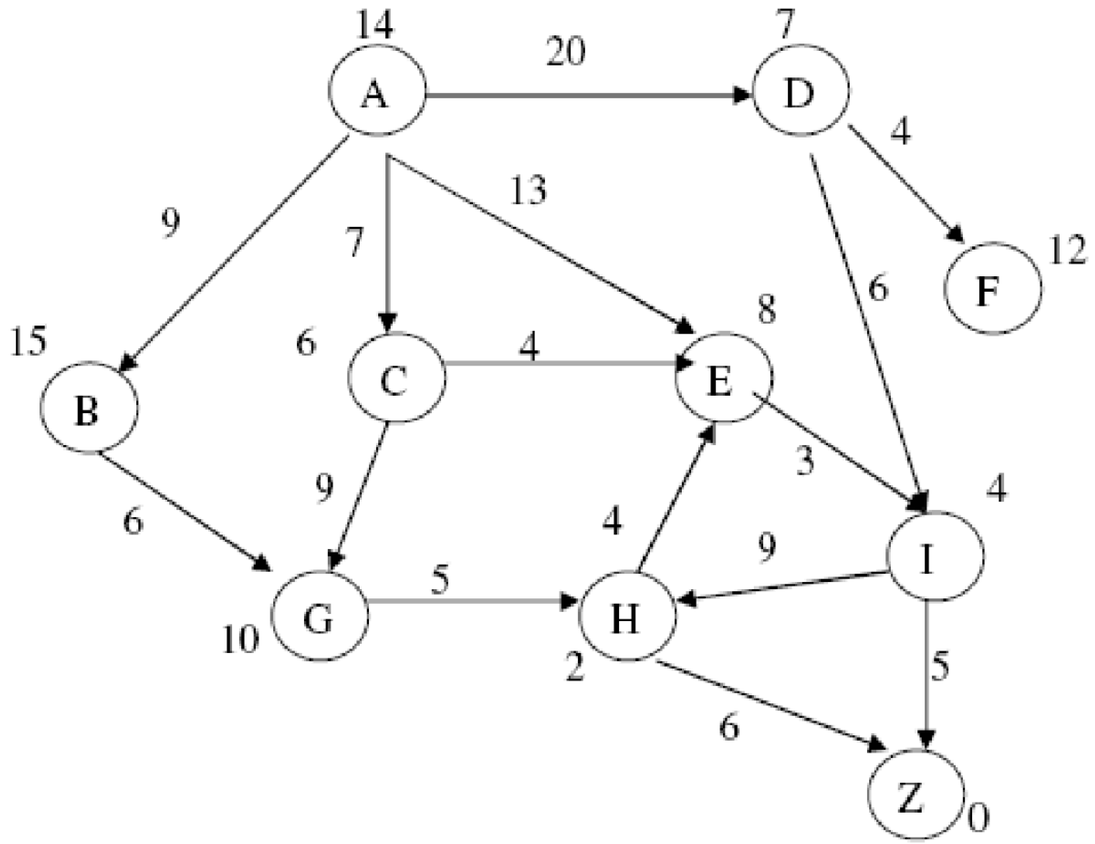


Thuật toán A*

```
Procedure  A*
begin
   Khởi tạo danh sách  OPEN = {trạng thái ban đầu};
   while true do
      if (Open rỗng) then
         {thông báo tìm kiếm thất bại; stop};
   Loại trạng thái u ở đầu danh sách OPEN;
      if u là trạng thái kết thúc then
         {thông báo tìm kiếm thành công; stop};
   for mỗi v kề u
         g(v) = g(u) + k(u,v); // k(u,v) là chi phí thực tế để đi từ
u tới v, được cho trước
         f(v) = g(v)  + h(v); // ưu tiên g nhỏ, cho phép duyệt lại v
nếu f mới tốt hơn
         Chèn v vào OPEN sao cho OPEN được sắp xếp theo thứ tự tăng
dần của f;
end
```

