

# **TRÍ TUỆ NHÂN TẠO**

**TS. Đào Duy Tuấn**

# **Tham khảo:**

- **Giáo trình Trí tuệ Nhân tạo, TS. Nguyễn Văn Hiệu, khoa CNTT, Trường Đại học Bách Khoa – Đại học Đà Nẵng**
- **Giáo trình Trí tuệ Nhân tạo, TS. Hoàng Lê Uyên Thục, khoa ĐTVT, Trường Đại học Bách Khoa – Đại học Đà Nẵng**

## Nội dung

1. Introduction to Artificial Intelligence, history of AI, course logistics
2. Intelligent agents, uninformed search
3. Heuristic search, A\* algorithm
4. Adversarial search, games
5. Adversarial search, games (continue)
6. **Mid-term**
7. Machine Learning: Basic concepts, linear models, perceptron, K nearest neighbors
8. Machine Learning: Basic concepts, linear models, perceptron, K nearest neighbors (continue)
9. Machine Learning: advanced models, neural networks, SVMs, decision trees and unsupervised learning
10. Machine Learning: advanced models, neural networks, SVMs, decision trees and unsupervised learning (continue)
11. Final-exam

## Chương 1: Giới thiệu

### Nội dung

- Khái niệm về trí tuệ nhân tạo(AI)
- Ứng dụng của trí tuệ nhân tạo
- Nền tảng của trí tuệ nhân tạo
- Lịch sử phát triển trí tuệ nhân tạo

# Khái niệm về trí tuệ nhân tạo (AI)

---

- Trí tuệ: Khả năng học hỏi và giải quyết vấn đề  
*<Từ điển Webster>*
- Trí tuệ nhân tạo là trí tuệ được thể hiện bằng máy móc hoặc phần mềm, trái ngược với trí tuệ tự nhiên  
*<Wikipedia>*
- Ngành Khoa học và kỹ thuật làm cho máy móc thông minh  
*<McCarthy>*
- Nghiên cứu và thiết kế các tác nhân thông minh, trong đó mỗi tác nhân thông minh là một hệ thống nhận thức được môi trường của nó và thực hiện các hành động nhằm tối đa hóa cơ hội thành công.  
*<Sách của Russell & Norvig >*

# Khái niệm về trí tuệ nhân tạo(AI)

---

- Trên thế giới có nhiều khái niệm khác nhau về trí tuệ nhân tạo. Hiện nay vẫn chưa thống nhất về khái niệm trí tuệ nhân tạo.
- Tuy vậy, có hai trường phái khái niệm về trí tuệ nhân tạo
  - **Strong AI:** Tạo ra thiết bị và chương trình máy tính thông minh hơn người
  - **Weak AI:** Tạo ra thiết bị và chương trình máy tính có thể mô phỏng hành vi thông minh của con người

# Khái niệm về trí tuệ nhân tạo

---

- Hiện nay tồn tại 4 quan niệm về trí tuệ nhân tạo
  - Trí tuệ nhân tạo suy nghĩ như người
  - Trí tuệ nhân tạo hành động như người
  - Trí tuệ nhân tạo hành động có lý trí
  - Trí tuệ nhân tạo suy nghĩ có lý trí
- Bài giảng sẽ tập trung theo quan niệm hành động như người
  - Học máy
  - Xử lý ngôn ngữ tự nhiên
  - Xử lý ảnh
  - Robotics

# Trí tuệ nhân tạo suy nghĩ như người

---

- Cách tiếp cận cuối thế kỷ 19, đầu thế kỷ 20 về tâm lý học nhận thức.
- Nghiên cứu xem trí tuệ của con người là gì?
- Nghiên cứu các chức năng thể hiện trí tuệ như: xử lý ngôn ngữ, nghĩ, học và lập luận được thực hiện như thế nào?
- Yêu cầu của trường phái này: chương trình máy tính không chỉ giải đúng bài toán, mà còn đúng từng bước giải của con người.
- Hai cách tiếp cận:
  - Trên xuống: Tâm lý học nhận thức ->Symbolism (Simon & Newell, 1961).
  - Dưới lên: Neural and Brain Science (Mcculloch, Pitt 1950s)  
Artificial Neural Networks.

# Có 4 quan điểm về AI

---

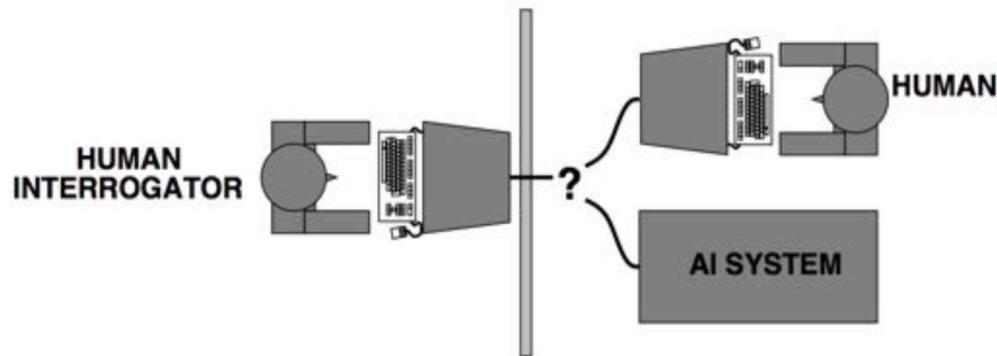
Suy nghĩ như người	Suy nghĩ có lý trí
Hành động như người	Hành động có lý trí

Tài liệu tập trung vào nhóm quan điểm “hành động có lý trí”.

# Trí tuệ nhân tạo hành động như người

---

- Turing (1950) "Computing machinery and intelligence":
- “Máy tính có thể nghĩ?” -> “Máy tính có thể hành động thông minh?”
- Turing Test: Một hệ thống vượt qua bài kiểm tra, nếu nó có thể đánh lừa được người kiểm tra.



- Ưu điểm của Turing Test
  - Khái niệm khách quan về trí tuệ
  - Tránh đi những thảo luận về quá trình bên trong và ý thức
  - Loại trừ định kiến thiên vị của người thẩm vấn

# Trí tuệ nhân tạo hành động như người

---

- Phản đối Turing Test: Trói buộc sự thông minh máy tính theo kiểu con người, trong khi con người có:
  - Bộ nhớ giới hạn
  - Có khuynh hướng nhầm lẫn
- Tuy nhiên Turing Test đã tạo cơ sở cho nhiều đánh giá về chương trình trí tuệ nhân tạo hiện đại



# Trí tuệ nhân tạo suy nghĩ có lý trí

---

- Suy nghĩ đúng với logic
- Bắt đầu từ thời Hylap cổ đại (Rule of Arguments) cho đến G. Boole (Mathematical Model of Thoughts), cho đến Hilbert (Logics) xây dựng các hình thức của logic khác nhau: ký hiệu và quy tắc
- Các vấn đề:
  - Không phải tất cả tri thức đều có thể biểu diễn bằng ký hiệu logic.
  - Mục đích của suy nghĩ là gì?

# Trí tuệ nhân tạo hành động có lý trí

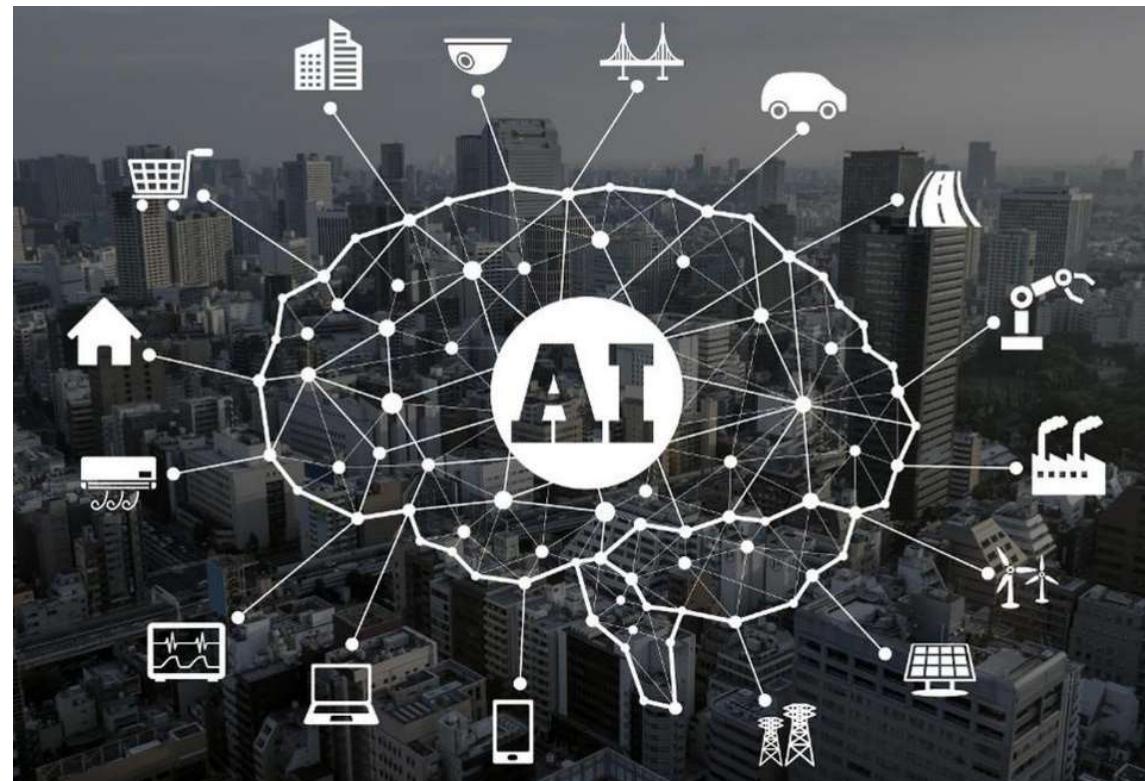
---

- Doing the right thing (not “Doing the thing right”).
- Hành động được coi là thông minh nếu giúp cho tác nhân (agent) thực hiện hành động tăng cơ hội đạt được đích đặt ra trong môi trường mà nó tồn tại
- Như vậy: Lợi điểm của định nghĩa:
  - Thông minh không nhất thiết phải là con người hay giống người!!
  - Hành vi thông minh không nhất thiết phải thực hiện thông qua suy nghĩ, lý luận.

# Định nghĩa về trí tuệ nhân tạo

---

Trí tuệ nhân tạo là khoa học nghiên cứu các hành vi thông minh nhằm giải quyết các vấn đề được đặt ra đối với các chương trình máy tính!!!



# Ứng dụng của trí tuệ nhân tạo

---

ChatGPT



# Ứng dụng của trí tuệ nhân tạo

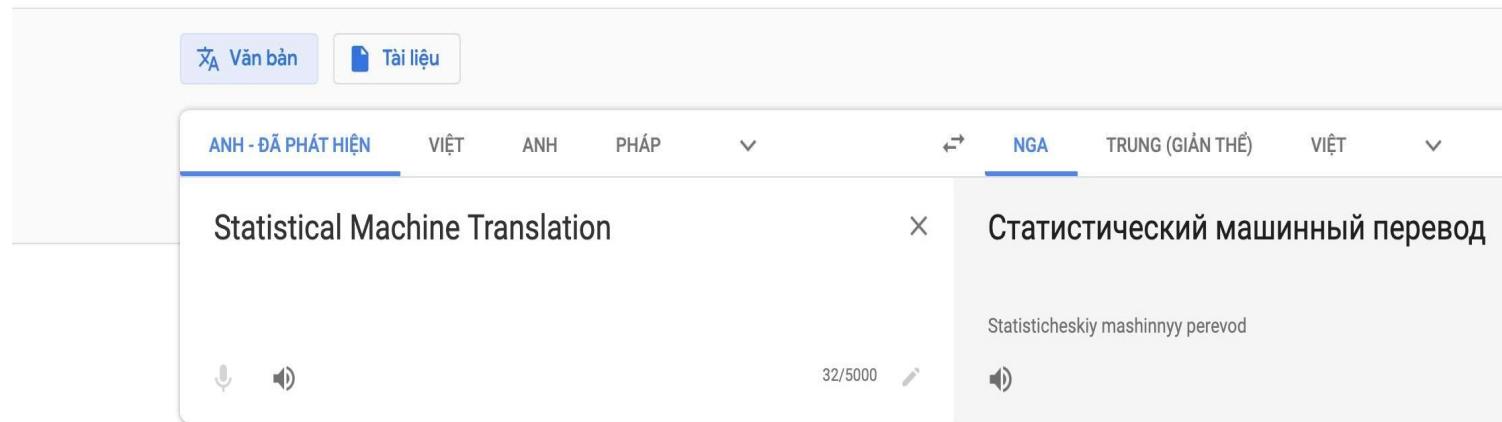
---

- Nhận diện giọng nói (Speech recognition)
  - Trợ lý ảo: Siri (Apple), Echo,Alexa(Amazon), Google Now(Microsoft).
  - Hỗ trợ công việc như: gửi email, đặt lịch hẹn, tìm nhà hàng, cho bạn biết thời tiết, ....vv.
  - Sử dụng mạng học sâu để xử lý nhận dạng giọng nói và hiểu ngôn ngữ tự nhiên.

# Ứng dụng của trí tuệ nhân tạo

- Dịch máy (Machine translation)
  - Ngày nay, Dịch máy thống kê phát triển rộng với số lượng của kho văn bản dịch có sẵn.
  - Dịch máy đang ngày càng cải tiến và đã có những tiến bộ đáng kể.
  - Sequence to Sequence

≡ Google Dịch



# Ứng dụng của trí tuệ nhân tạo

---

- Người máy (Robots)
  - Trên thế giới có nhiều người máy tuyêt vời như NAO, ASIMO, Sophia,...vvv
  - Danh sách 10 người máy tuyêt vời trên thế giới

<https://www.youtube.com/watch?v=u3vdgJVyKeg>



# Ứng dụng của trí tuệ nhân tạo

- Hệ thống gợi ý(Recommender system)
  - Amazon, NetFlix

The screenshot shows an Amazon product page for a Delidigi Wall Mount ABS Echo Dot Bracket Holder Shelf (Built-in Cable Management). The main image displays the Echo Dot device mounted on the bracket. The page includes a sidebar with frequently bought together items like a cable management stand and an Echo Dot. Below the main image, there's a section for 'Customers who viewed this item also viewed' featuring various other Echo Dot accessories.

Product details:

- Price: £8.99
- Voucher:  Apply 10% voucher Details
- Pay £8.99 £0.00: get a £20 Amazon Gift Card on approval for the Amazon Platinum Note: This item is eligible for click and collect. Details
- Colour Name: White
- £8.99
- £8.99

Product description:

- ALL NEW DESIGN: Especially designed for Amazon Echo Dot 3rd Generation, the latest model released in 2018.
- WIRELESS CHARGING: The Echo Dot 3rd Generation has a built-in wireless charging pad.
- CABLE MANAGEMENT: Built-in cable management, perfectly hides the charging cable.
- DRILL TO INSTALL: Comes with screws and anchor. Install it in the bedroom/bathroom.
- WARRANTY: 30-Day Money Back Guarantee & 1 Year Replacement Warranty (Echo Dot 3rd Gen).

Compare with similar items

Wall Mount Holder [Shop now](#)

Hole-free Metal Wall Mount Wall Stand [Shop now](#)

Ad feedback

Frequently bought together

Total price: £58.98 [Add both to Basket](#)

These items are dispatched from and sold by different sellers. Show details

This item: Delidigi Wall Mount ABS Echo Dot Bracket Holder Shelf [Built-in Cable Management] for Dot 3rd... £8.99

Echo Dot (3rd Gen) - Smart speaker with Alexa - Charcoal Fabric £49.99

Customers who viewed this item also viewed

- Magnetic Wall Mount Compatible with Echo Input, in Bedrom, Kitchen, Bathroom, Kitchen, ... [View details](#)
- Hole-free Metal Wall Mount Stand Holder for Dot 3rd Generation [View details](#)
- AhaStyle ABS Wall Mount Stand Hanger [Need to Drill] [Built-in Cord...] [View details](#)
- Alexa Echo 3rd Generation Echo Wall Mount 3rd Generation [Need to Drill] [Built-in Cord...] [View details](#)
- Small Wall Shelf, COMSOON Wall Mount Shelves for Sonos Play:1, Google Home, Cell Phone, ... [View details](#)
- Suptek Smart Speaker Mount for Echo Dot 1st and 2nd Generation, Google Home, Cell Phone, ... [View details](#)
- AhaStyle ABS Wall Mount Stand Hanger [Need to Drill] [Built-in Cord...] [View details](#)
- LinTimes Dot 3rd Gen Holder Owl Table Mount Stand for Smart Speakers 7. White [View details](#)
- Gelink Wall Mount Shelf - Holds Any Devices Up to 150g in Seconds - Integrated... [View details](#)
- Bovon Table Holder for Echo Dot 3rd Generation, 360° Rotatable, ... [View details](#)
- Stouchi Echo Dot 3rd Wall Mount Holder Stand Compatible with Echo Dot 2nd / 3rd Generation... [View details](#)
- Wall Mount Shelf Holder Stand for E (2nd / 3rd Generation)... [View details](#)

# Ứng dụng của trí tuệ nhân tạo

- Hệ thống tìm kiếm (Search engines)
  - Google, Yandex

The screenshot shows a Google search results page. The search query is "Artificial Intelligence A-Z™: Learn How To Build An AI | Udemy". The results are as follows:

- Quảng cáo · www.udemy.com/**  
**Artificial Intelligence A-Z™: Learn How To Build An AI | Udemy**  
Learn Data Science Online At Your Own Pace. Start Today and Become an Expert in Days. Join Over 40 Million People Learning Online with Udemy. 30-Day Money-Back Guarantee! View Our Courses. Lifetime Access. Download Our Mobile App. Expert Instructions.
- Trending New Courses**  
Find Out What is New In Your Field  
Start Learning a New Skill Today
- Teach the World Online**  
Share Your Knowledge, Make Money  
Reach Students Across The Globe
- Quảng cáo · executive-education-online.mit.edu/MIT-Sloan/Online-Course** ▾ +1 617-997-4979  
**MIT Sloan Online AI Course | Earn Your Certificate Online**  
Learn How To Transform Your Organization with AI. Apply to MIT's Online Course Today. Become a Leader in Artificial Intelligence Business Strategy, Online at MIT. Apply Now! Manage AI. 6 Week Program. MIT Certificate. Personalized Support.

# Ứng dụng của trí tuệ nhân tạo

---

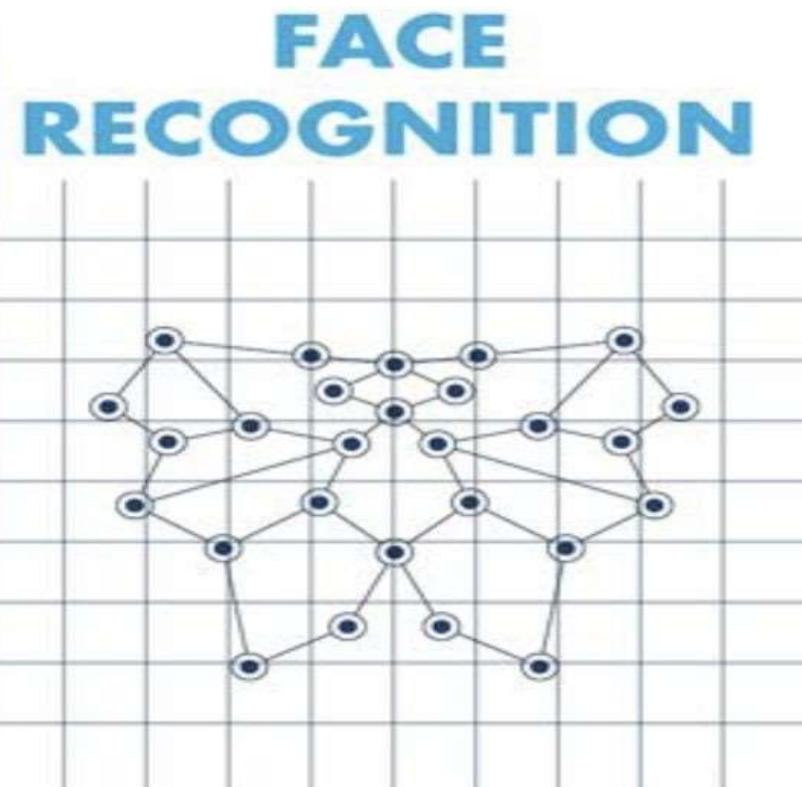
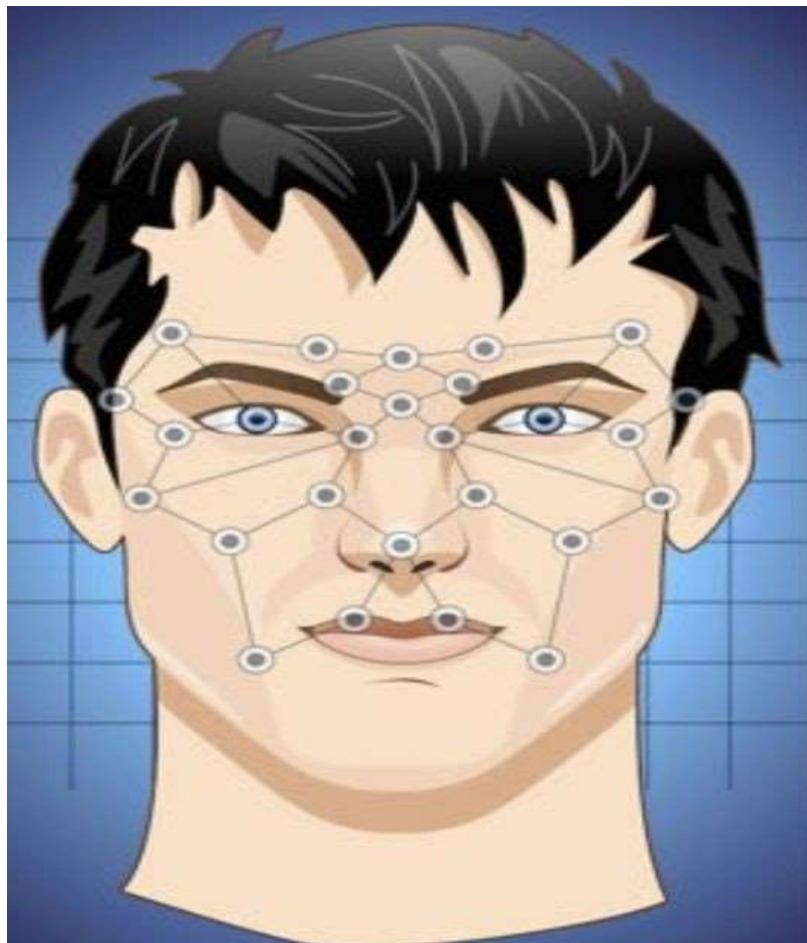
- Phát hiện khuôn mặt (Face detection)
  - Viola-Jones method.