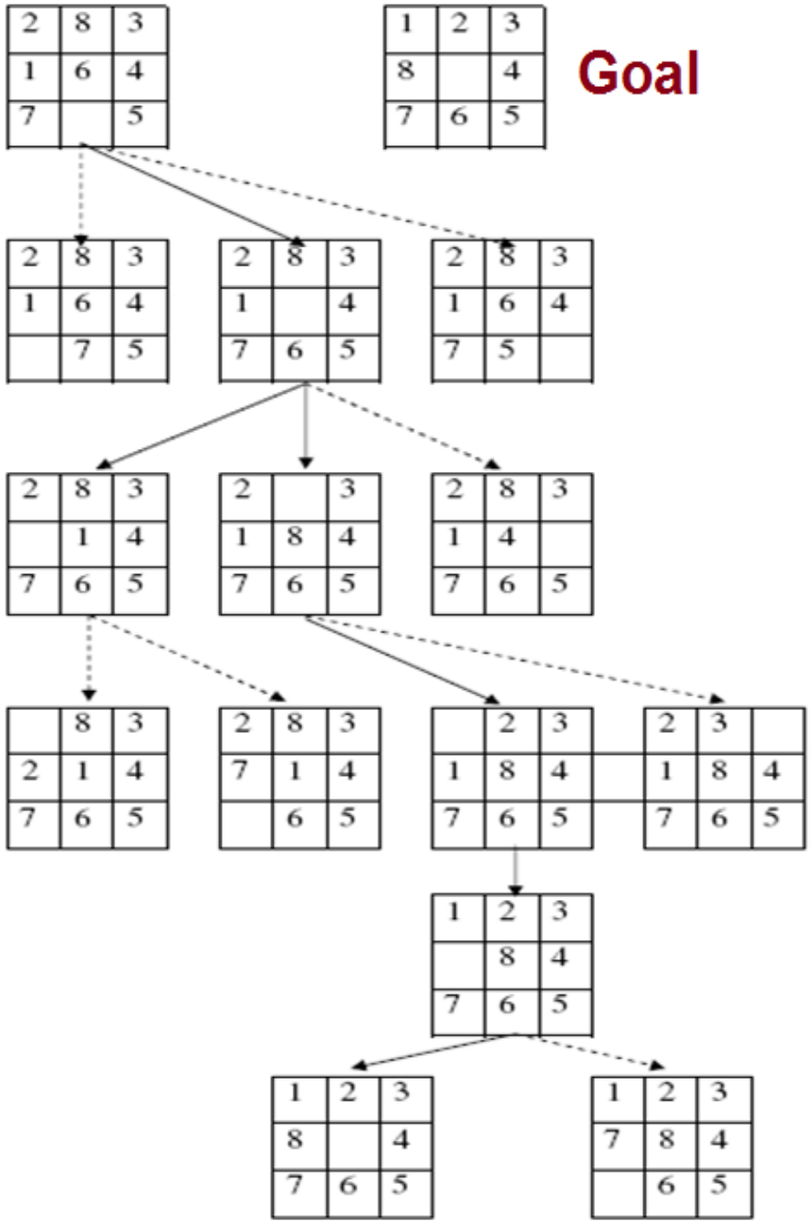
Thuật toán A*

- Tìm đường đi ngắn nhất từ A tới Z bằng A*:
  - Giá trị gắn tại mỗi đỉnh là h(u)
  - Giá trị gắn tại mỗi cạnh là chi phí để chuyển trạng thái k(u,v)
  - Lưu ý: khi g+h như nhau thì ưu tiên g bé hơn