# Ứng dụng của trí tuệ nhân tạo

---

- Nhận dạng khuôn mặt (Face recognition)



**PROCESSING**

# Ứng dụng của trí tuệ nhân tạo

---

- Cờ vua (Chess)
  - 1997, IBM Deep Blue đã thắng kiện tướng cờ vua Kasparov
  - Thuật toán tìm kiếm



# Ứng dụng của trí tuệ nhân tạo

---

- Trò chơi trên truyền hình trên kênh Jeopardy
  - 2011, IBM Watson đã thắng cả Brad Rutter và Ken Jennings
  - Hiểu ngôn ngữ tự nhiên và kỹ thuật trích thông tin



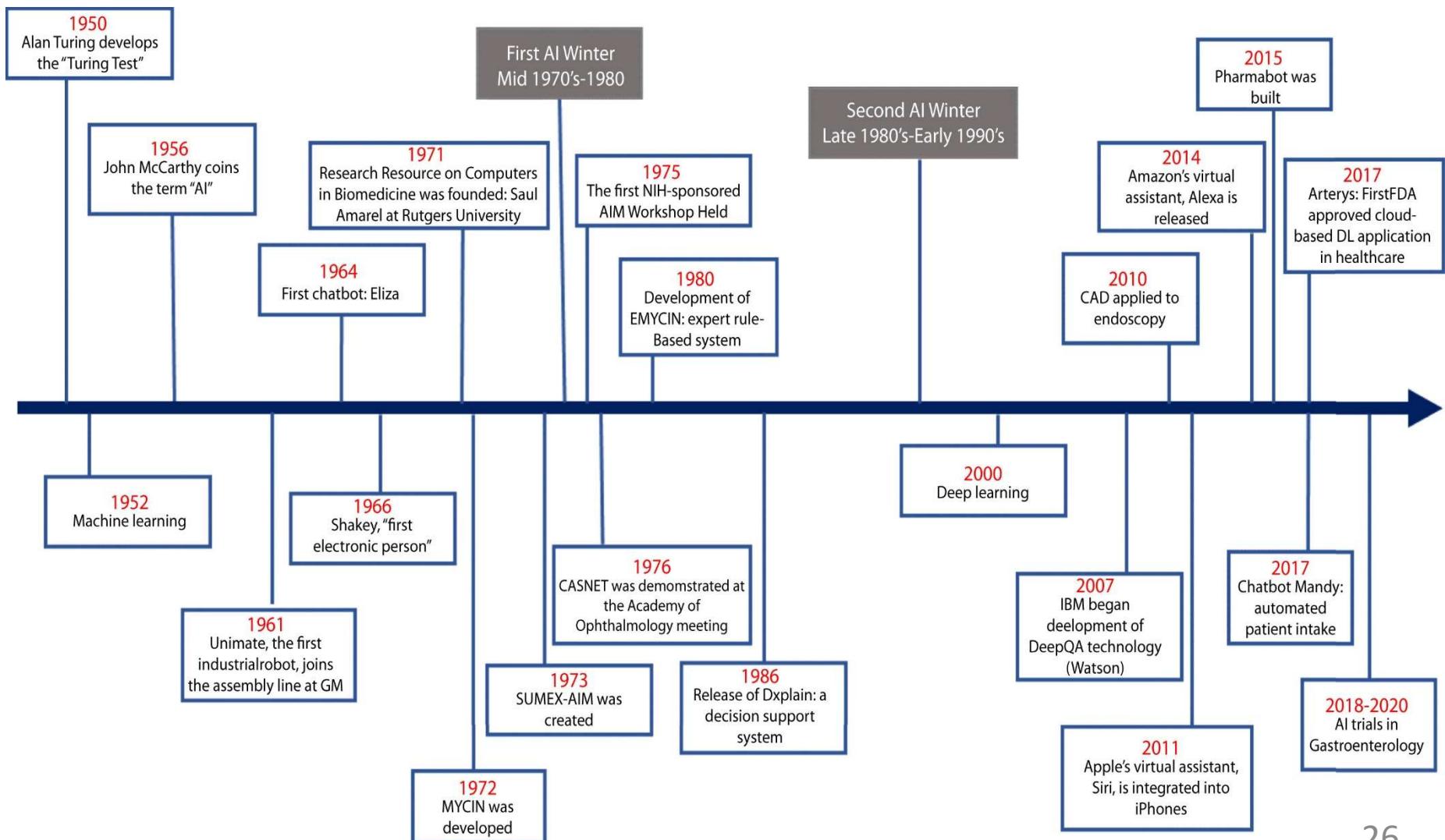
# Ứng dụng của trí tuệ nhân tạo

---

- Cờ vây
  - 2016, Google AlphaGo đã thắng kiện tướng cờ vây Lee Sedol
  - Sử dụng kỹ thuật học sâu, kỹ thuật học tăng cường và kỹ thuật tìm kiếm

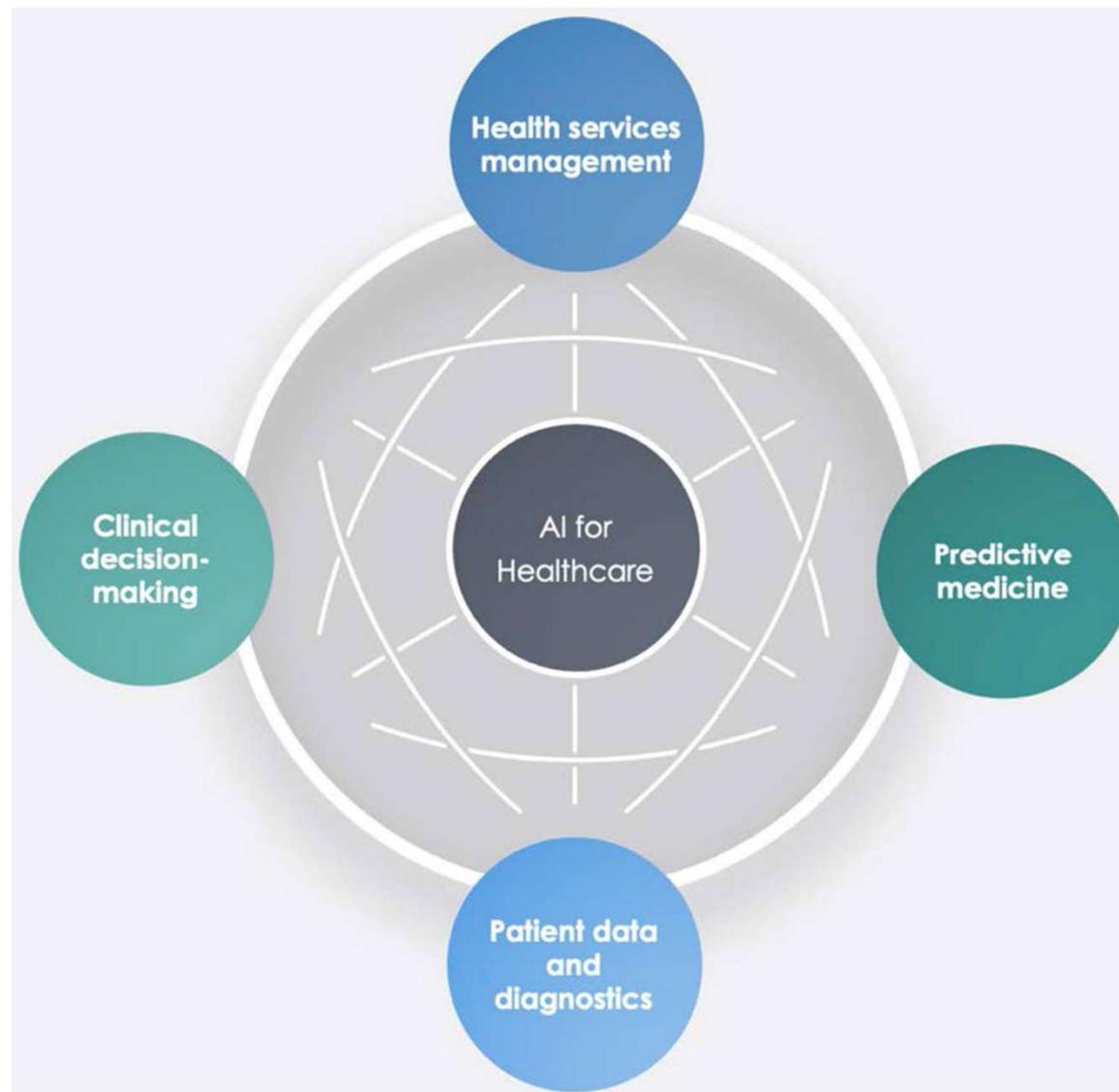


# Trí tuệ nhân tạo trong lĩnh vực y tế

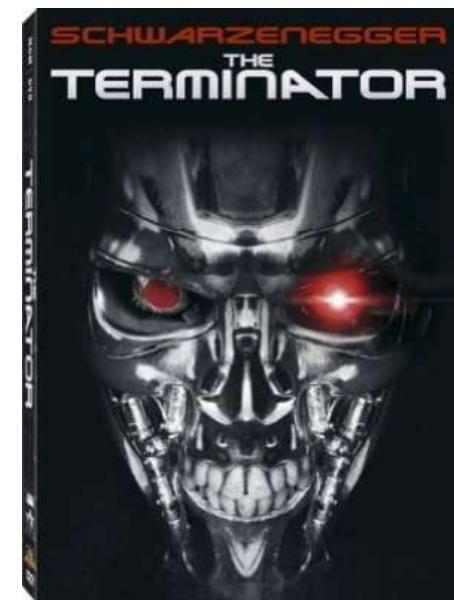
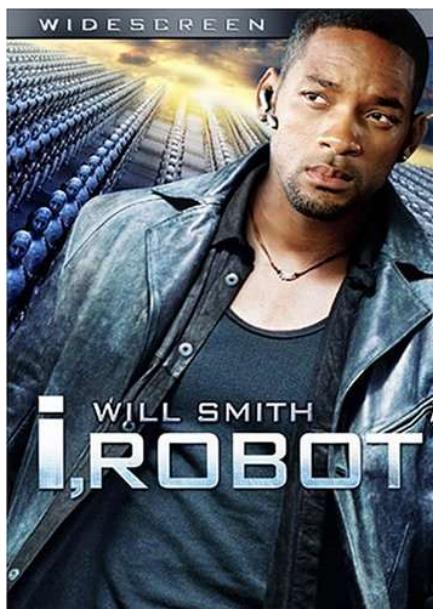
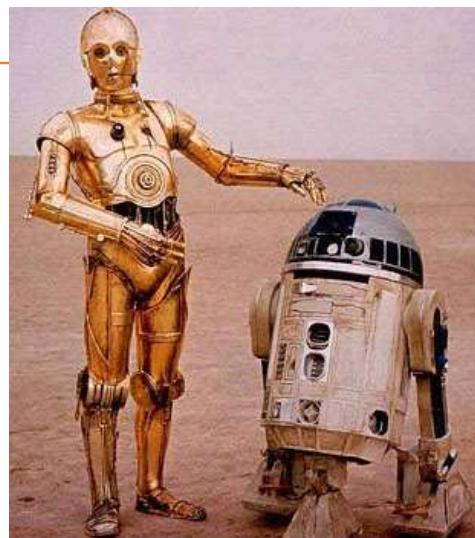


# Trí tuệ nhân tạo trong lĩnh vực y tế

---



# Artificial Intelligence in the Movies



# Why the interest in AI?



Labor



Science



Appliances



Search engines



Medicine/  
Diagnosis

What else?



# A.L.I.C.E. AI Foundation

*New: Find a Bot in the A. I. Foundation Bot Directory!*

*Begin your journey here:* Chat with a bot!



[A. L. I. C. E.](#)

[DAVE E.S.L. bot](#)

[DAVE E. S. L. Bot](#)

[C.L.A.U.D.I.O Personality Test](#)

[Take the C. L. A. U. D. I. O. Personality Test](#)

*Learn how to become a  
Botmaster:*

[Create your own A. I. chat robot...](#)

[GET \[V\]HOST™ AVATARS](#)

[Host your bot](#)

[A.L.I.C.E Silver Edition](#)

[Join the A. I. Foundation](#)

[Download Free AIML Software](#)



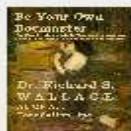
[Get Documentation about AIML and ALICE software](#)

*Launch your career in  
Artificial Intelligence:*

[Make Money with your bot.](#)

[Read the Testimonials](#)

[Take the TAO of AIML Seminar](#)



[Read \*Be Your Own Botmaster\* by Dr. Richard S. Wallace](#)

## ALICEBOT

ALICE A.I. FOUNDATION NEWS ITEMS, ANNOUNCEMENTS, PRESS RELEASES AND OPINIONS. ANYTHING RELATED TO ARTIFICIAL INTELLIGENCE MARKUP LANGUAGE (AIML), THE TURING TEST, THE LOEBNER PRIZE, BOT HOSTING SERVICES, FREE AND PROPRIETARY CHAT BOT SOFTWARE.



FOLLOW DRWALLACE ON TWITTER



Dr. Richard Wallace  
[drwallace](#)

Virtual DJ Denise is the First Artificial Intelligent Radio Host  
<http://t.co/A7sSNHh>  
8 days ago · reply · retweet · favorite

Robot teaches English as Second Language <http://t.co/6MpwQip>  
10 days ago · reply · retweet · favorite

Superbot Offer – Free Consulting <http://t.co/NEC9ceT>  
44 days ago · reply · retweet · favorite

Emma the Cybrarian:  
<http://t.co/eIWJMzr>  
57 days ago · reply · retweet · favorite

ALICE, Dick Cavett, R.L. Stine and me on the WFMU radio show 7 Second Delay  
<http://t.co/YyPhFOB>  
85 days ago · reply · retweet · favorite



WEDNESDAY, JULY 27, 2011

### AIML Superbot Special Offer - Free Consulting

The AIML Superbot is a blank template that helps you create a custom chat bot from scratch. If you want to create a bot with its own proprietary, unique personality, the Superbot helps you by providing a rich set of patterns that allow you to "fill in the blanks" with your own bot responses.

Building a bot from scratch can still be a daunting task for those new to AIML and chat bot creation. That is why we are now offering 2 hours of free consulting time with your Superbot purchase. You will be able to talk directly by phone or Skype with Dr. Richard S. Wallace or another AIML expert and receive support and help with your bot project.

Use your consultation time to get specific answers to all your questions about bot content creation and help with your project, including help linking the bot to speech recognition systems, avatars, speech synthesis and other third-party applications.

This offer expires on October 1, 2011.

POSTED BY DR. RICHARD S. WALLACE AT 8:37 AM 2 COMMENTS  
LABELS: AIML, AVATARS, BOT HOSTING, CHATBOTS, CONSULTING, SUPERBOT, TEXT TO SPEECH

[Dragon Medical Practice Sale | Dragon Medical Practice | Free shipp](#)

[Passion for Design? Turn Your Passion into a Profession at The Art](#)

[Top 10 HRIS Software 2011 Top 10 Human Resource Software Vendi](#)

[Microsoft® Office 365 Access Email, Documents & Calendars Anytim](#)



FRIDAY, MAY 27, 2011

### New Release of ALICE AIML on Google Code



Click here to chat with Talking Animated Fake Captain Kirk

#### SUBSCRIPTION BOTS

[A.L.I.C.E. Silver Edition](#)

[CLAUDIO Personality Test](#)

[DAVE E.S.L. Bot](#)

#### MORE AIML RESOURCES

[Pandorabots](#)

[Oddcast Avatars](#)

[Robot-Hosting](#)

[Riot Software](#)

#### MORE BOT RESOURCES

[Wikipedia](#)

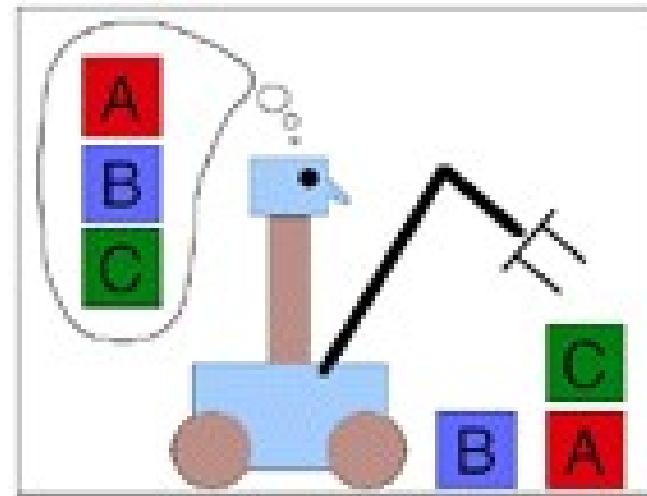
[Agentland](#)

[Open Directory](#)

[Virtual Humans](#)

# Nền tảng cơ sở của trí tuệ nhân tạo

- Trí tuệ nhân tạo kế thừa các ý tưởng, quan điểm và các kỹ thuật từ các ngành học khác nhau
  - Triết học
  - Toán học
  - Kinh tế học
  - Khoa học thành kinh
  - Tâm lý học
  - Kỹ thuật máy tính
  - Điều khiển học
  - Ngôn ngữ học



# Nền tảng cơ sở của trí tuệ nhân tạo

---

- Triết học (Philosophy)

- Xây dựng logic và kỹ thuật suy luận
- Xây dựng hệ thống hoạt động như một tập luật
- Xác định cơ sở của việc học, ngôn ngữ và hành động

- Toán học (Mathematics)

- Xây dựng Lôgic
- Xây dựng thuật toán, quy trình tính toán
- Xác suất thống kê

- Kinh tế học (Economics)

- Ra quyết định
- Lý thuyết trò chơi
- Chuỗi markov

# Nền tảng cơ sở của trí tuệ nhân tạo

---

- Khoa học thành kinh (Neuroscience)
  - Nghiên cứu chức năng của Não
  - Phân biệt sự giống và khác giữa Não và Máy tính
- Tâm lý học (Psychology)
  - Làm thế nào để suy nghĩ và hành động
  - Xem bộ não là cỗ máy xử lý thông tin
  - Làm thế nào để máy tính có thể học ngôn ngữ, ghi nhớ, suy nghĩ
- Kỹ thuật máy tính(Computer engineering)
  - Xây dựng cỗ máy mạnh để thông minh thêm(DeepBlue)
  - Minh họa ô tô tự lái

# Nền tảng cơ sở của trí tuệ nhân tạo

---

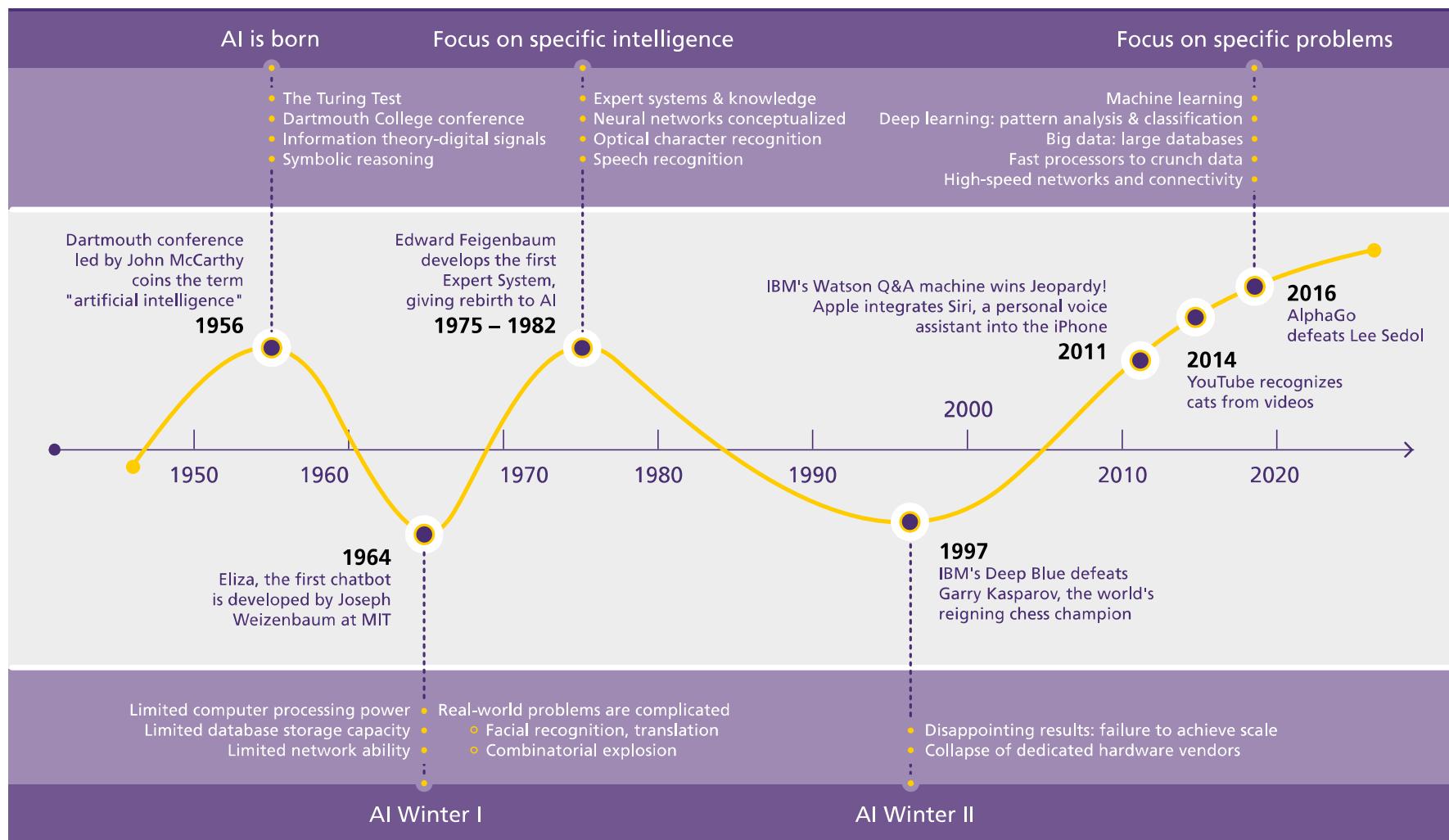
- Điều khiển học (cybernetics)

- Thiết kế tối ưu tác tử nhận phản hồi từ môi trường
  - Thiết kế hệ thống điều khiển tối ưu mục tiêu theo thời gian

- Ngôn ngữ học (Linguistics)

- Xây dựng mối quan hệ giữa “ngôn ngữ” và “suy nghĩ”
  - Kết hợp giữa ngôn ngữ + trí tuệ nhân tạo = xử lý ngôn ngữ tự nhiên

# Lịch sử của trí tuệ nhân tạo



# Lịch sử của trí tuệ nhân tạo

---

- **1940-1950: giai đoạn tiền AI**

- McCulloch & Pitts: Mạch Boolean để mô hình não bộ
- Máy tính toán Turing và Trí thông minh

<http://www.turingarchive.org/viewer/?id=463&title>

=1

- **1950-1970: Giai đoạn tham vọng**

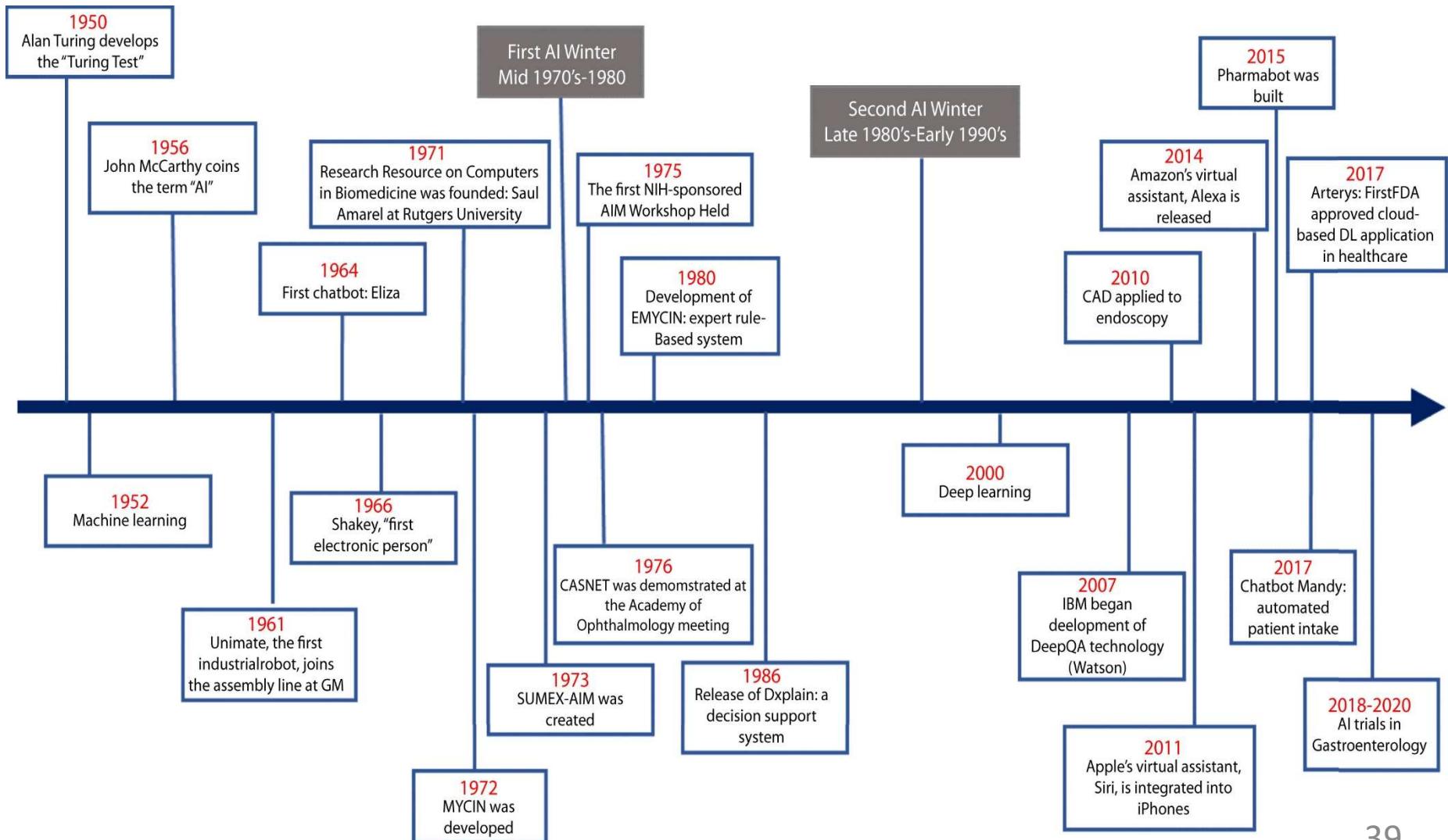
- Hội nghị **AI-Summer**, @Dartmouth, Mỹ - 1956
- Tham vọng máy tính hiểu được con người
- Phương thức giao tiếp giữa người và máy tính
- Phương thức biểu diễn tri thức như mạng ngữ nghĩa, đồ thị khái niệm

# Lịch sử của trí tuệ nhân tạo

---

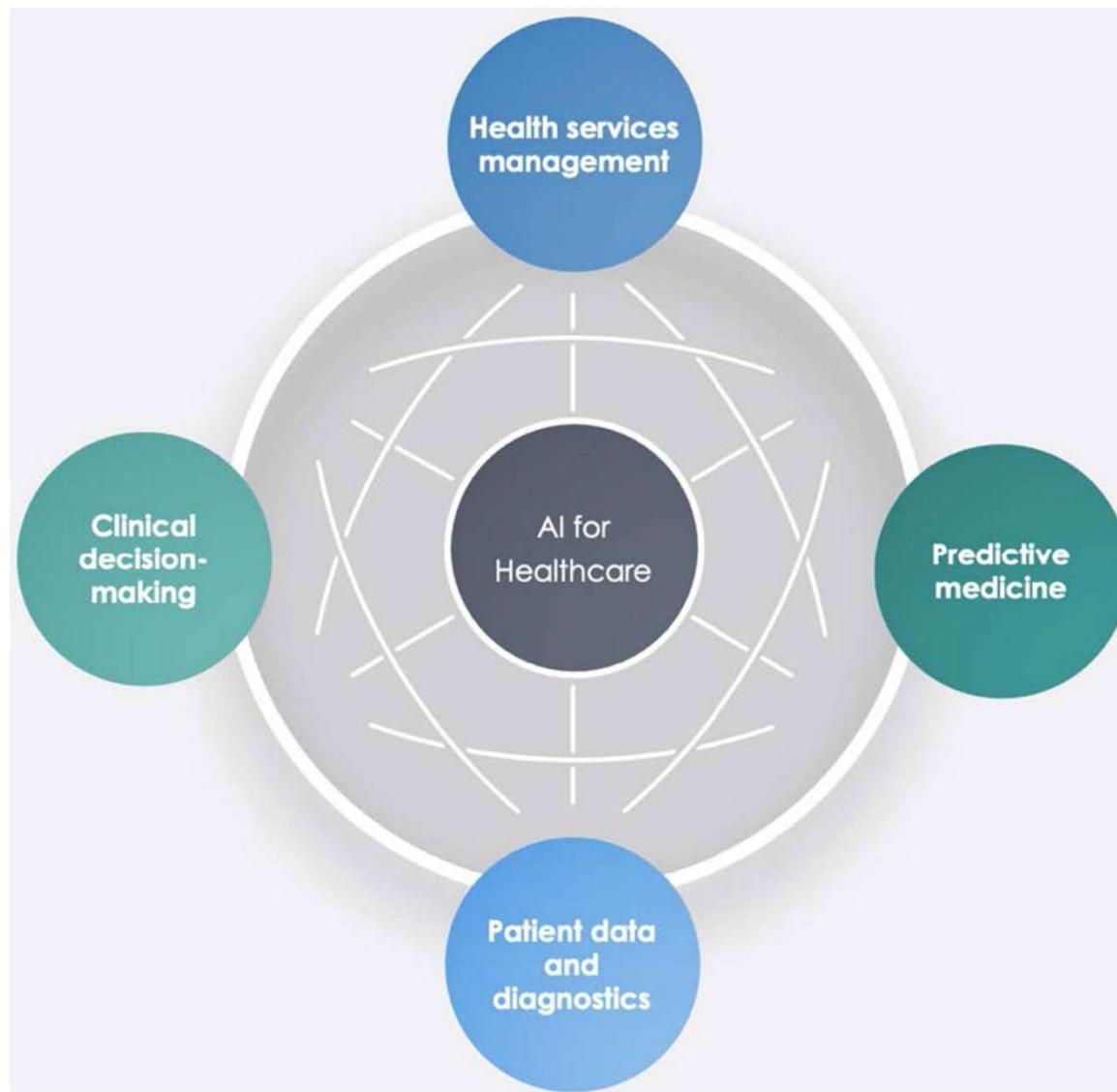
- **1970-1990: Giai đoạn dựa vào tri thức** (Knowledge based AI)
  - Hệ chuyên gia, AI trở thành một ngành công nghiệp
  - **Hội nghị AI-Winter**
- **1990 đến nay:** phương pháp tiếp cận khoa học
  - Mạng lưới thần kinh
  - Trí tuệ nhân tạo trở thành ngành khoa học, sử dụng xác suất để mô phỏng tính không chắc chắn
  - **Hội nghị AI - Spring, 2013**
  - Sự xuất hiện các bộ dữ liệu rất lớn(Big Data): Dữ liệu sẽ thúc đẩy những khám phá trong tương lai

# Trí tuệ nhân tạo trong lĩnh vực y tế



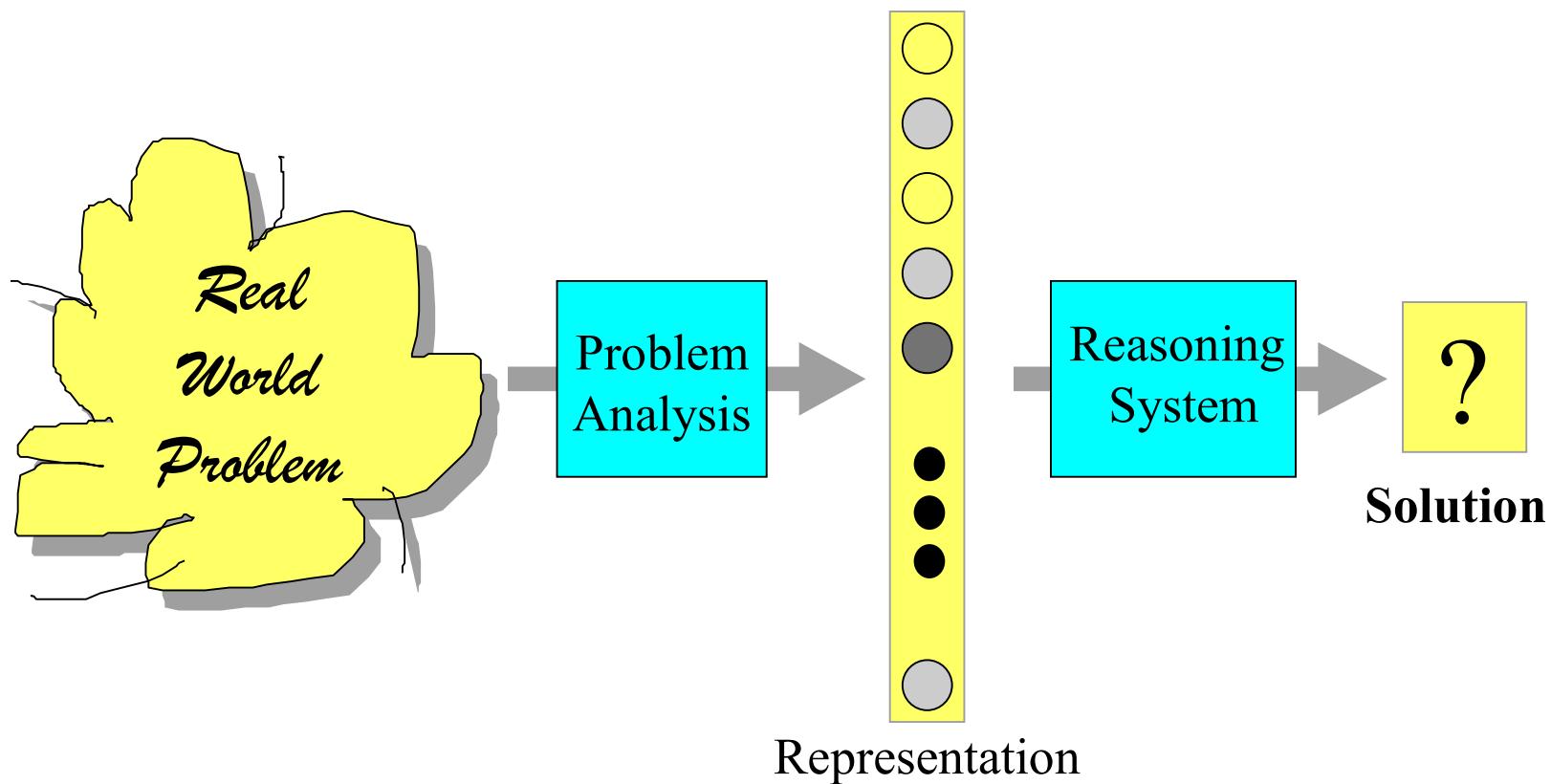
# Trí tuệ nhân tạo trong lĩnh vực y tế

---

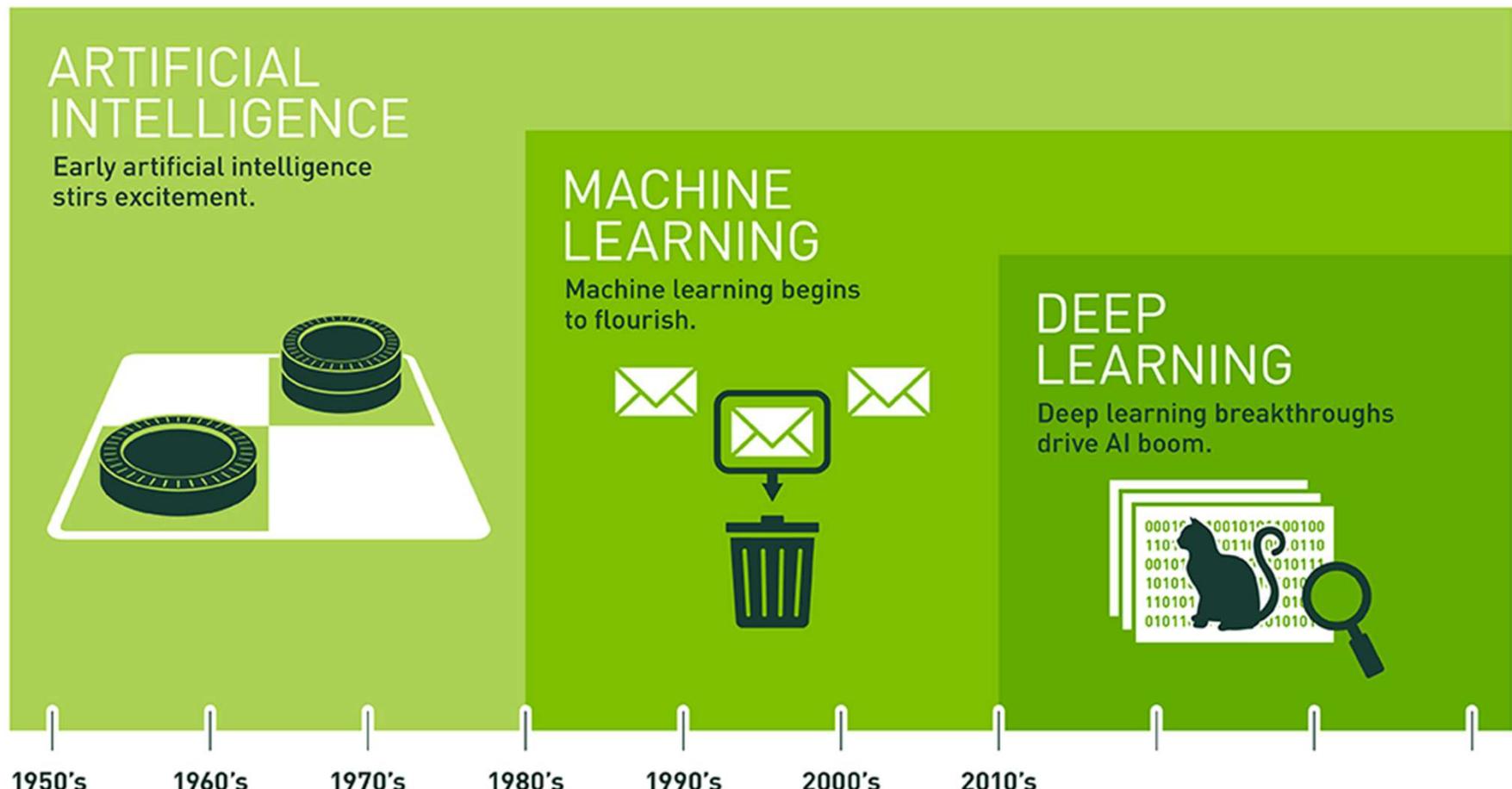


# A model of knowledge-based systems development

---



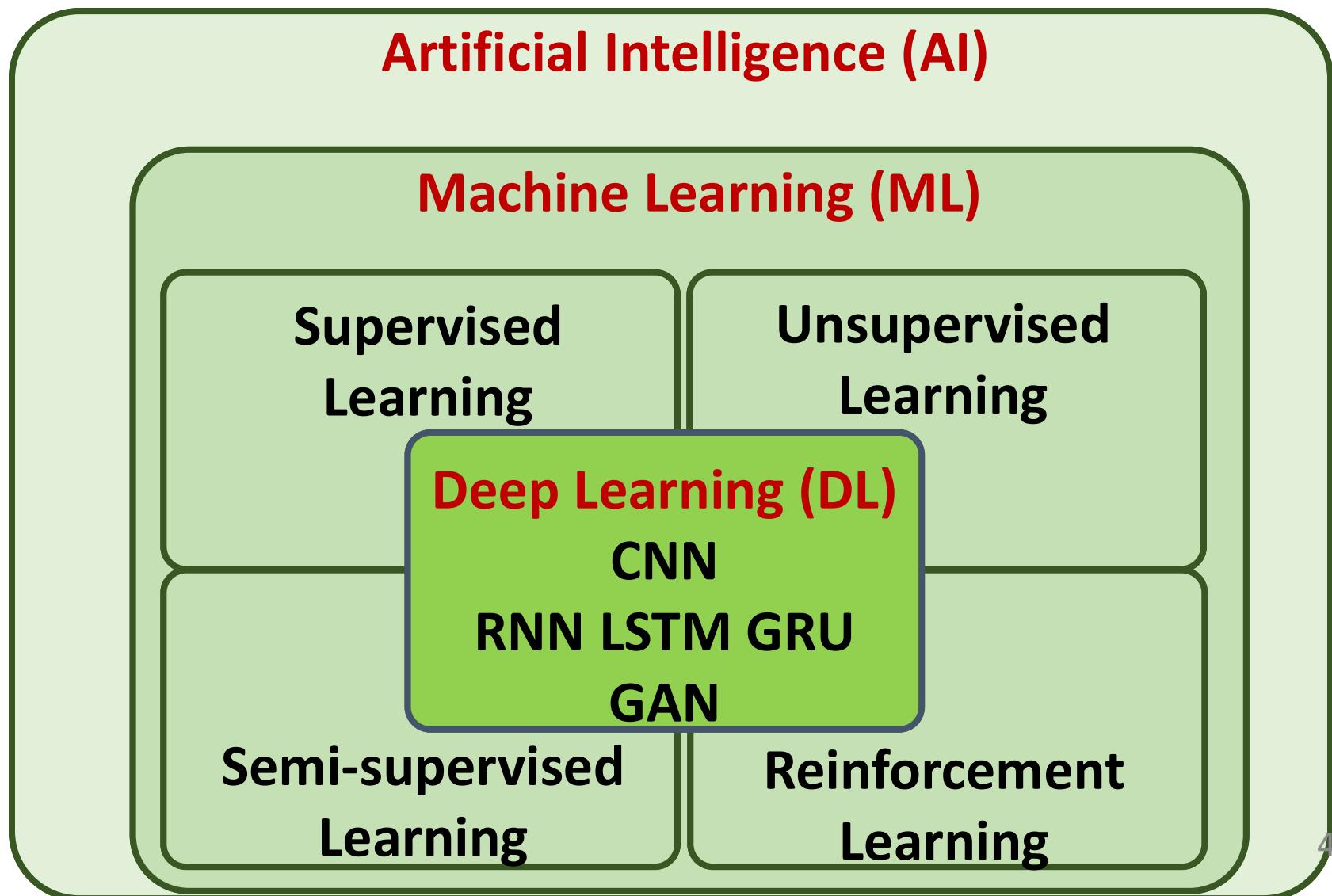
# Artificial Intelligence Machine Learning & Deep Learning