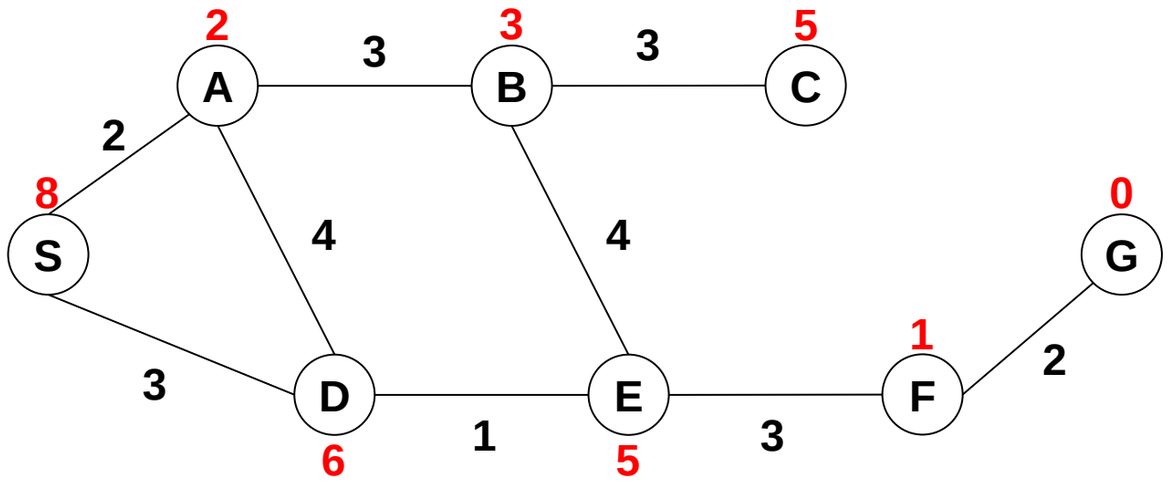
Thuật toán A*

- Bài tập 1:
- Tìm cách chơi nhanh nhất với hàm heuristic h1,h2



Thuật toán A*

Bài tập 2: Cho sơ đồ nối các thành phố. Tìm đường đi từ S tới G bằng A*



Thuật toán A*

- Trường hợp đặc biệt : h(u) = 0 với mọi u
  - f(u) = g(u) + h(u) = g(u)
  - Khi đó, A* là TK tốt nhất đầu tiên với hàm đánh giá g(u)

- Định lý: Nếu h(u) là chấp nhận được, thì A* là TK tối ưu

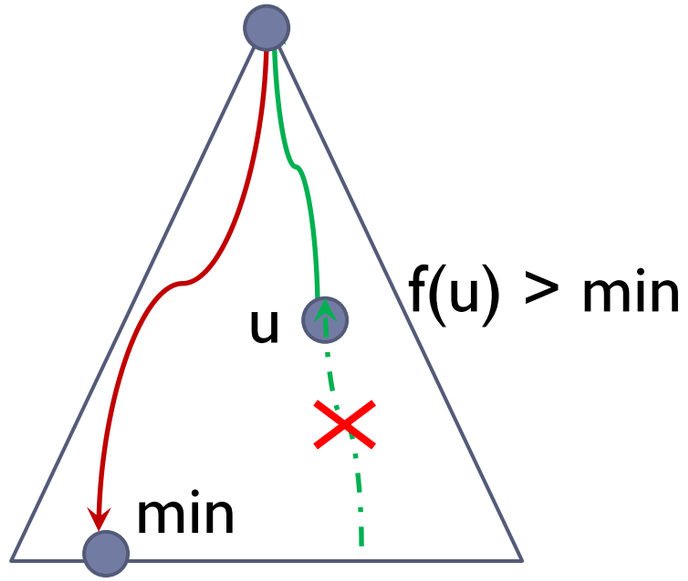
- Thuật toán A* đã được chứng tỏ là thuật toán hiệu quả nhất trong số các thuật toán đầy đủ và tối ưu cho vấn đề tìm kiếm đường đi ngắn nhất.

Thuật toán nhánh cận

- Branch-and-bound search
- Thuật toán nhánh cận là sự cải tiến của thuật toán tìm kiếm leo đồi
  - TK leo đồi: gặp nghiệm thì dừng tìm kiếm. Kết quả là 1 tối ưu cục bộ
  - TK nhánh cận: gặp nghiệm thì vẫn tiếp tục tìm kiếm nghiệm tốt hơn. Kết quả là nghiệm tối ưu
- Mục đích: chuyển từ tìm kiếm nghiệm tối ưu cục bộ sang tìm kiếm tốt nhất toàn cục