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

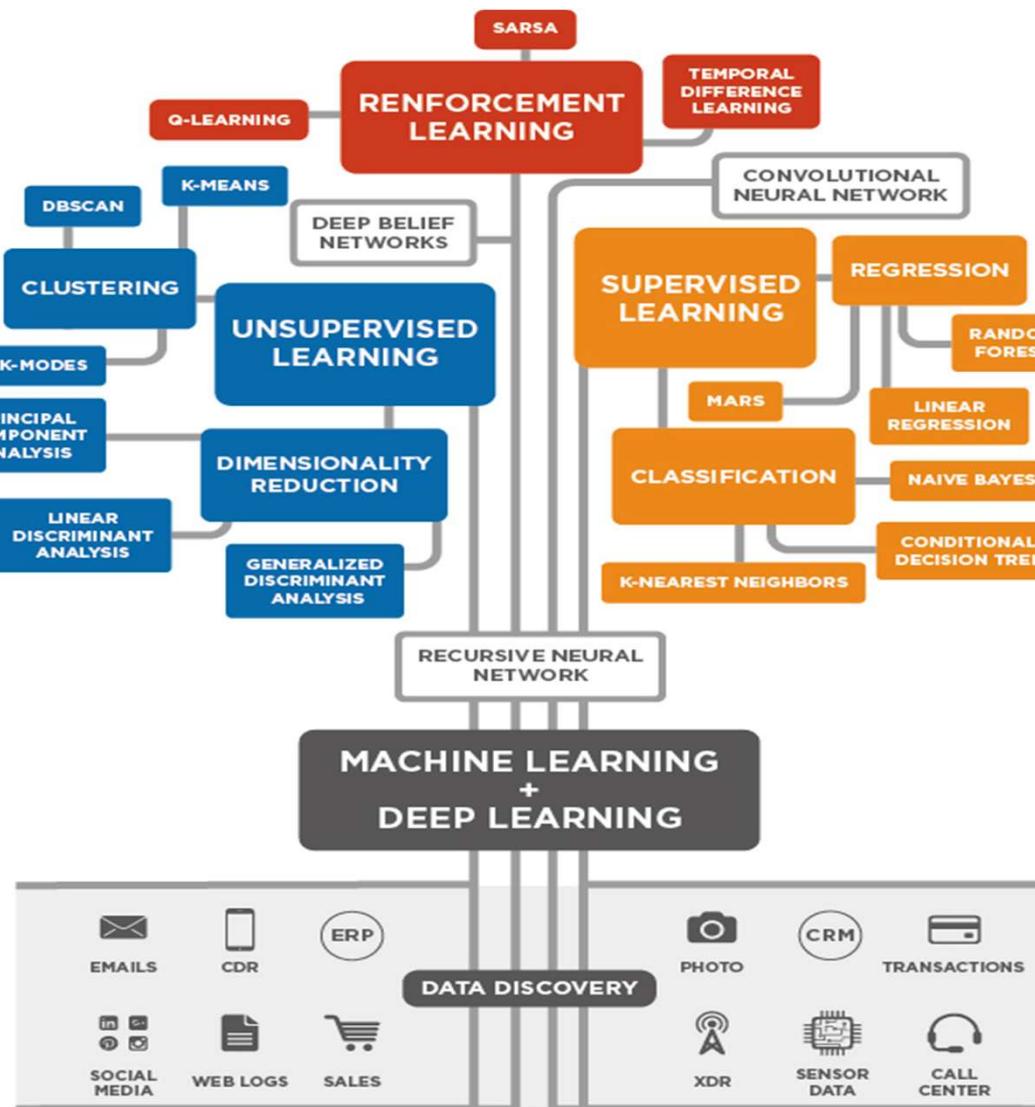
# Artificial Intelligence

## Machine Learning & Deep Learning

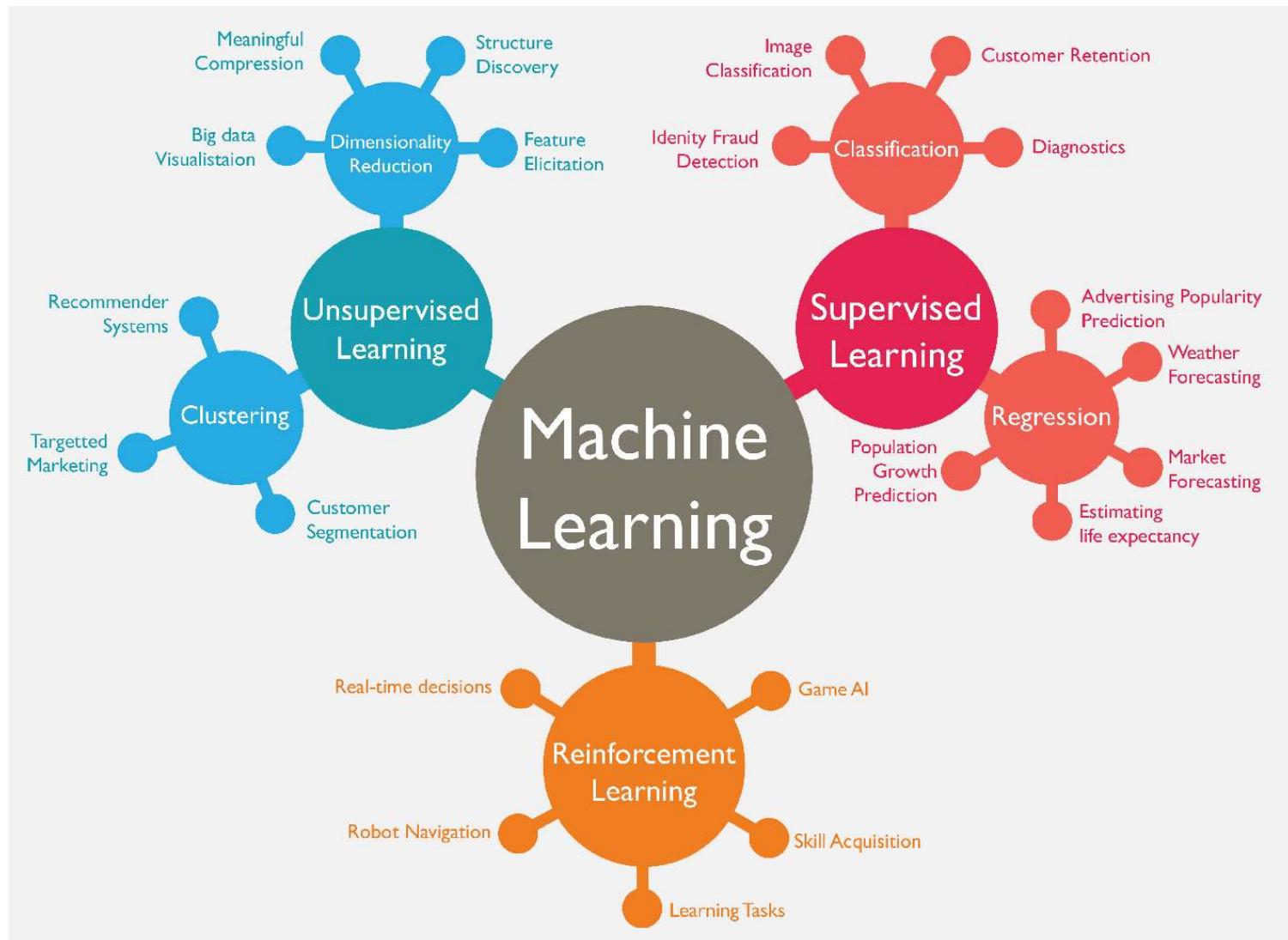


# Artificial Intelligence

## Machine Learning & Deep Learning

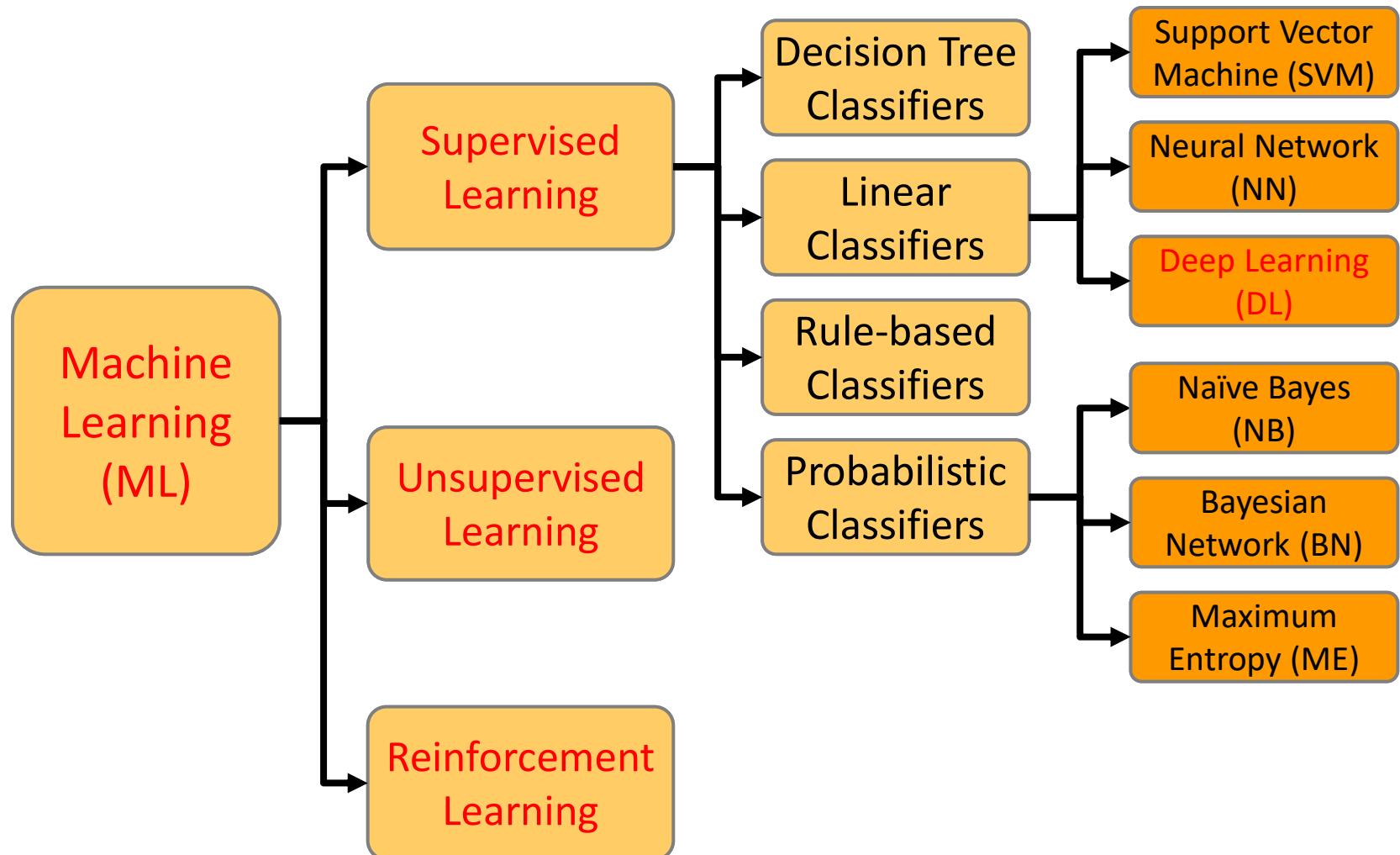


# Machine Learning

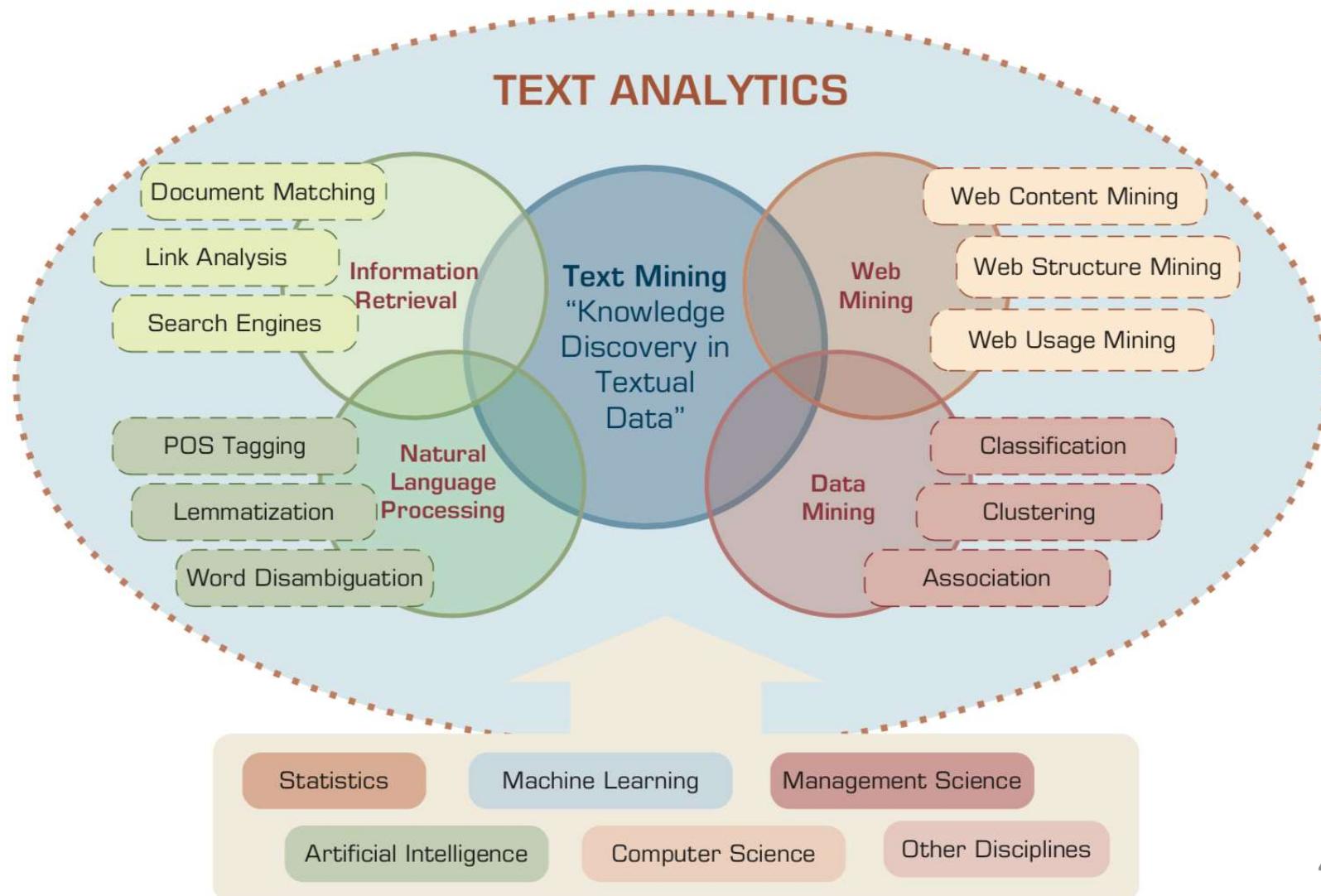


# Machine Learning (ML) / Deep Learning (DL)

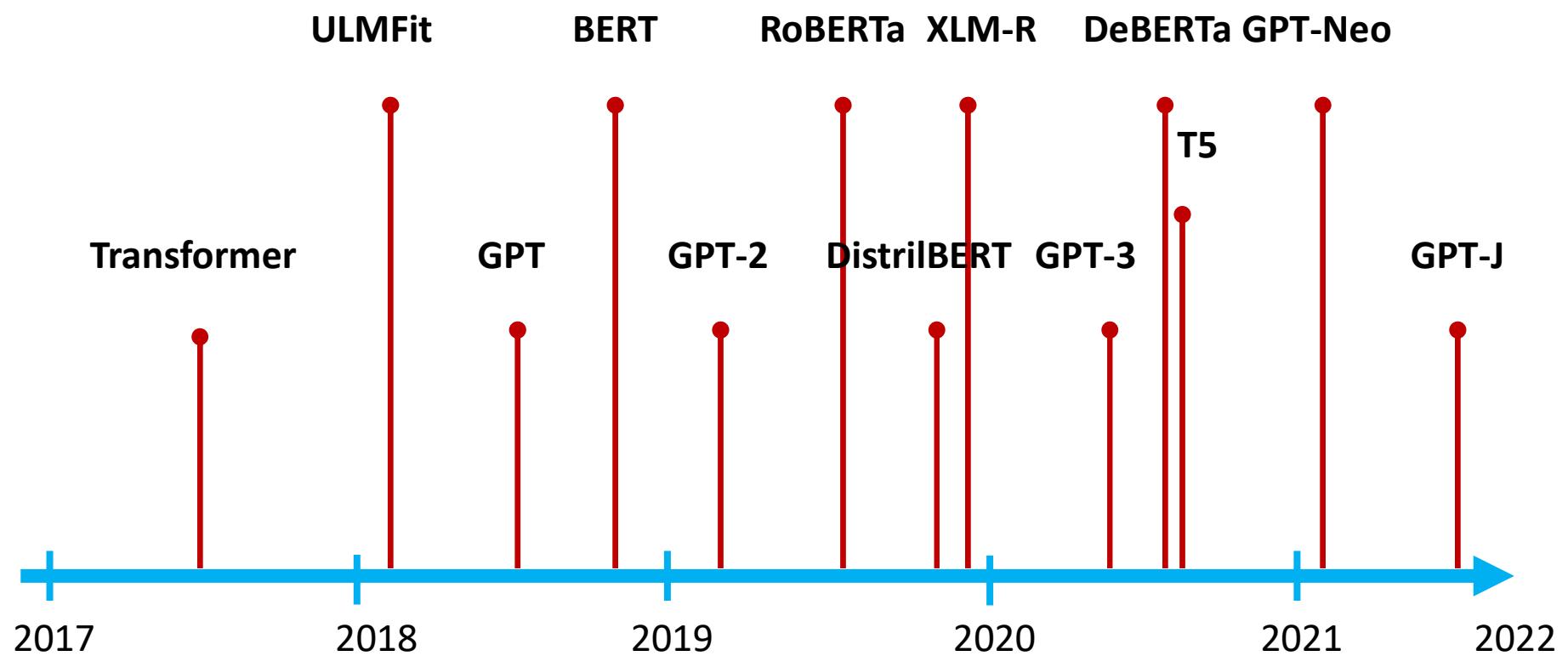
---



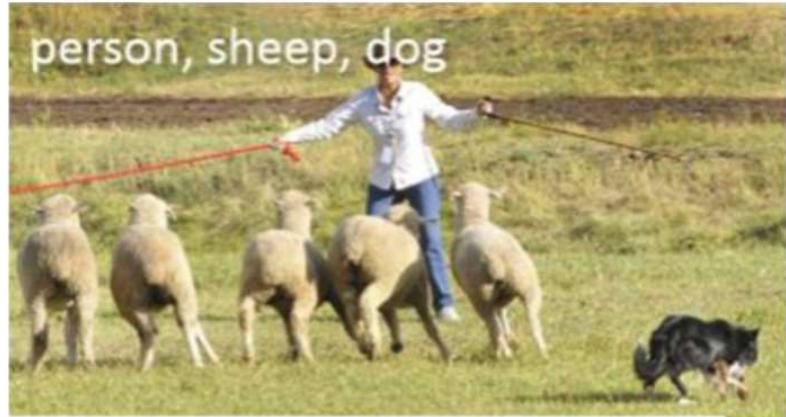
# AI for Text Analytics



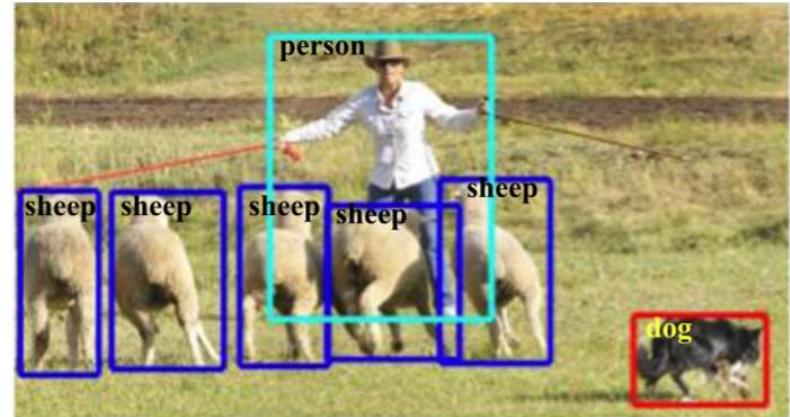
# The Transformers Timeline



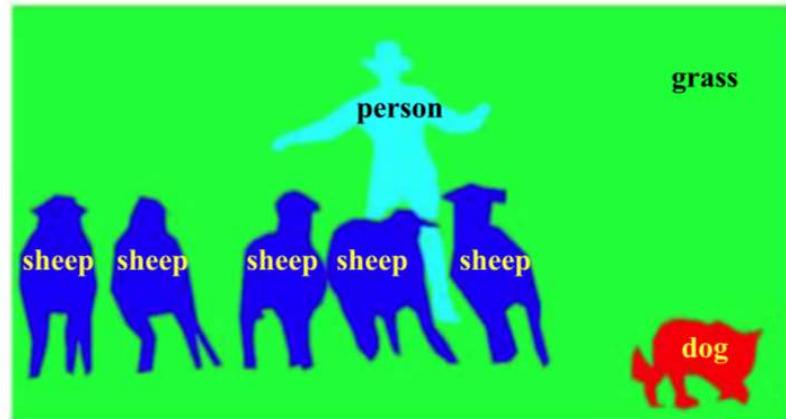
# Computer Vision: Object Detection



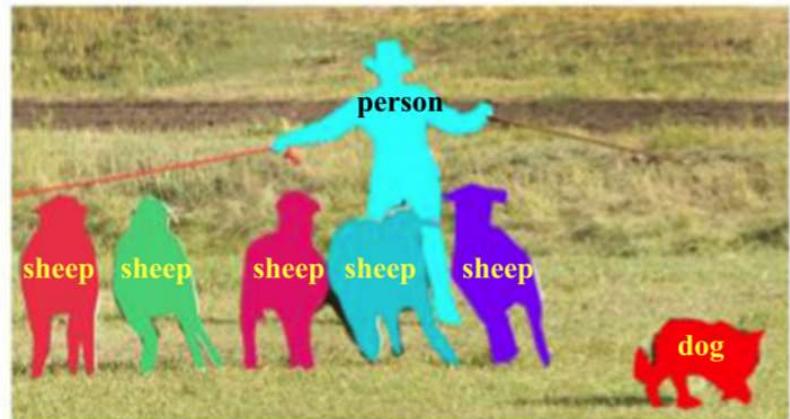
**(a)** Object Classification



**(b)** Generic Object Detection  
(Bounding Box)

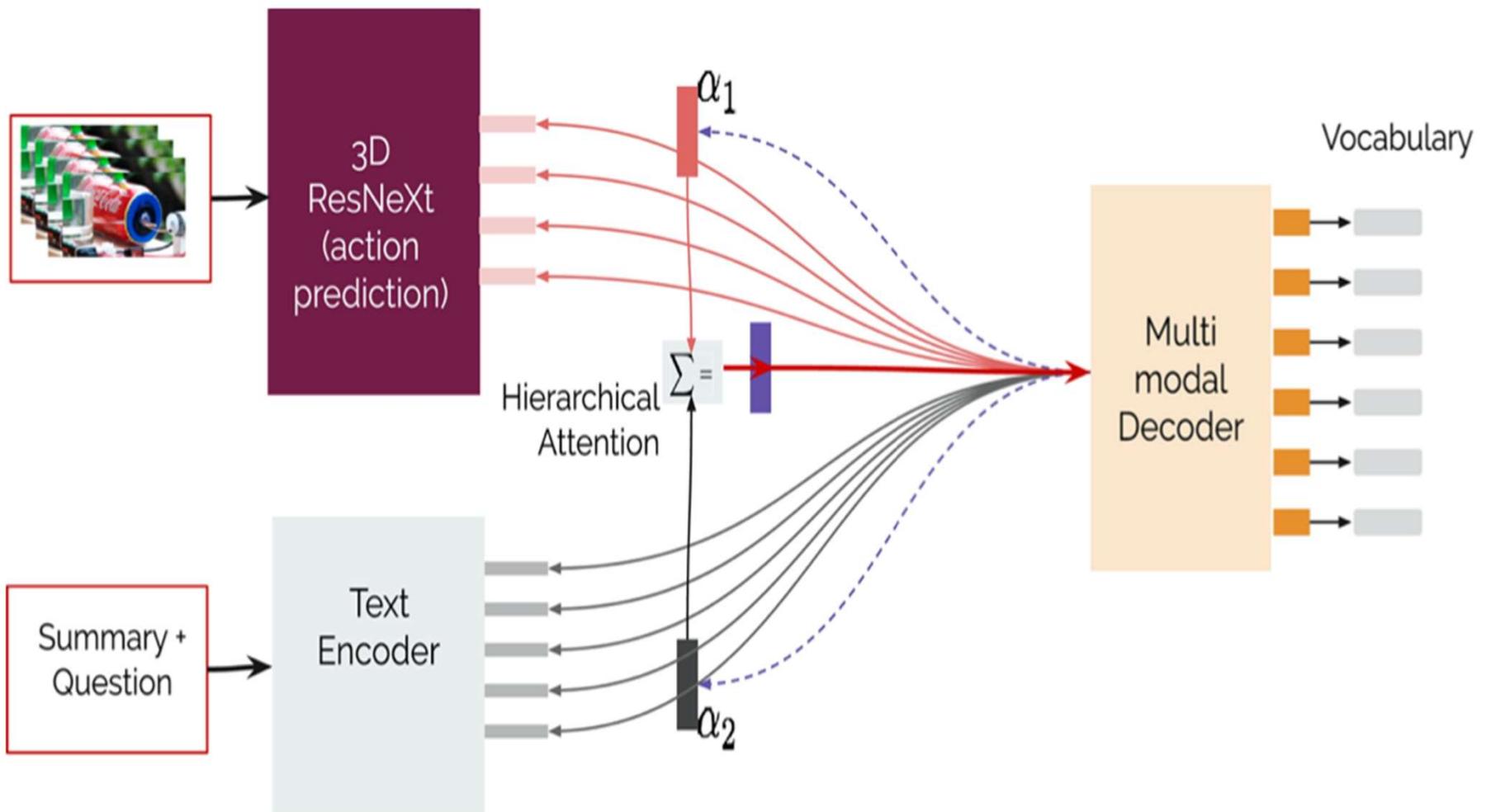


**(c)** Semantic Segmentation

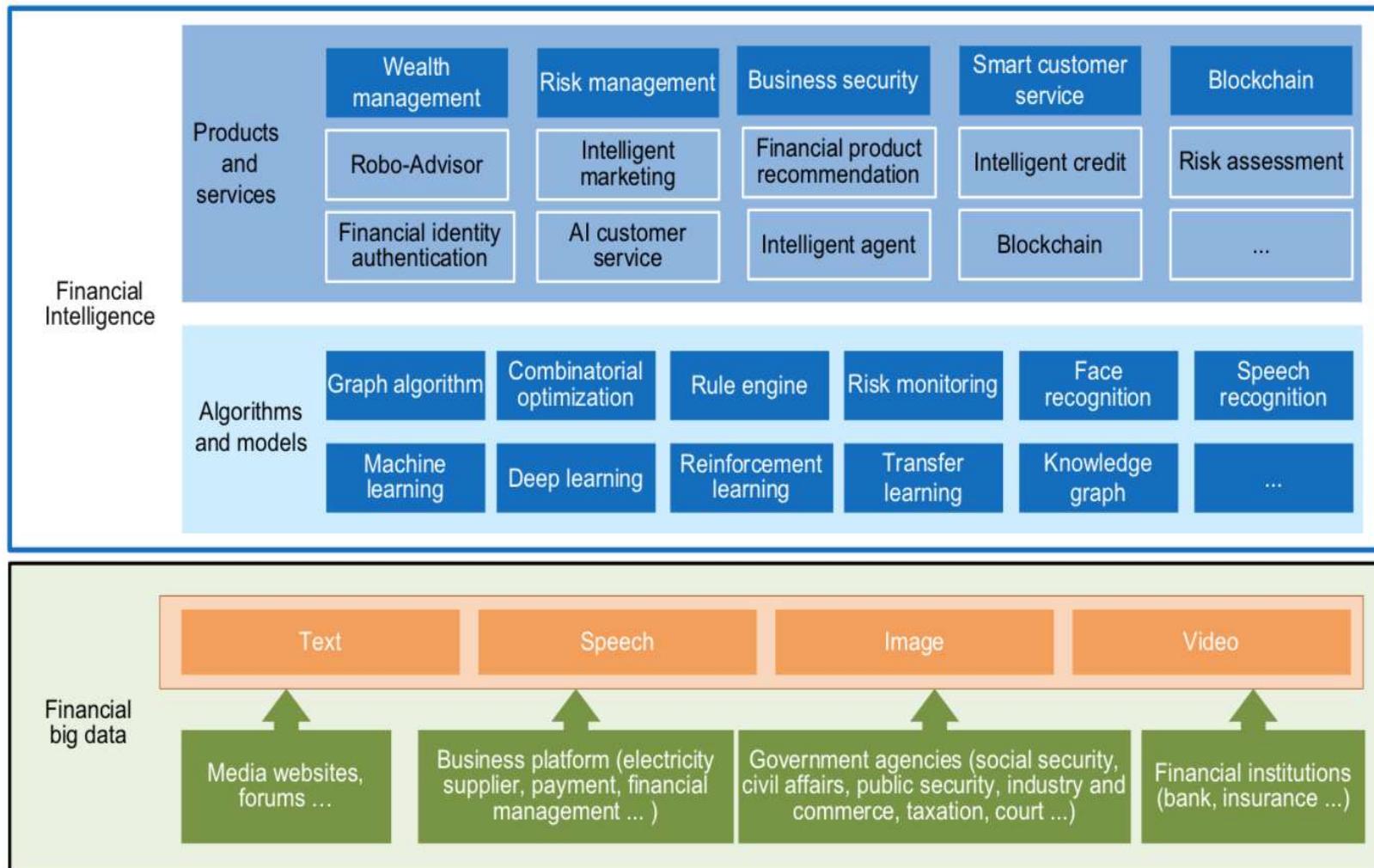


**(d)** Object Instance Segmentation

# Text-and-Video Dialog Generation Models with Hierarchical Attention



# FinBrain: when Finance meets AI 2.0 (Zheng et al., 2019)



# Technology-driven Financial Industry Development

Development stage	Driving technology	Main landscape	Inclusive finance	Relationship between technology and finance
Fintech 1.0 (financial IT)	Computer	Credit card, ATM, and CRMS	Low	Technology as a tool
Fintech 2.0 (Internet finance)	Mobile Internet	Marketplace lending, third-party payment, crowdfunding, and Internet insurance	Medium	Technology-driven change
Fintech 3.0 (financial intelligence)	AI, Big Data, Cloud Computing, Blockchain	Intelligent finance	High	Deep fusion

# Honda Humanoid Robot

---



Walk



Turn



Stairs

# Domestic Robots

---



[Domestic Robot Accessories](#)



[Domestic Robot Replacement Parts](#)



[Robot Vacuums](#)



[Robot Floor Cleaners](#)



[Robot Pool Cleaners](#)



[Robotic Lawn Mowers | Lawnbot](#)  
[Robot Mower](#)



[Robot Pet Care](#)



[Robotic Window Cleaners](#)



[Sweeping Robots](#)



[RobotShop Packages](#)



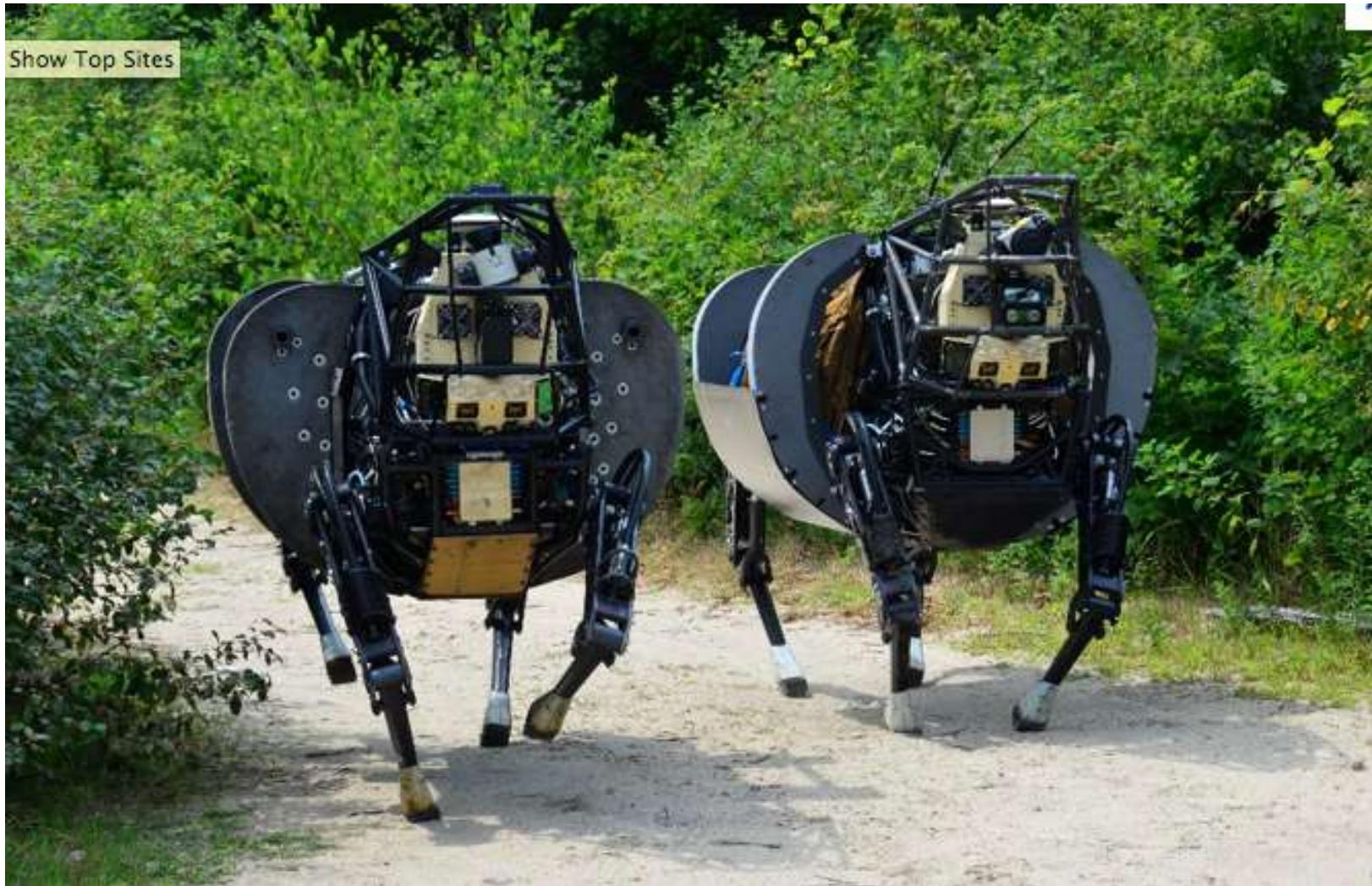
[Robots for Sports](#)



[Surveillance and Companion Robots](#)

# Military robots

---



# [www.aibo.com](http://www.aibo.com)



## AIBO® Entertainment Robot

Official U.S. Resources and Online Destinations



- [Purchase AIBO® Products](#)
- [Software Resources](#)
- [AIBO® News & Events](#)
- [Support](#)



## AIBO NEWS!

October 22, 2003

[ERS-7 Pre-orders have begun!  
FREE SHIPPING!](#)

Only 10 days left for the [\\$200 rebate](#) on select ERS-210A models

[ERS-7 Product Tour](#)  
at [sonystyle.com](http://sonystyle.com)

[ERS-7 Press Release](#)  
[click here](#)

[AIBO DAYS!](#)  
Celebrate AIBO and see special demonstrations!

[AIBO EYES 1.1 patch](#)  
now available

[Check out our AIBO EYES Cartoon \(Manga\)](#)

Yes, I would like to receive information about AIBO®



[Privacy Policy](#) | [Legal Trademark](#)

©1999-2003 Sony Electronics, Inc. All rights reserved



English   Français   Deutsch

Links

AIBO Tips

AIBO History



Downloads

Support

User Guides

News

### Important announcement

Following the Sony Corporation FY05 3Q announcement, the production of AIBO Entertainment Robots has been discontinued as of end March 2006. As for the sales activity, we will discontinue the sales of AIBO once all remaining stock runs out.

[Read more](#)

[Terms and conditions of web site](#)

©2006 Sony Entertainment Robot Europe