Thuật toán nhánh cận

- min: chi phí ngắn nhất tạm thời tìm thấy từ lúc bắt đầu tìm kiếm
- Khi xét nút u, nếu **f(u) > min** thì sẽ cắt bỏ nhánh con của u
  - Toàn bộ các nút con/cháu v của u đều có **f(v) > f(u) > min** nên không thể là nghiệm tốt hơn (tối ưu hơn)
- Nếu tìm thấy 1 đường đi mới tốt hơn đường đi tốt nhất tạm thời (có chi phí min), cập nhật lại min và đường đi tốt nhất tạm thời đó



Thuật toán nhánh cận

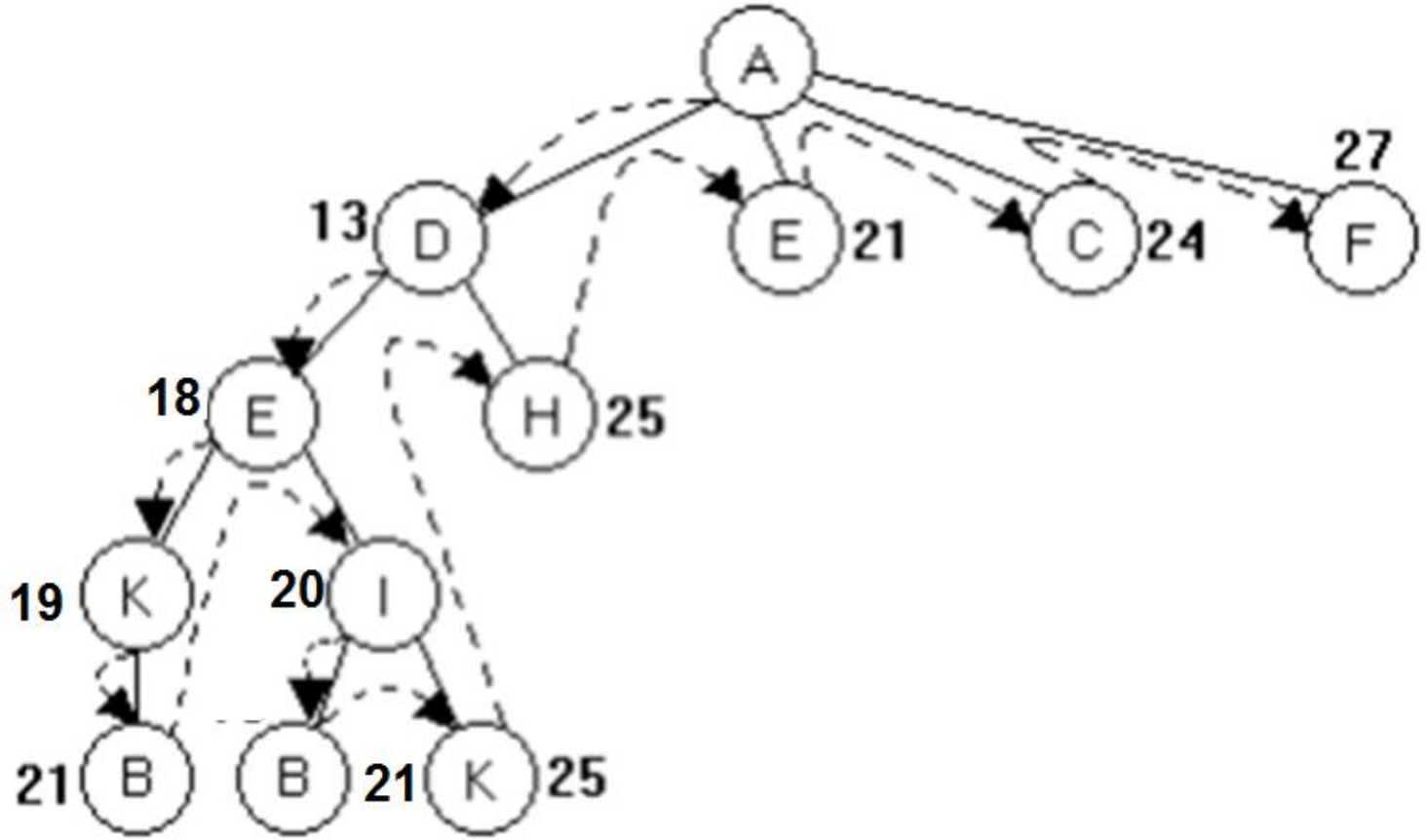
- Branch-and-bound search
- Thuật toán nhánh cận là sự cải tiến của thuật toán tìm kiếm leo đồi
  - TK leo đồi: gặp nghiệm thì dừng tìm kiếm. Kết quả là 1 tối ưu cục bộ
  - TK nhánh cận: gặp nghiệm thì vẫn tiếp tục tìm kiếm nghiệm tối ưu. Kết quả là nghiệm tối
- Mục đích: chuyển từ tìm kiếm nghiệm tối ưu cục bộ sang tìm kiếm tốt nhất toàn cục

Thuật toán nhánh cận

```
begin
    1. Khởi tạo danh sách OPEN chỉ chứa trạng thái ban đầu;
       Gán giá trị ban đầu cho min ← +∞;
    2. loop do
       2.1 if OPEN rỗng then stop;
       2.2 Loại trạng thái u ở đầu danh sách OPEN;
    2.3 if u là trạng thái kết thúc then
            if f(u)  < min then {min ←f(u); Quay lại 2.1}; // cập nhật lại min
    2.4 if f(u) > min then Quay lại 2.1; //cắt bỏ nhánh con
    2.5 for mỗi trạng thái v kề u do
        {
            g(v) ← g(u) + k(u,v);
            f(v) ← g(v) + h(v);
            Đặt v vào danh sách L
        }
    2.6 Sắp xếp L theo thứ tự tăng của hàm f;
    2.7 Chèn L vào đầu danh sách OPEN
end;
```

Thuật toán nhánh cận

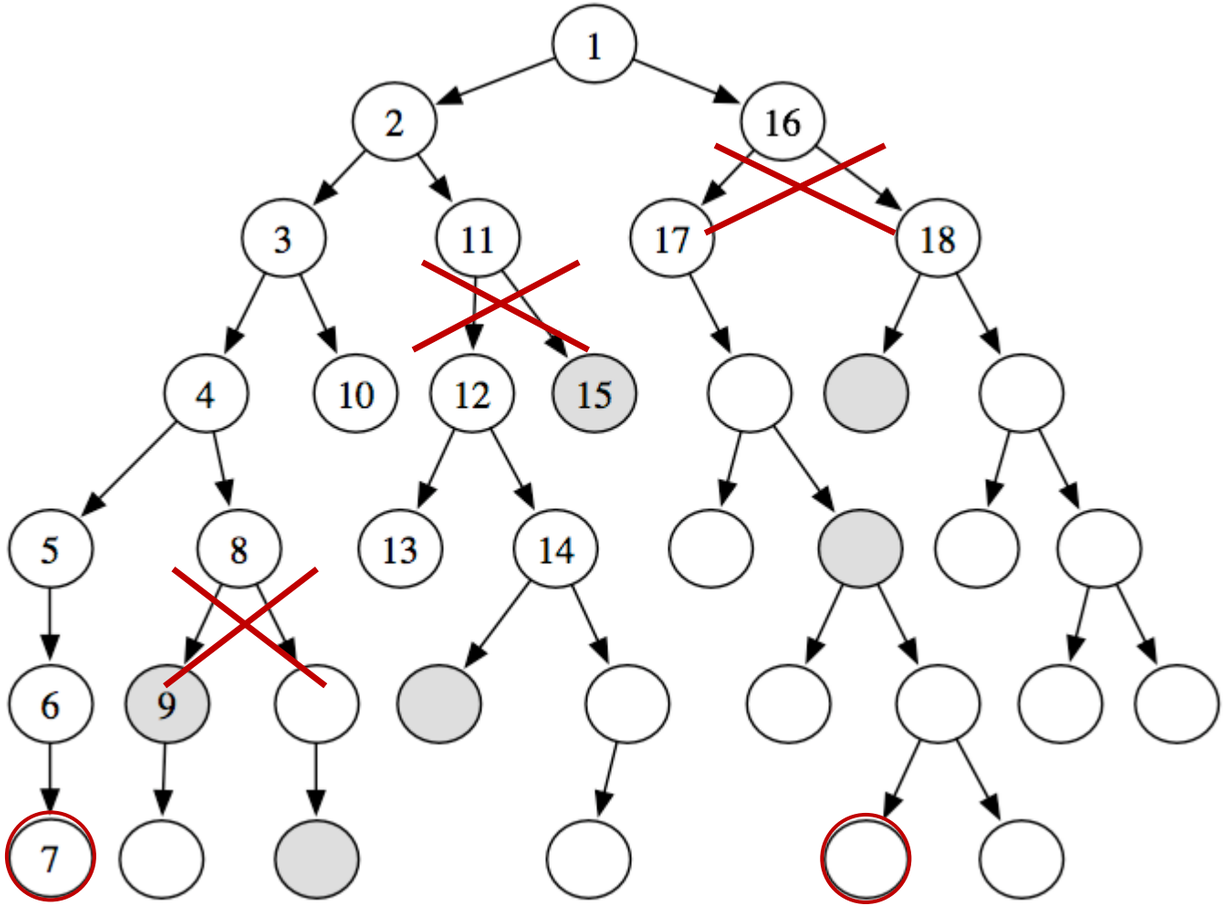
- Khuyết h(u). Thông tin trên mỗi nút là chi phí g(u)