[Privacy statement](#)

like.no.other™

SONY



[www.robocup.org](http://www.robocup.org)

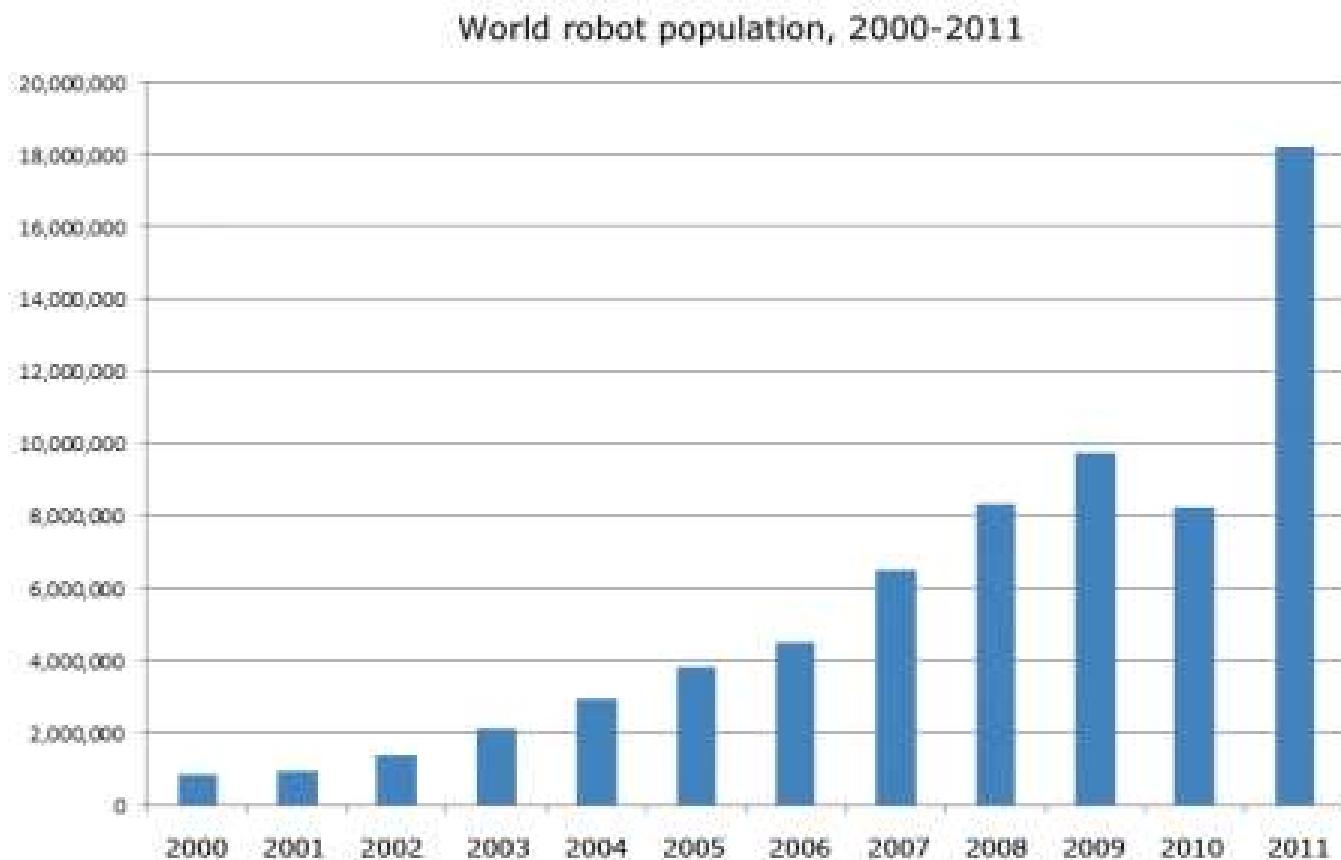
# General intelligence of a frog?

---



# World robot population

The number of robots worldwide continued to grow rapidly and is now on course to reach 100 million by 2020.

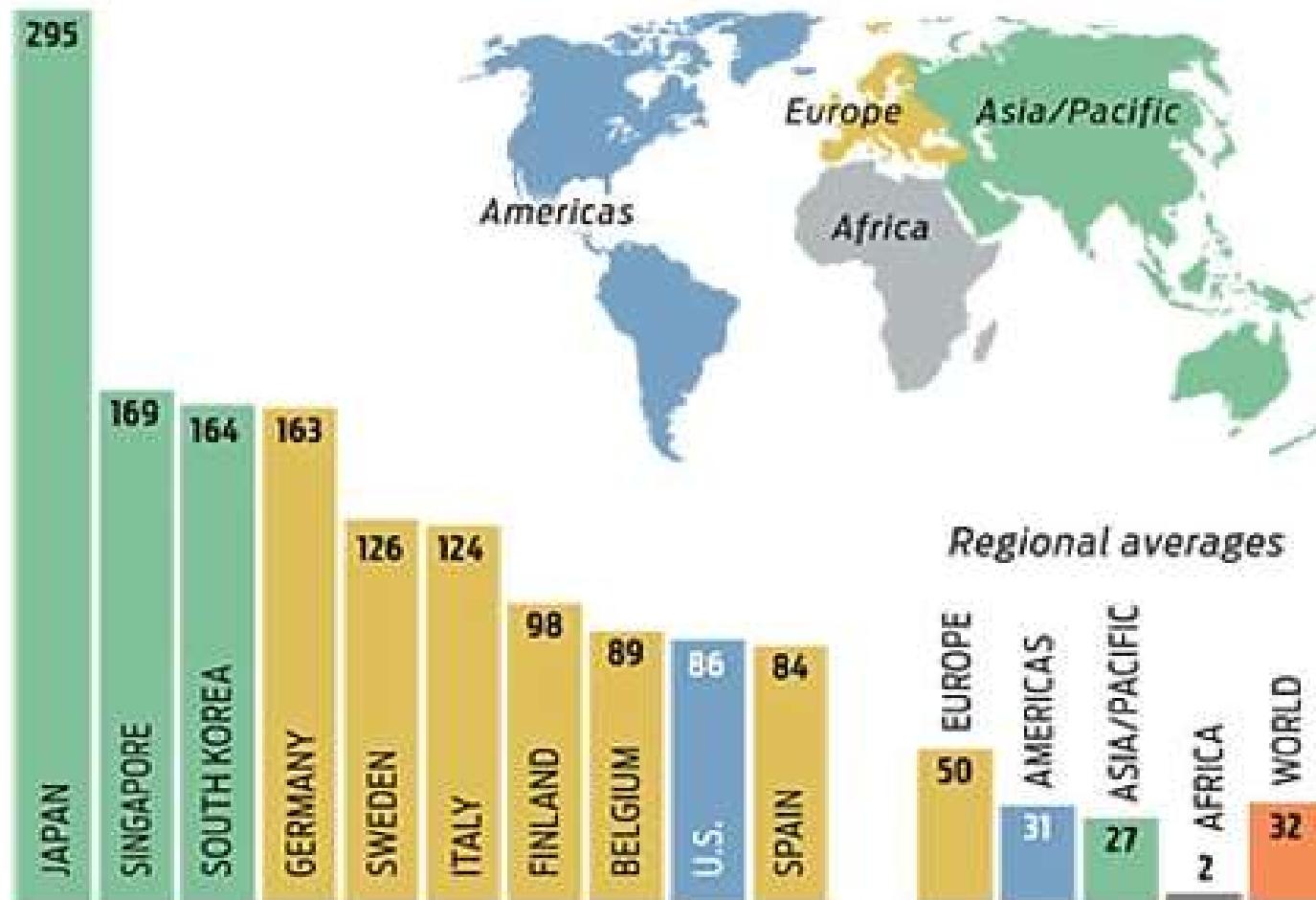


Source: International Federation of Robotics

# World robot population

## TOP 10 COUNTRIES BY ROBOT DENSITY

(Industrial robots per 10 000 manufacturing workers)



# Yêu cầu về khóa học

---

- **Mức độ khoá học:** sinh viên

- **Điều kiện bắt buộc:** có kiến thức về lập trình và hiểu xác suất.

- **Đánh giá:**

- Điểm danh + bài tập ngắn (10 %)
- Thi giữa kỳ / bài tập lớn(30 %)
- Thi cuối kỳ (60 %)

# Nội dung học tập

---

- Giới thiệu về trí tuệ nhân tạo
- Tìm kiếm cơ bản
- Tìm kiếm nâng cao
- Học máy
- Ứng dụng của trí tuệ nhân tạo

# Dự án về trí tuệ nhân tạo

---

- Apple machine learning journal

<https://machinelearning.apple.com/>

- Google Brains (Deep learning)

<https://ai.google/research/teams/brain/>

- Facebook AI research (FAIR)

<https://ai.facebook.com/research/>

- <https://aws.amazon.com/ai/>

# Papers with Code

## State-of-the-Art (SOTA)



### Browse State-of-the-Art

1509 leaderboards • 1327 tasks • 1347 datasets • 17810 papers with code

Follow on Twitter for updates

#### Computer Vision

Semantic Segmentation 33 leaderboards 667 papers with code	Image Classification 52 leaderboards 564 papers with code	Object Detection 54 leaderboards 467 papers with code	Image Generation 51 leaderboards 231 papers with code	Pose Estimation 40 leaderboards 231 papers with code
--	---	---	---	--

See all 707 tasks

#### Natural Language Processing

Machine Translation	Language Modelling	Question Answering	Sentiment Analysis	Text Generation
---------------------	--------------------	--------------------	--------------------	-----------------

<https://paperswithcode.com/sota>

# Python in Google Colab

The screenshot shows a Google Colab notebook titled "python101.ipynb". The notebook interface includes a toolbar with file operations, a sidebar with "CODE" and "TEXT" buttons, and a top bar indicating "CONNECTED" and "EDITING". The code cells contain the following Python code:

```
# Future Value
pv = 100
r = 0.1
n = 7
fv = pv * ((1 + (r)) ** n)
print(round(fv, 2))

[11] amount = 100
interest = 10 #10% = 0.01 * 10
years = 7
future_value = amount * ((1 + (0.01 * interest)) ** years)
print(round(future_value, 2))

[12] # Python Function def
def getfv(pv, r, n):
    fv = pv * ((1 + (r)) ** n)
    return fv
fv = getfv(100, 0.1, 7.)
print(round(fv, 2))

[13] # Python if else
score = 80
if score >=60 :
    print("Pass")
else:
    print("Fail")

[14] Pass
```

# **TRÍ TUỆ NHÂN TẠO**

Chương 2: Không gian trạng thái và tìm  
kiếm mù

# Chương 2: Không gian trạng thái và tìm kiếm mù

## Nội dung

- Giải quyết vấn đề
- Không gian trạng thái
- Tìm kiếm trên không gian trạng thái
- Tìm kiếm theo chiều rộng
- Tìm kiếm theo chiều sâu
- Tìm kiếm chiều sâu có giới hạn
- Bài tập
- Tìm kiếm đều giá

# Giải quyết vấn đề

---

- Phát biểu chính xác bài toán:
  - Hiện trạng ban đầu;
  - Kết quả mong muốn,...
- Phân tích bài toán.
- Thu thập và biểu diễn dữ liệu, tri thức cần thiết để giải bài toán.
- Ra quyết định chọn kỹ thuật/ giải quyết thích hợp.

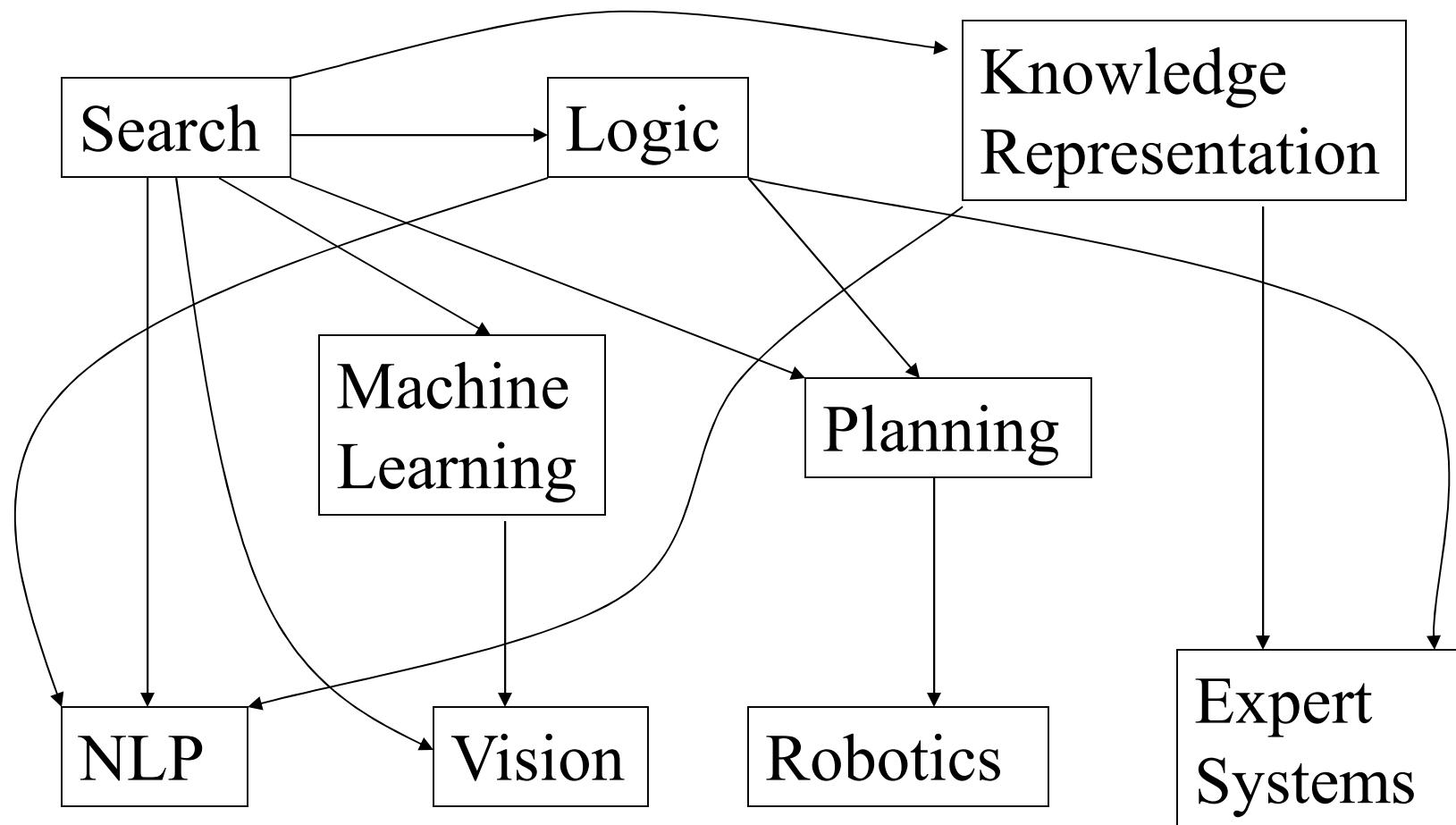
# Tại sao phải tìm kiếm?

---

- Tìm kiếm cái gì?
- Biểu diễn và tìm kiếm là kỹ thuật phổ biến giải các bài toán trong lĩnh vực AI
- Các vấn đề khó khăn trong tìm kiếm với các bài toán AI
  - Đặc tả vấn đề phức tạp
  - Không gian tìm kiếm lớn
  - Đặc tính đổi trượng tìm kiếm thay đổi
  - Đáp ứng thời gian thực
  - Meta knowledge và kết quả “tối ưu”
- Khó khăn về kỹ thuật

# Lĩnh vực AI và các mối liên quan

---

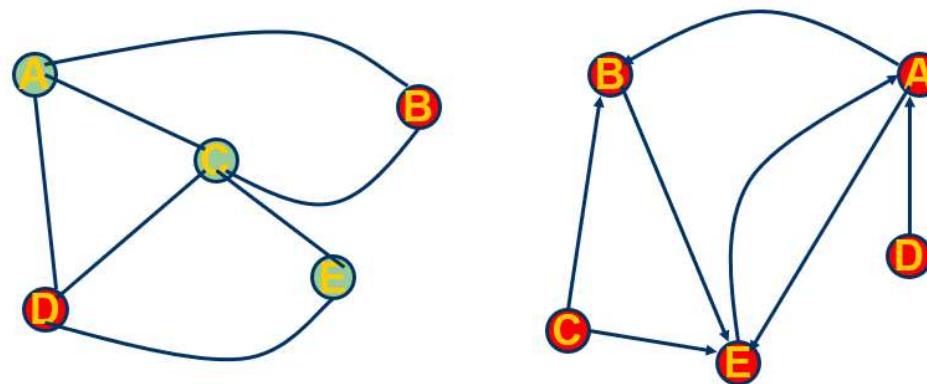


# Lý thuyết đồ thị - Review

Đồ thị: là một cấu trúc bao gồm:

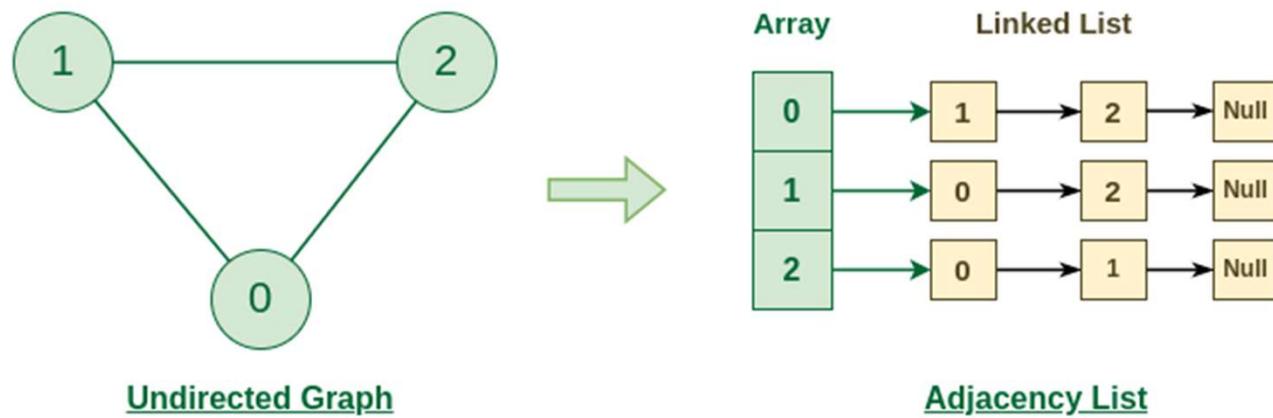
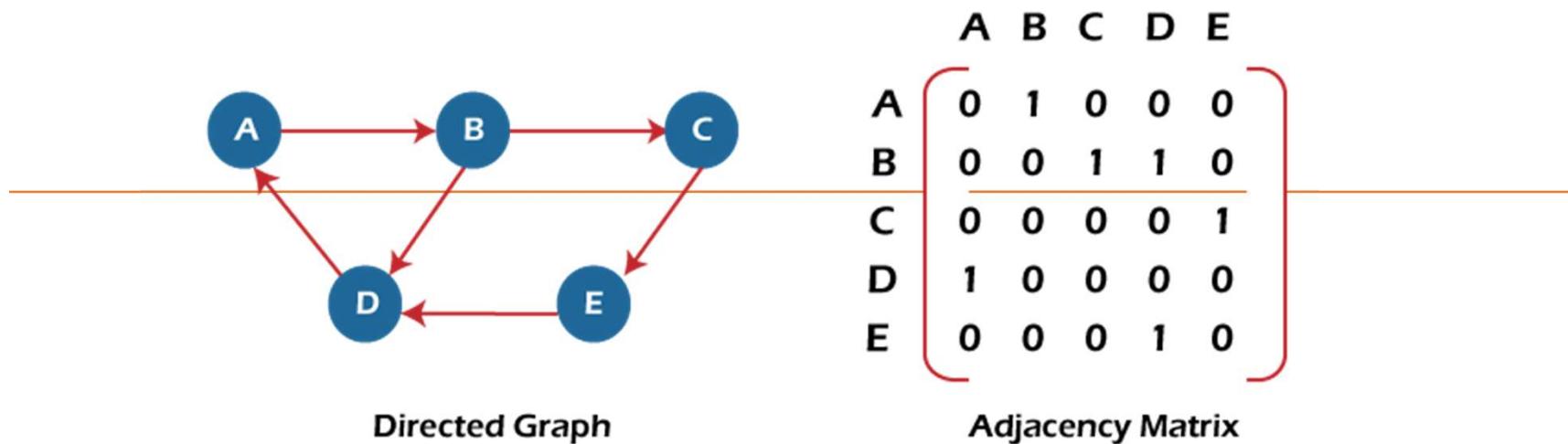
Tập các nút  $N_1, N_2, \dots, N_n, \dots$  Không hạn chế

Tập các cung nối các cặp nút, có thể có nhiều cung trên một cặp nút



Nút: {A,B,C,D,E}

Cung: {(a,d), (a,b), (a,c), (b,c), (c,d), (c,e), (d,e) }



[Graph Representation of Undirected graph to Adjacency List](#)

# Đặc tính đồ thị

---

- Đồ thị có hướng: là đồ thị với các cung có định hướng, nghĩa là cặp nút có quan hệ thứ tự trước sau theo từng cung. Cung  $(Ni, Nj)$  có hướng từ  $Ni$  đến  $Nj$ , Khi đó  $Ni$  là nút cha và  $Nj$  là nút con.
- Nút lá: là nút không có nút con.
- Path: là chuỗi có thứ tự các nút mà 2 nút kế tiếp nhau tồn tại một cung.
- Đồ thị có gốc: Trên đồ thị tồn tại nút X sao cho tất cả các path đều đi qua nút đó. X là gốc - Root
- Vòng : là một path đi qua nút nhiều hơn một lần
- Cây: là graph mà không có path vòng
- Hai nút nối nhau :nếu có một path đi qua 2 nút đó

# Không gian trạng thái

---

- Tập tất cả trạng thái có thể có và tập toán tử của bài toán
- Tồn tại nhiều cách để biểu diễn bài toán
- Đồ thị để biểu diễn bài toán:- đồ thị không gian trạng thái
- Bài toán 7 cây cầu ở Konigsberg của Euler
- Đồ thị không gian trạng thái để phân tích cấu trúc và độ phức tạp của bài toán.

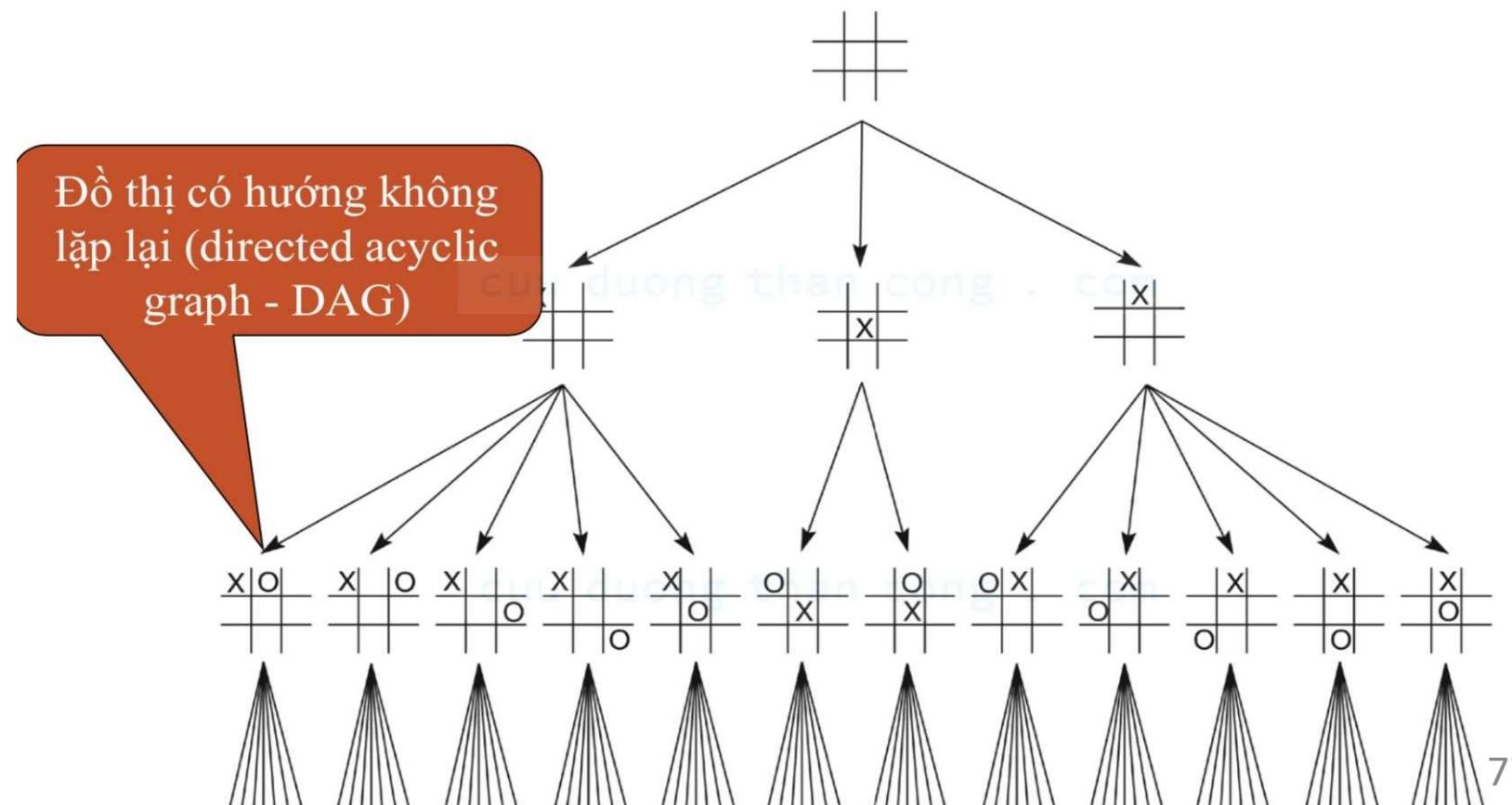
# Không gian trạng thái

---

- Đồ thị không gian trạng thái là một bộ ký hiệu  $[V, E, S, G]$  trong đó:
  - $V$  - tập đỉnh/tập tất cả các trạng thái;
  - $E$  - tập cạnh/ tập cung/ tập toán tử;
  - $S$  - trạng thái đầu;
  - $G$ - trạng thái đích.
- Đặc tính của trạng thái đích  $G$ :
  - Đặc tính có thể đo lường được trong quá trình tìm kiếm.
  - Ví dụ: Tic-tac-toe, chess,...
  - Đặc tính của đường đi được hình thành trong quá trình tìm kiếm.  
Ví dụ : travelling salesman problem, transportation problem,...
- Lời giải (solution path) là một con đường đi từ trạng thái  $S$  đến trạng thái  $G$ .

# Không gian trạng thái

- Ví dụ một phần không gian trạng thái của trò chơi Tic-tac-toe



# Không gian trạng thái

- Ví dụ trò chơi 8 ô chữ

trạng thái đầu

1	2	3
8		4
7	6	5

trạng thái đích

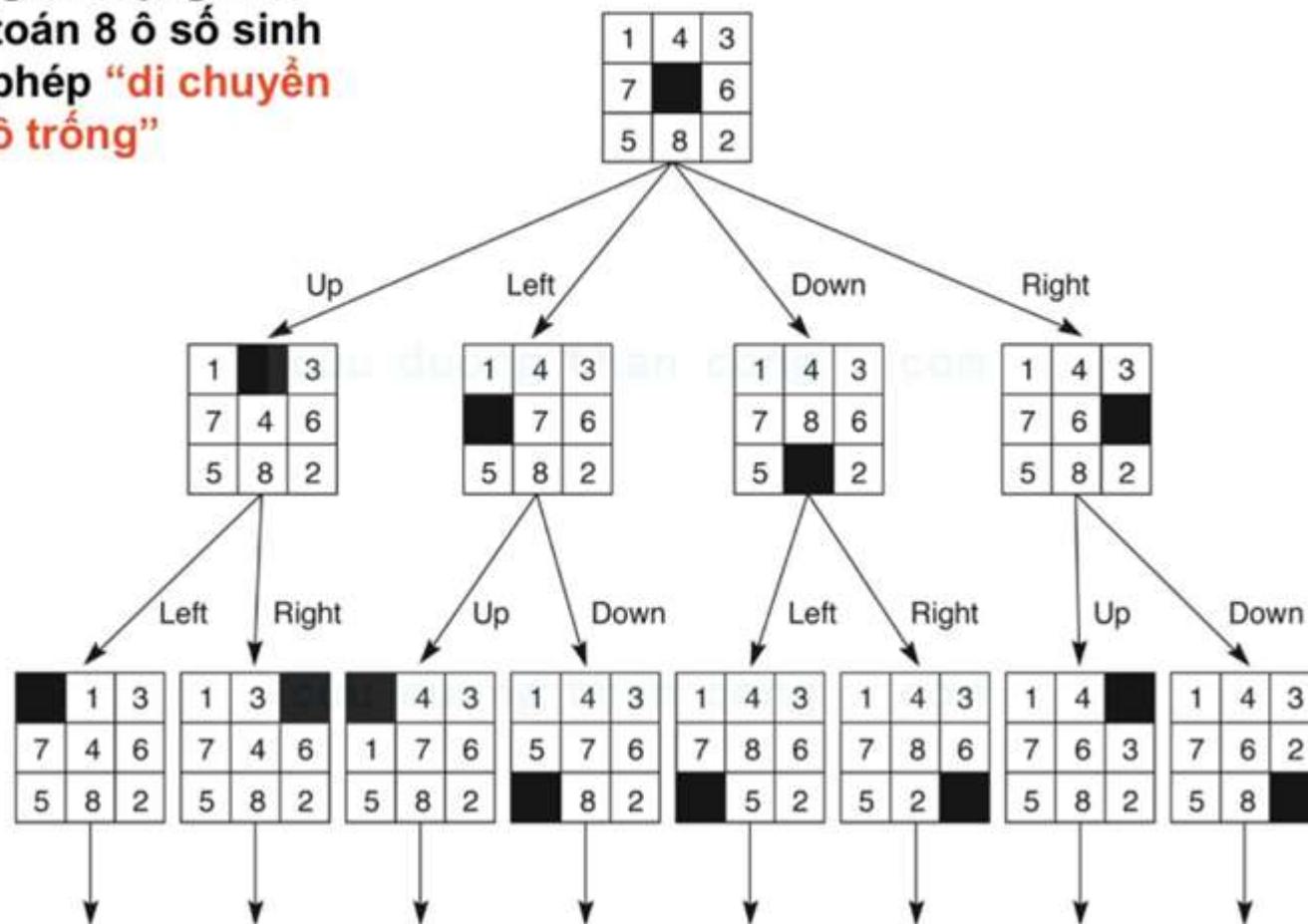
	2	8
3	5	7
6	2	1

- Cách biểu diễn trạng thái của trò chơi này như thế nào ?

# Không gian trạng thái

- 

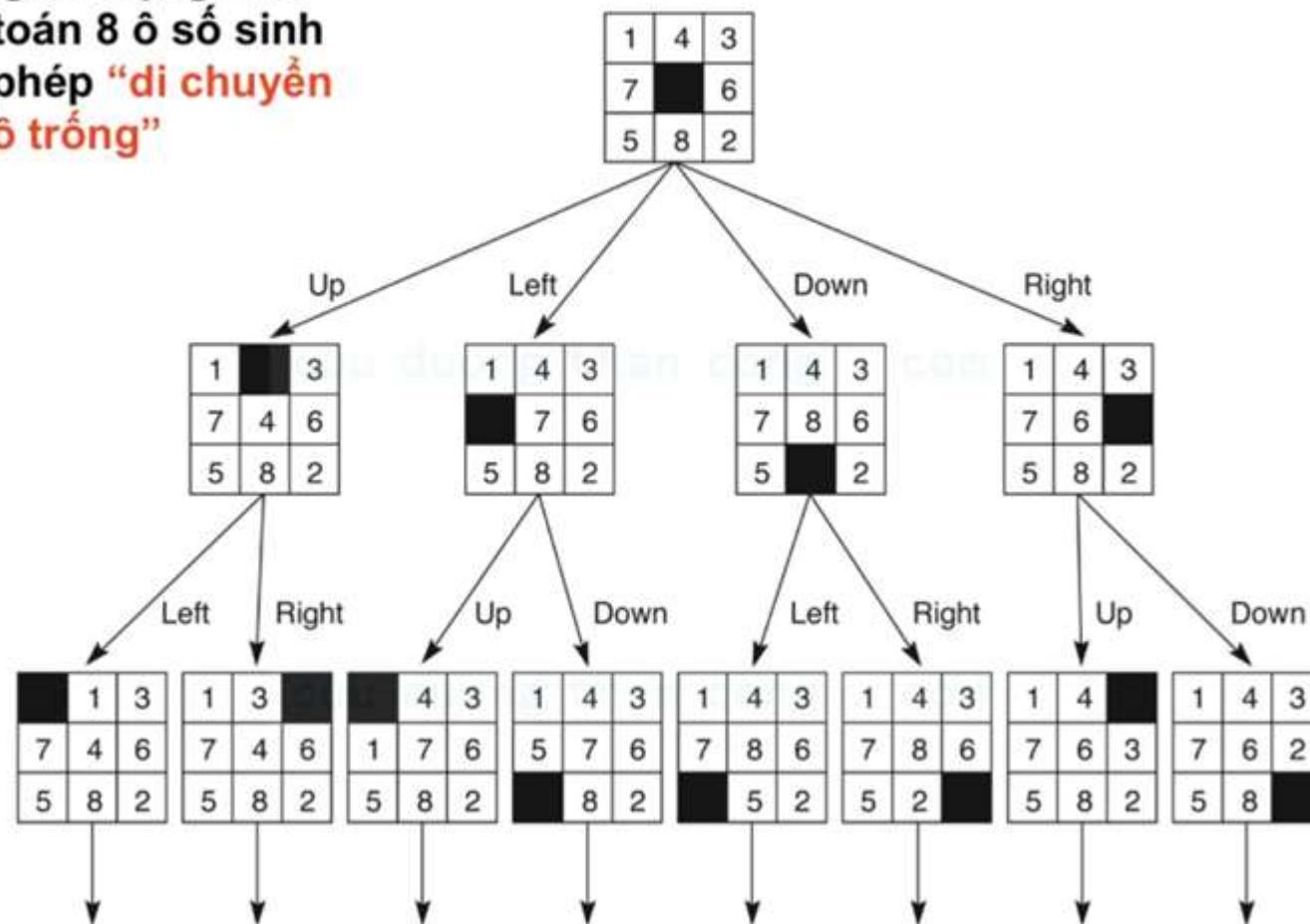
**Không gian trạng thái  
của bài toán 8 ô số sinh  
ra bằng phép “di chuyển  
ô trống”**