Đinh Mạnh Tường, Trí tuệ nhân tạo, Nhà xuất bản khoa học kỹ thuật, 2002

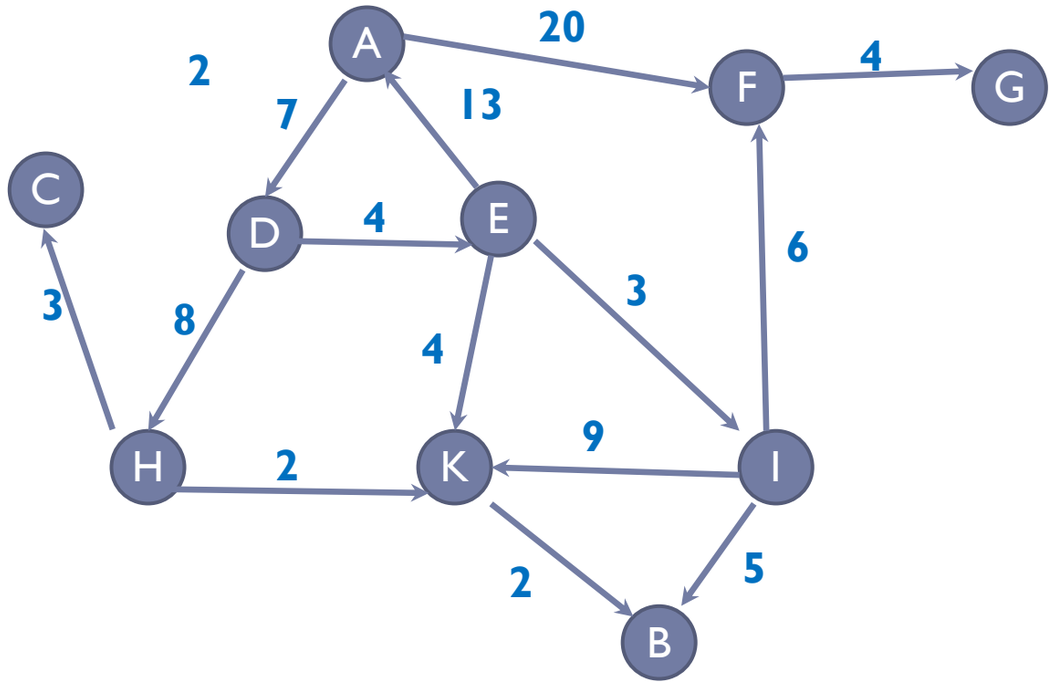
Thuật toán nhánh cận

- Khuyết h(u). Thông tin ghi trên mỗi nút là chi phí tổng chi phí g(u) từ nút start đến u.



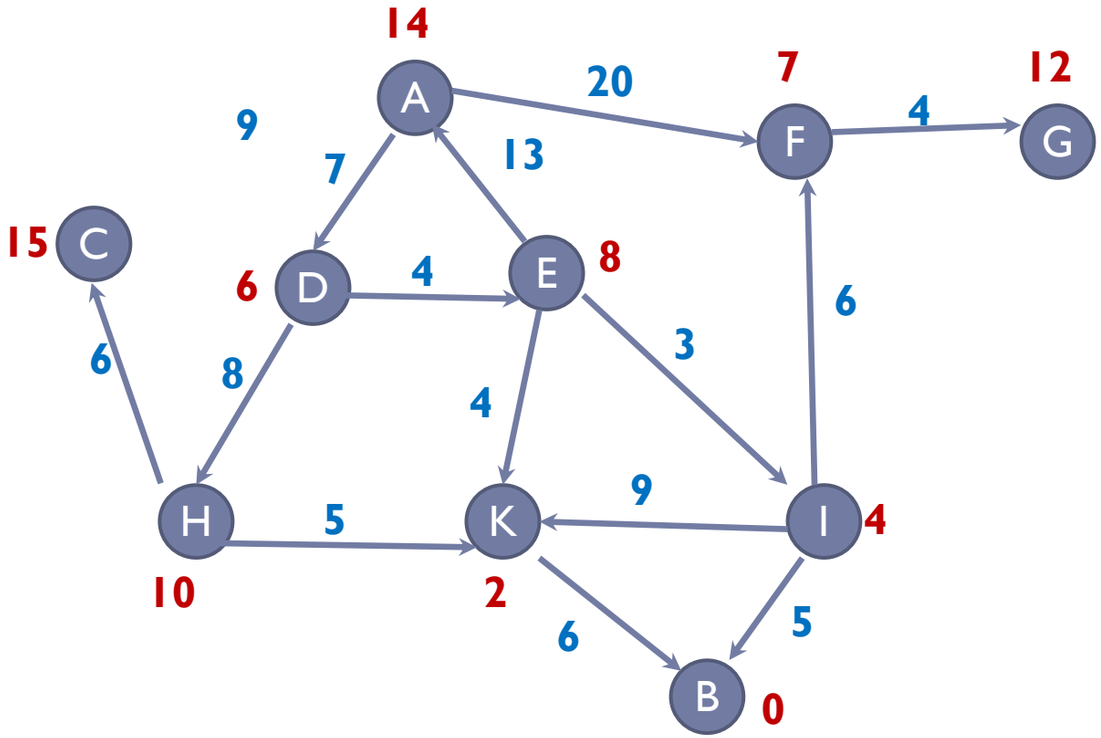
Thuật toán nhánh cận

- Tìm đường đi từ A → B bằng giải thuật nhánh cận
  - Thông tin trên các cạnh là k_{uv}
  - Thuật toán nhánh cận cũng thường áp dụng với bài toán chỉ có thông tin g



Thuật toán nhánh cận

- Tìm đường đi từ A → B bằng giải thuật nhánh cận
  - Có thêm thông tin h(u) gắn tại các đỉnh)



Thuật toán nhánh cận

A*	Nhánh cận
❑Tìm thấy thì dừng ❑Phương án tìm được là tối ưu chỉ khi hàm h là hàm chấp nhận được ❑Khi hàm h không phải là chấp nhận được, phương án tìm được chỉ là phương án tìm thấy đầu tiên, chưa chắc đã là tối ưu nhất	❑Tìm thấy thì vẫn tiếp tục tìm các phương án khác để so sánh ❑Luôn tối ưu ❑Kết hợp đánh chặn (chặt bỏ) trước các nhánh không tốt ngay khi có thể (*phát hiện ra nhánh không tốt so với đường đi tốt nhất tạm thời tìm được*) ❑Cập nhật lại đường đi tốt nhất tạm thời nếu tìm thấy có đường đi khác tốt hơn