# Không gian trạng thái

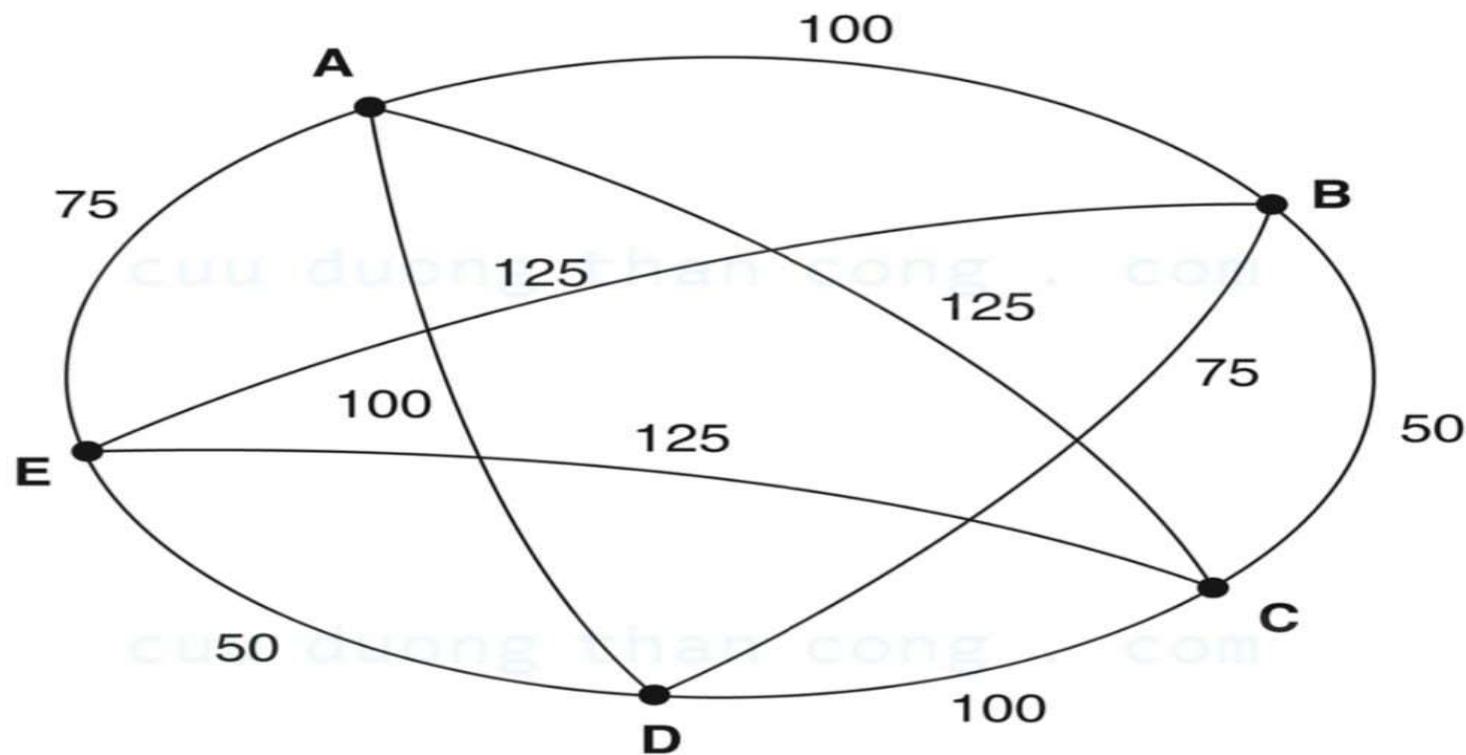
- 

**Không gian trạng thái  
của bài toán 8 ô số sinh  
ra bằng phép “di chuyển  
ô trống”**



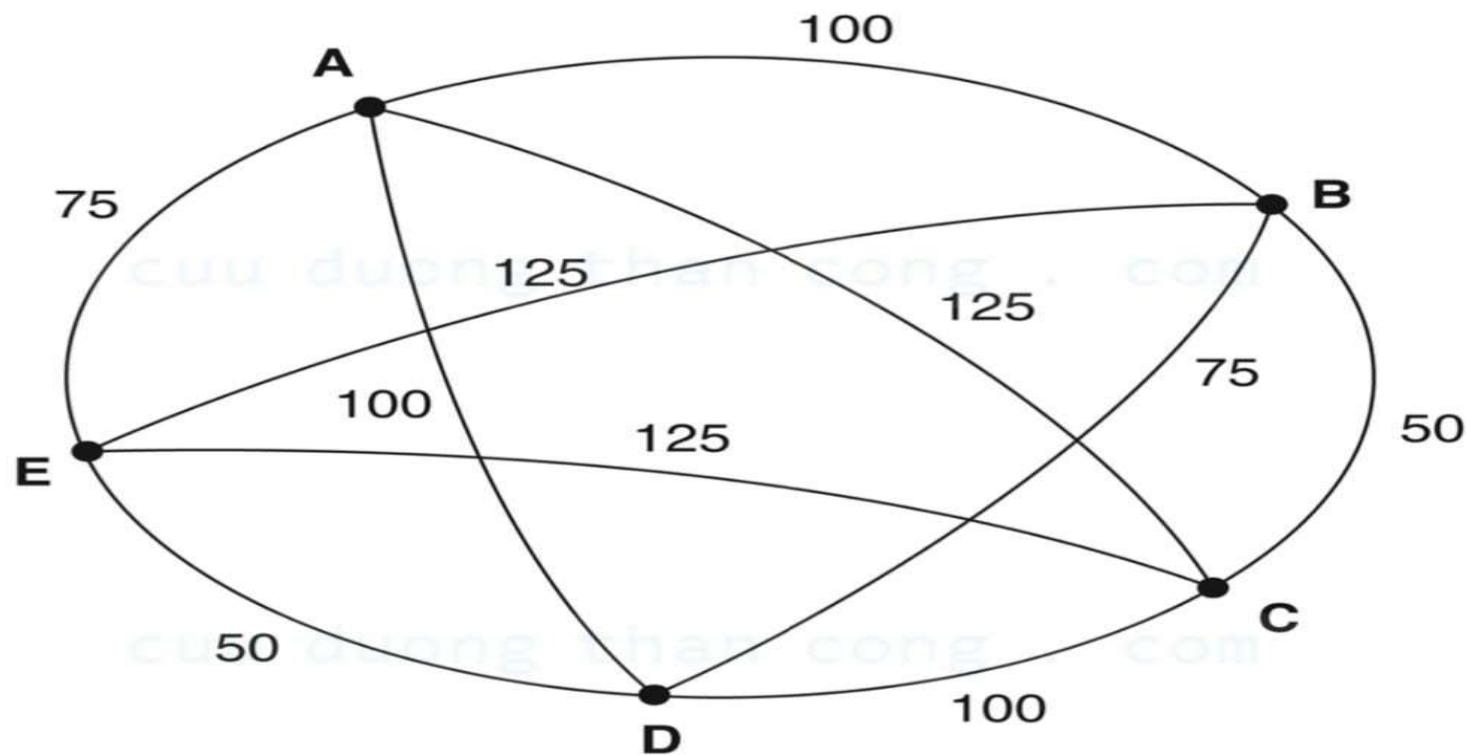
# Không gian trạng thái

- Cần biểu diễn không gian trạng thái cho bài toán người giao hàng như thế nào?

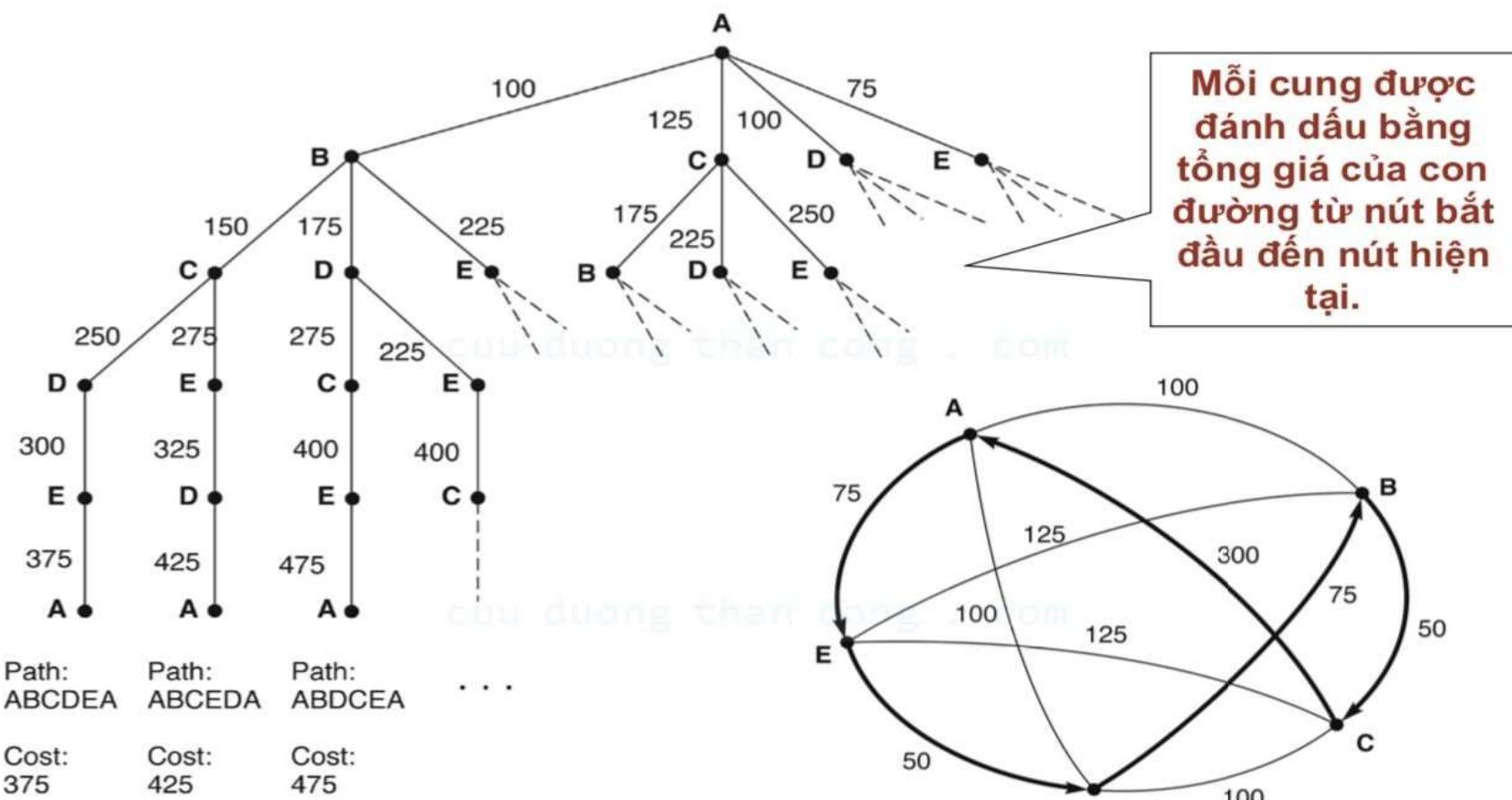


# Không gian trạng thái

- Cần biểu diễn không gian trạng thái cho bài toán người giao hàng như thế nào?



# Không gian trạng thái



# Không gian trạng thái

---

- Yếu tố chính để xác định không gian trạng thái:
- Trạng thái;
- Hành động(toán tử);
- Kiểm tra trạng thái thỏa đích;
- Chi phí mỗi bước chuyển trạng thái.

# Tìm kiếm trên không gian trạng thái

---

- Chiến lược tìm kiếm: lựa chọn thứ tự xét các trạng thái.
- Đánh giá chiến lược :
  - **đủ**: liệu có tìm được lời giải (nếu có).
  - **độ phức tạp thời gian**: số lượng trạng thái phải xét.
  - **độ phức tạp lưu trữ**: tổng dung lượng bộ nhớ phải lưu trữ (các trạng thái trong quá trình tìm kiếm).
  - **tối ưu**: có lời giải tối ưu.
- Độ phức tạp thời gian và lưu trữ có thể được đo bằng:
  - **b**: độ phân nhánh của cấu trúc cây.
  - **d**: độ sâu của lời giải ngắn nhất.
  - **m**: độ sâu tối đa của không gian trạng thái (có thể vô hạn).

# Tìm kiếm trên không gian trạng thái

---

- **Chiến lược tìm kiếm mù**
  - Tìm kiếm theo chiều rộng.
  - Tìm kiếm đều giá (uniform-cost search).
  - **Tìm kiếm theo chiều sâu.**
  - Tìm kiếm theo chiều sâu có giới hạn.

# Tìm kiếm lối ra mê cung?

---



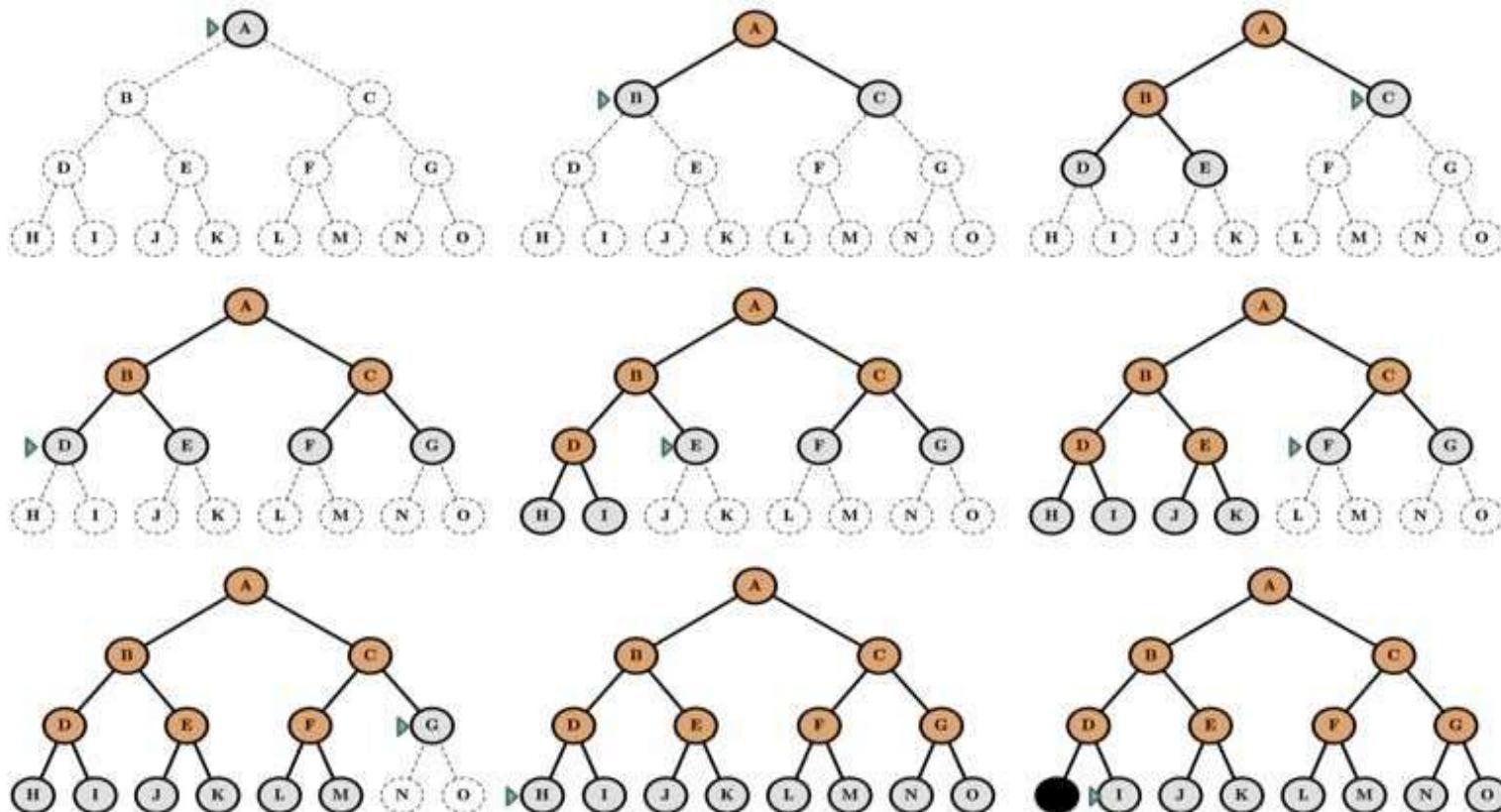
# Tìm kiếm theo chiều rộng

---

- Tìm kiếm theo từng tầng ( từ trên xuống)
- Phát triển trạng thái gần với trạng thái hiện tại.
- Tập trạng thái được chia thành 3 khu vực:
  - Khu vực đã phát triển (explored): tập trạng thái đã xét
  - Khu vực chờ phát triển (frontier): tập trạng thái đang chờ xét
  - Khu vực chưa phát triển (unexplored): tập trạng thái chưa xét

# Tìm kiếm theo chiều rộng

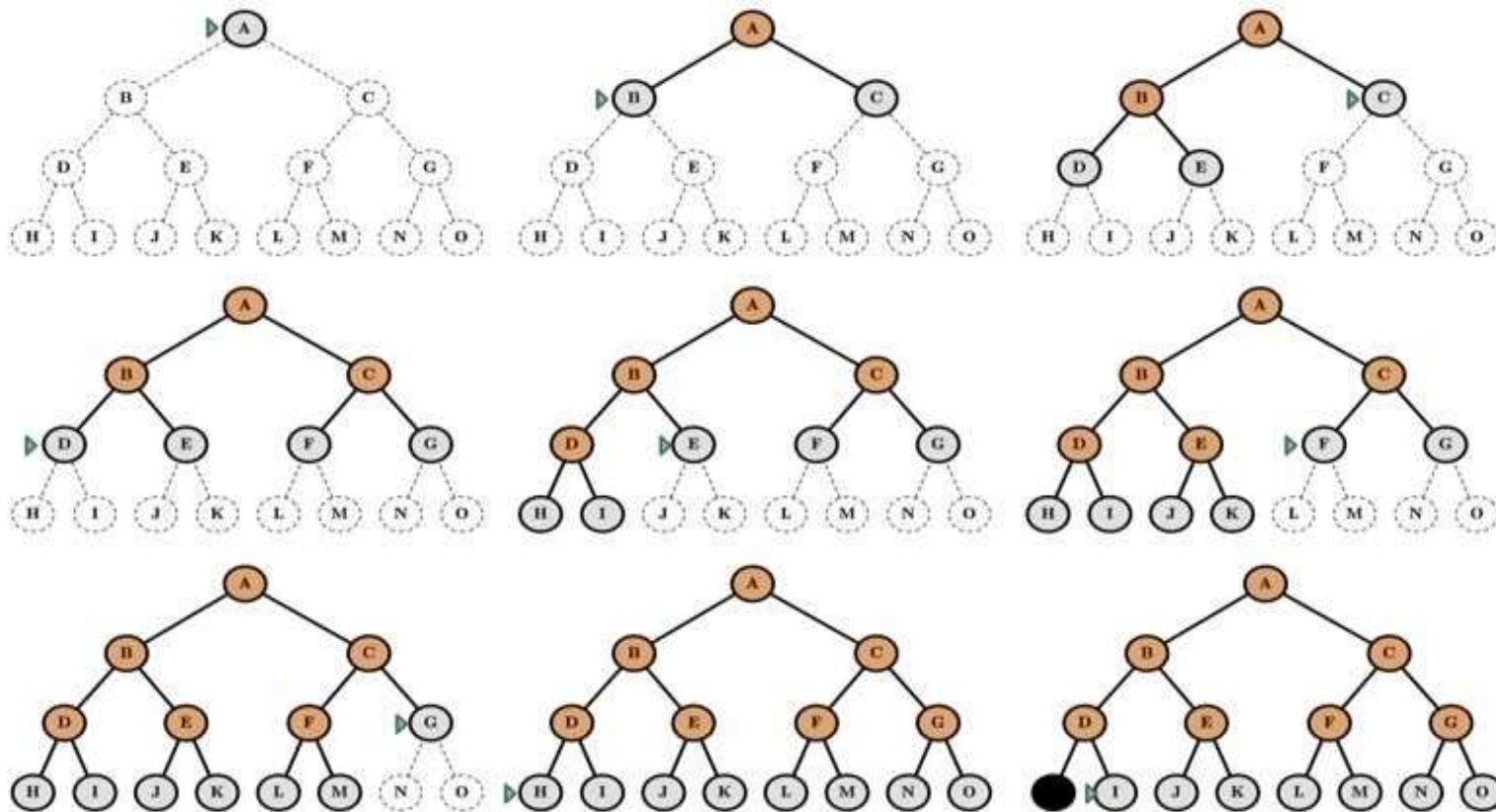
- Minh họa sự phát triển các đỉnh theo thuật toán tìm kiếm theo chiều rộng



- A->B->C->D->E->F->G->H->I->J->K->L->M->N->O

# Tìm kiếm theo chiều rộng

- Minh họa sự phát triển các đỉnh theo thuật toán tìm kiếm theo chiều rộng



- A->B->C->D->E->F->G->H->I->J->K->L->M->N->O

# Tìm kiếm theo chiều rộng

---

- Thuật toán tìm kiếm theo chiều rộng được mô tả bởi thủ tục:

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier  $\cup$  explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

# Tìm kiếm theo chiều rộng

- Bài tập[5 phút]. Viết danh sách các biến đổi trạng thái trong hàng đợi **frontier** ?

frontier = {A}

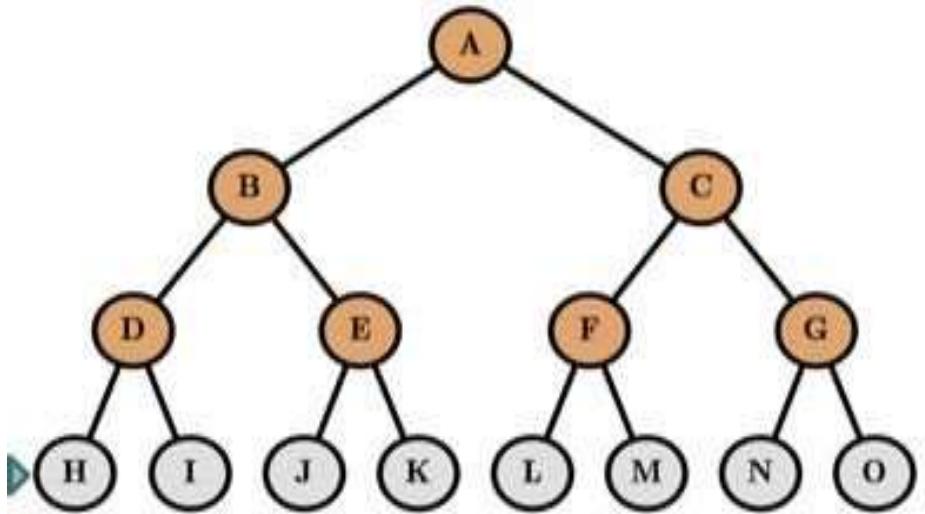
frontier = {B, C}

frontier = {C, D, E}

frontier = {D, E, F, G}

frontier = {E, F, G, H,

I} frontier = ...



```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

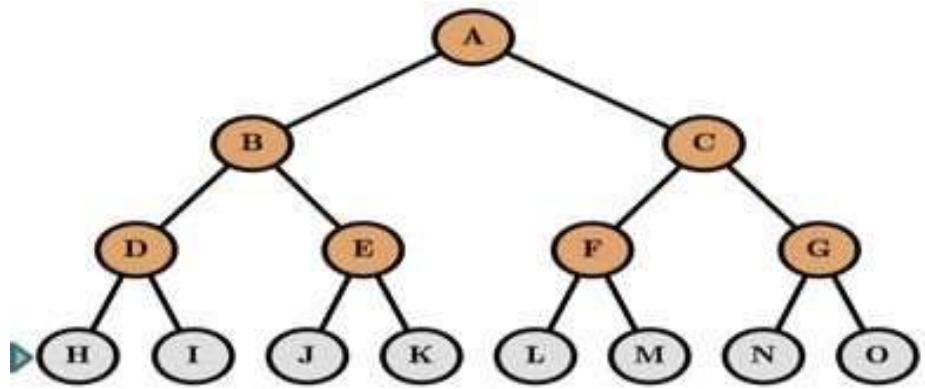
        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

# Demo 1(10 phút)

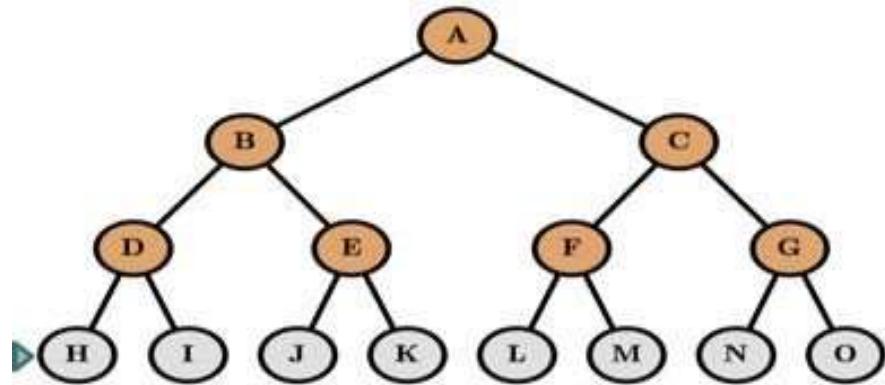
```
13 if __name__ == '__main__':
14     graph = {
15         'A' : ['B','C'],
16         'B' : ['D','E'],
17         'C' : ['F','G'],
18         'D' : ['H','I'],
19         'E' : ['J','K'],
20         'F' : ['L','M'],
21         'G' : ['N','O'],
22         'H' : [],
23         'I' : [],
24         'J' : [],
25         'K' : [],
26         'L' : [],
27         'M' : [],
28         'N' : [],
29         'O' : []
30     }
31     result = BFS('A', 'O')
32     if result:
33         s = 'explored: '
34         for i in result:
35             s += i + ' '
36             print(s)
37     else:
38         print("404 Not Found!")
```



```
1 def BFS(initialState, goal):
2     frontier = [initialState]
3     explored = []
4     while frontier:
5         state = frontier.pop(0)
6         explored.append(state)
7         if goal == state:
8             return explored
9         for neighbor in graph[state]:
10            if neighbor not in (explored and frontier):
11                frontier.append(neighbor)
12    return False
```

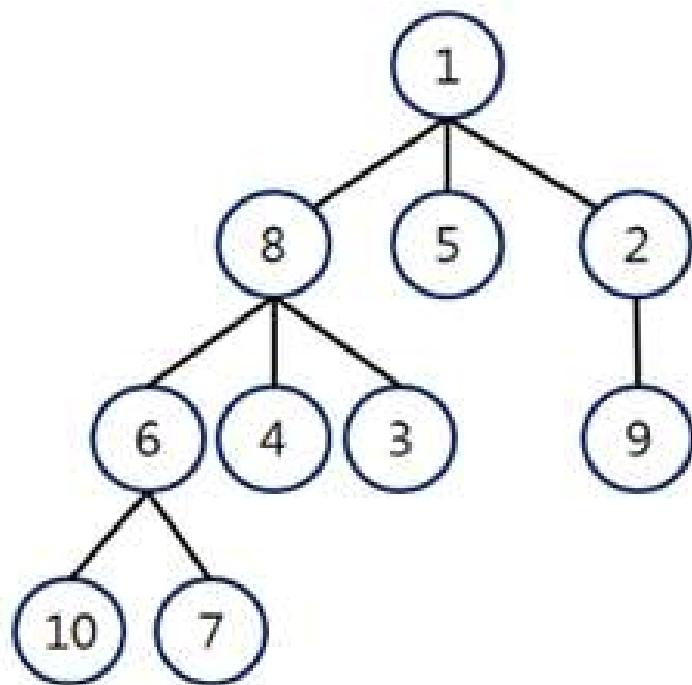
# Demo 2(10 phút)

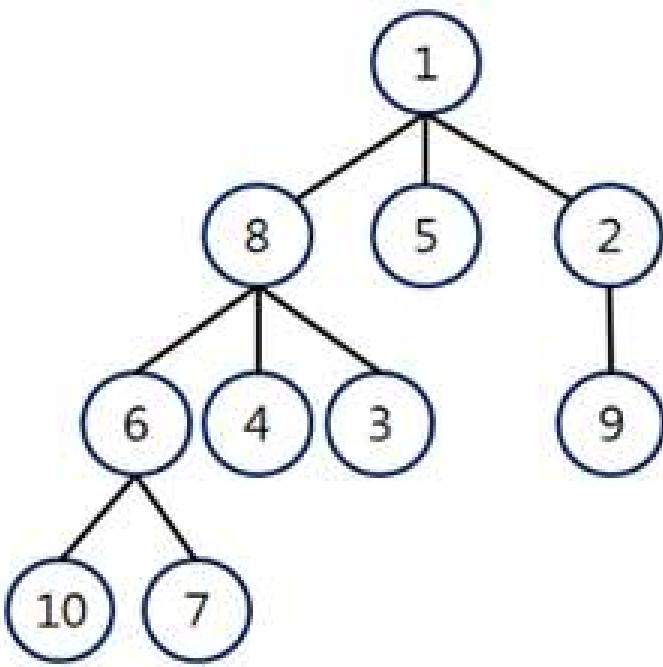
```
14 if __name__ == '__main__':
15     tree = Tree()
16     tree.create_node('A', 'A') #root
17     tree.create_node('B', 'B', 'A')
18     tree.create_node('C', 'C', 'A')
19     tree.create_node('D', 'D', 'B')
20     tree.create_node('E', 'E', 'B')
21     tree.create_node('F', 'F', 'C')
22     tree.create_node('G', 'G', 'C')
23     tree.create_node('H', 'H', 'D')
24     tree.create_node('I', 'I', 'D')
25     tree.create_node('J', 'J', 'E')
26     tree.create_node('K', 'K', 'E')
27     tree.create_node('L', 'L', 'F')
28     tree.create_node('M', 'M', 'F')
29     tree.create_node('N', 'N', 'G')
30     tree.create_node('O', 'O', 'G')
31     result = BFS(tree.get_node("A"), 'O')
32
33     if result:
34         s = 'explored: '
35         for i in result:
36             s += i.tag + ' '
37             print(s)
38     else:
39         print("404 Not Found!")
```



```
1 from treelib import Tree, Node
2 def BFS(initialState: Node, goal):
3     frontier = [initialState]
4     explored = []
5     while frontier:
6         state = frontier.pop(0)
7         explored.append(state)
8         if goal == state.tag:
9             return explored
10        for neighbor in tree.children(state.identifier):
11            if neighbor not in (explored and frontier):
12                frontier.append(neighbor)
13    return False
```

**Cho biết kết quả duyệt đồ thị theo thuật toán BFS**





BFS: 1 8 5 2 6 4 3 9 10 7

# Tìm kiếm theo chiều rộng

---

- Trạng thái nào phát sinh trước sẽ được duyệt trước, nên frontier phải cấu trúc dữ liệu FIFO.
- Đánh giá thuật toán theo chiều rộng
  - **Tính đủ ?**
    - có ( nếu b hữu hạn)
  - **Độ phức tạp thời gian ?:**
    - $1 + 1.b + b.b + \dots + b^{(d-1)}.b = O(b^d)$
  - **Độ phức tạp về lưu trữ ?:**
    - $O(b^d)$
  - **Tính tối ưu ?:**
    - có(nếu chi phí cho mỗi bước chuyển là 1 đơn vị ).
- Độ phức tạp về thời gian và không gian theo cấp số nhanh.

# Tìm kiếm theo chiều rộng

---

- Tìm kiếm theo chiều rộng tệ đến mức nào ?

# Tìm kiếm theo chiều rộng

- Thực nghiệm

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

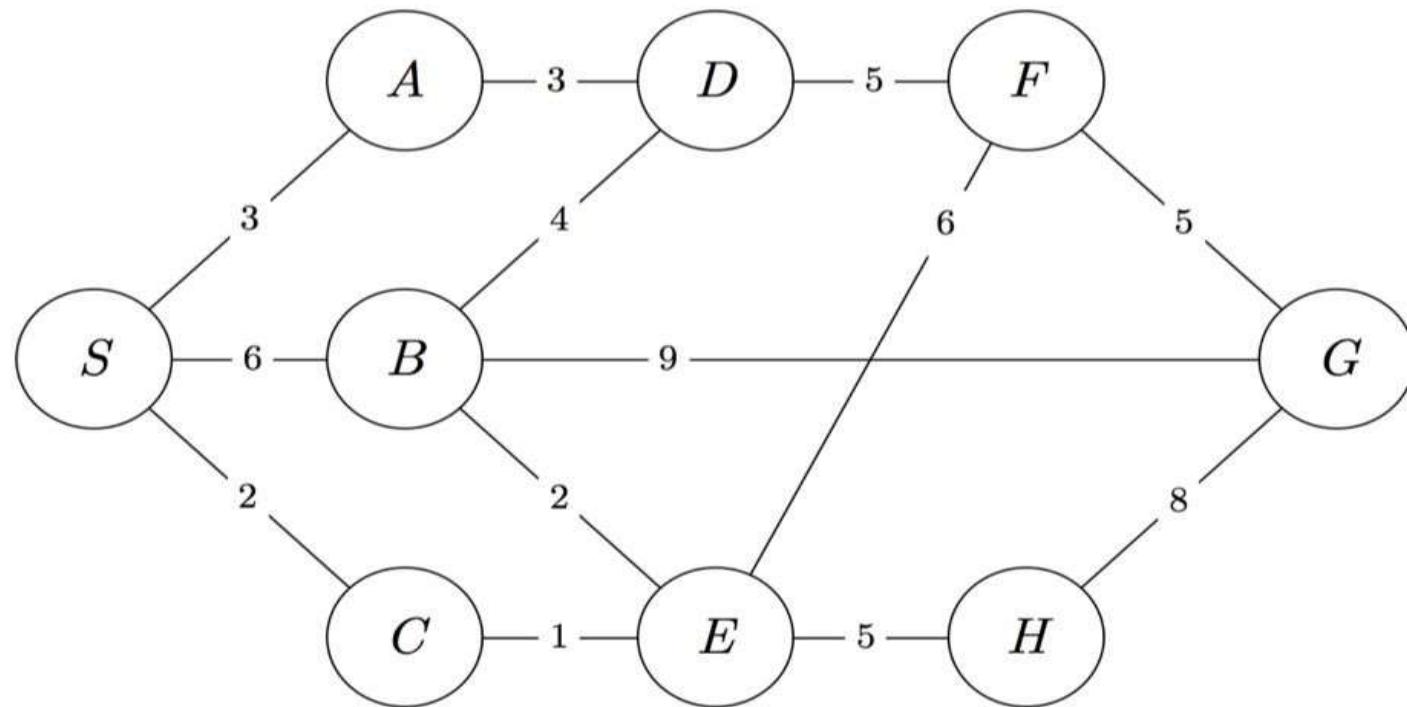
$b = 10$ , 1 giây kiểm tra được 1 triệu nodes, lưu 1 node mất 1.000byte

- $b = 10$ , 1 giây kiểm tra được 1 triệu nodes, lưu 1 node mất 1.000byte
- Độ phức tạp thời gian và không gian(bộ nhớ) là bất lợi của thuật toán**

## Bài tập

---

- Hãy xây dựng thứ tự khám phá và đường đi của các đỉnh sử dụng tìm kiêm theo chiêu rộng.



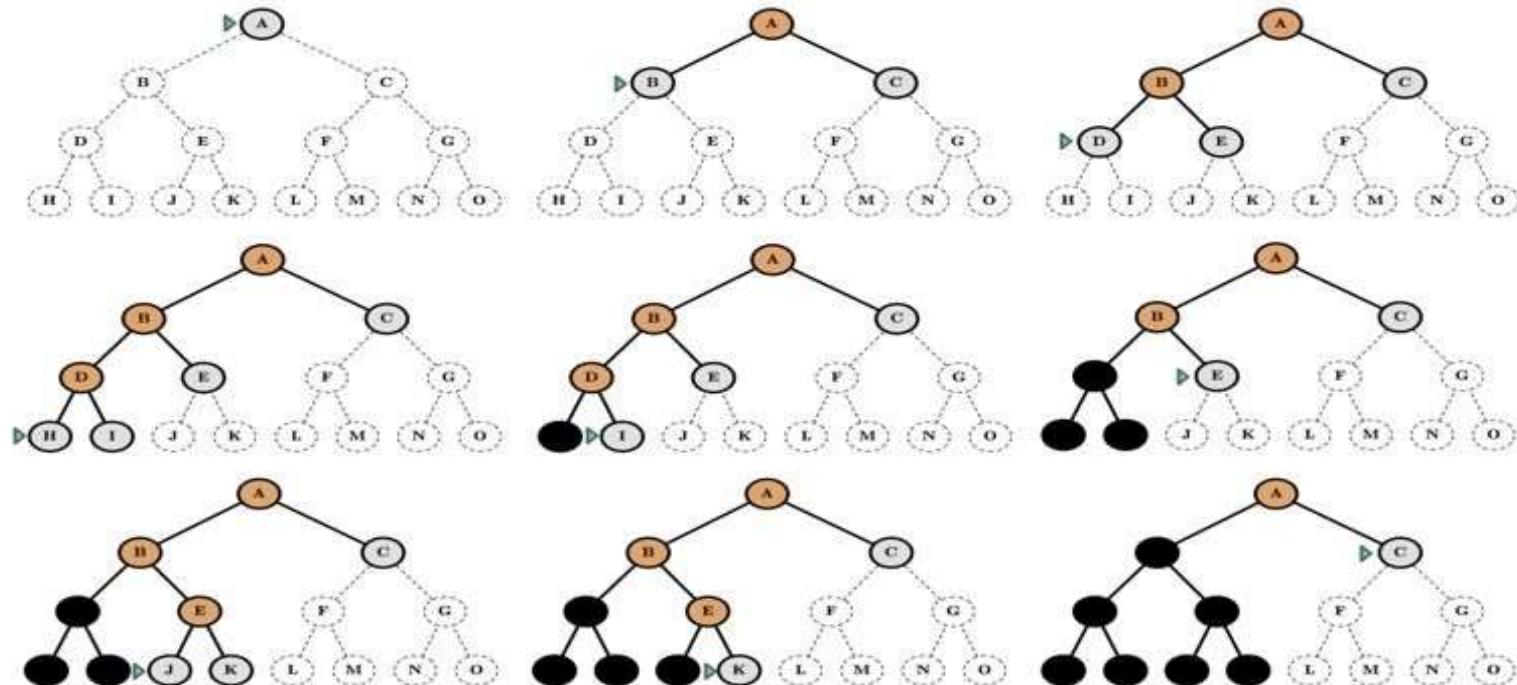
# Tìm kiếm theo chiều sâu

---

- Phát triển các trạng thái ở sâu so với trạng thái hiện tại
- Tập trạng thái được chia thành 3 khu vực:
  - Khu vực đã phát triển (explored): trạng thái đã xét;
  - Khu vực chờ phát triển (frontier): trạng thái đang chờ xét;
  - Khu vực chưa phát triển (unexplored): trạng thái chưa xét.

# Tìm kiếm theo chiều sâu

- Minh họa việc phát triển các đỉnh theo thuật toán tìm kiếm theo chiều sâu



# Tìm kiếm theo chiều sâu

---

- Thuật toán tìm kiếm theo chiều sâu được mô tả bởi thủ tục

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

# Tìm kiếm theo chiều sâu

- Hãy làm rõ quá trình biến đổi của danh sách **frontier** ?

Frontier = {A}

Frontier = {B,C,}

Frontier = {B, F,G}

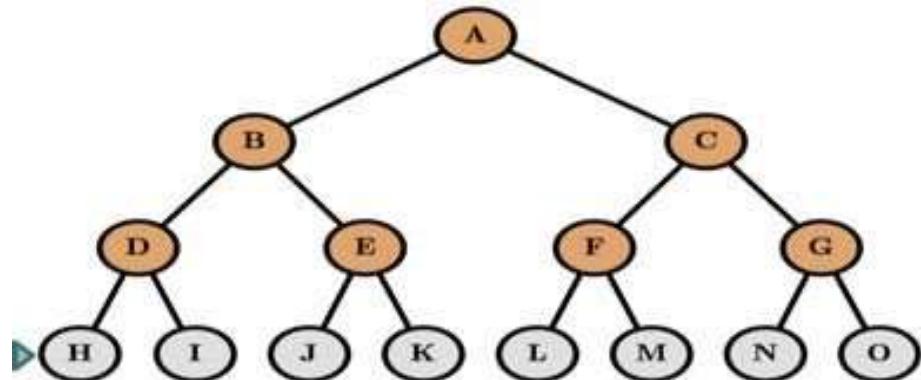
Frontier = {B, F, N,O}

Frontier = {B, F, N }

Frontier = {B, F}

Frontier = {B,L, M}

Frontier = {....}



```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :
        frontier = Stack.new(initialState)
        explored = Set.new()

        while not frontier.isEmpty():
            state = frontier.pop()
            explored.add(state)

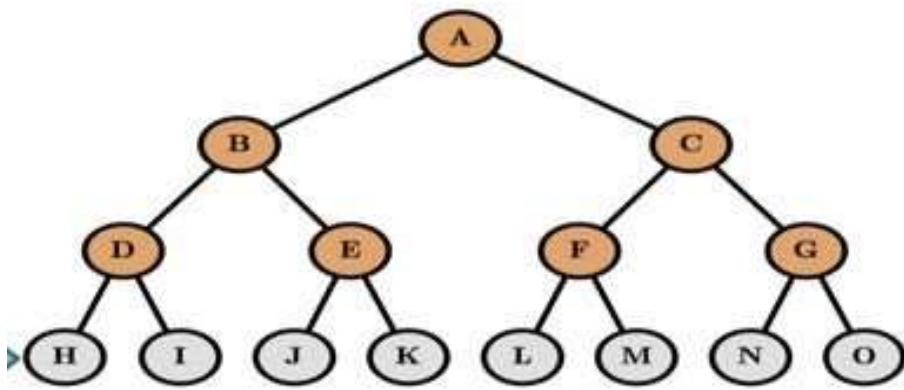
            if goalTest(state):
                return SUCCESS(state)

            for neighbor in state.neighbors():
                if neighbor not in frontier ∪ explored:
                    frontier.push(neighbor)

        return FAILURE
```

# Demo 3(10 phút)

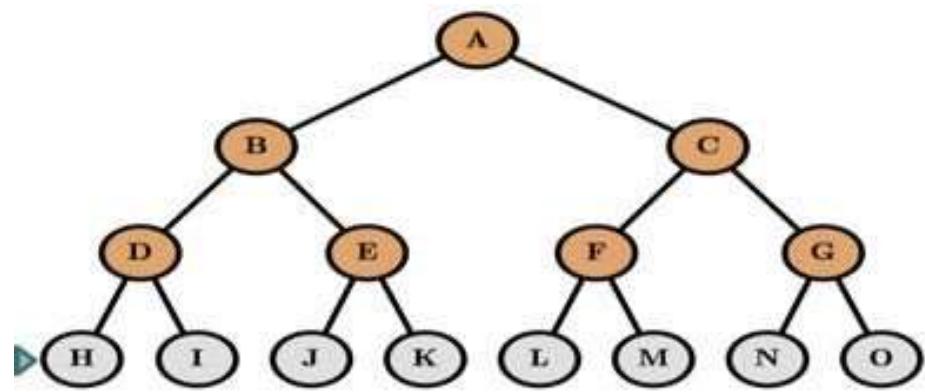
```
20 if __name__ == '__main__':
21     nodeA = Node("A")
22     nodeB = Node("B")
23     nodeC = Node("C")
24     nodeD = Node("D")
25     nodeE = Node("E")
26     nodeF = Node("F")
27     nodeG = Node("G")
28     nodeH = Node("H")
29     nodeI = Node("I")
30     nodeJ = Node("J")
31     nodeK = Node("K")
32     nodeL = Node("L")
33     nodeM = Node("M")
34     nodeN = Node("N")
35     nodeO = Node("O")
36     nodeA.addChild([nodeB, nodeC])
37     nodeB.addChild([nodeD, nodeE])
38     nodeC.addChild([nodeF, nodeG])
39     nodeD.addChild([nodeH, nodeI])
40     nodeE.addChild([nodeJ, nodeK])
41     nodeF.addChild([nodeL, nodeM])
42     nodeG.addChild([nodeN, nodeO])
43     result = DFS(nodeA, 'H')
44     if result:
45         s = 'explored: '
46         for i in result:
47             s += i.name + " "
48             print(s);
49     else:
50         print('404 Not Found!')
```



```
1 class Node:
2     def __init__(self, name):
3         self.name = name
4         self.children = []
5     def addChild(self, list):
6         for c in list:
7             self.children.append(c)
8     def DFS(initialState, goal):
9         frontier = [initialState]
10        explored = []
11        while frontier:
12            state = frontier.pop()
13            explored.append(state)
14            if goal == state.name:
15                return explored
16            for child in state.children:
17                if child not in (explored and frontier):
18                    frontier.append(child)
19        return False
```

# Demo 4(5 phút)

```
13 if __name__ == '__main__':
14     graph = {
15         'A' : ['B','C'],
16         'B' : ['D','E'],
17         'C' : ['F','G'],
18         'D' : ['H','I'],
19         'E' : ['J','K'],
20         'F' : ['L','M'],
21         'G' : ['N','O'],
22         'H' : [],
23         'I' : [],
24         'J' : [],
25         'K' : [],
26         'L' : [],
27         'M' : [],
28         'N' : [],
29         'O' : []
30     }
31     result = DFS('A', 'H')
32     if result:
33         s = 'explored: '
34         for i in result:
35             s += i + ' '
36             print(s)
37     else:
38         print("404 Not Found!")
```



```
1 def DFS(initialState, goal):
2     frontier = [initialState]
3     explored = []
4     while frontier:
5         state = frontier.pop(len(frontier)-1)
6         explored.append(state)
7         if goal == state:
8             return explored
9         for neighbor in graph[state]:
10            if neighbor not in (explored and frontier):
11                frontier.append(neighbor)
12    return False
```

# Tìm kiếm theo chiều sâu

---

- Nhận xét về thuật toán
  - Đối với BFS luôn tìm được nghiệm (nếu bài toán có nghiệm)
  - Đối với DFS chỉ tìm được nghiệm (nếu không gian hữu hạn), vì nếu không gian tìm kiếm vô hạn, thì thuật toán tìm kiếm theo chiều sâu chọn nhánh để đi, nếu đi đúng nhánh vô hạn mà nghiệm không nằm trên nhánh đó.

# Tìm kiếm theo chiều sâu

---

- Đánh giá thuật toán theo chiều rộng
  - **Tính đủ ?**
    - không (nếu không gian tìm kiếm vô hạn hoặc lặp)
    - đủ (nếu khử lặp và không gian tìm kiếm hữu hạn)
  - **Độ phức tạp thời gian ?:**
    - $O(b^m)$
    - Nếu hàm đánh giá tốt thì, thời gian thực tế giảm đáng kể
  - **Độ phức tạp không gian gian ?:**
    - $O(b \cdot m)$ : độ phức tạp tuyến tính
  - **Tính tối ưu ?:**
    - không

# Tìm kiếm theo chiều sâu

- Quay trở lại với bảng đánh giá tìm kiếm theo chiều rộng

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

- Với  $d = 16$ , có thể giảm dung lượng bộ nhớ từ 10 exabyte trên BFS xuống bao nhiêu ..... trên tìm kiếm theo chiều sâu(DFS) ?

# Tìm kiếm theo chiều sâu

- Quay trở lại với bảng đánh giá tìm kiếm theo chiều rộng

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

- Với  $d = 16$ , có thể giảm dung lượng bộ nhớ từ 10 exabytes trên BFS xuống 156 kilobytes DFS

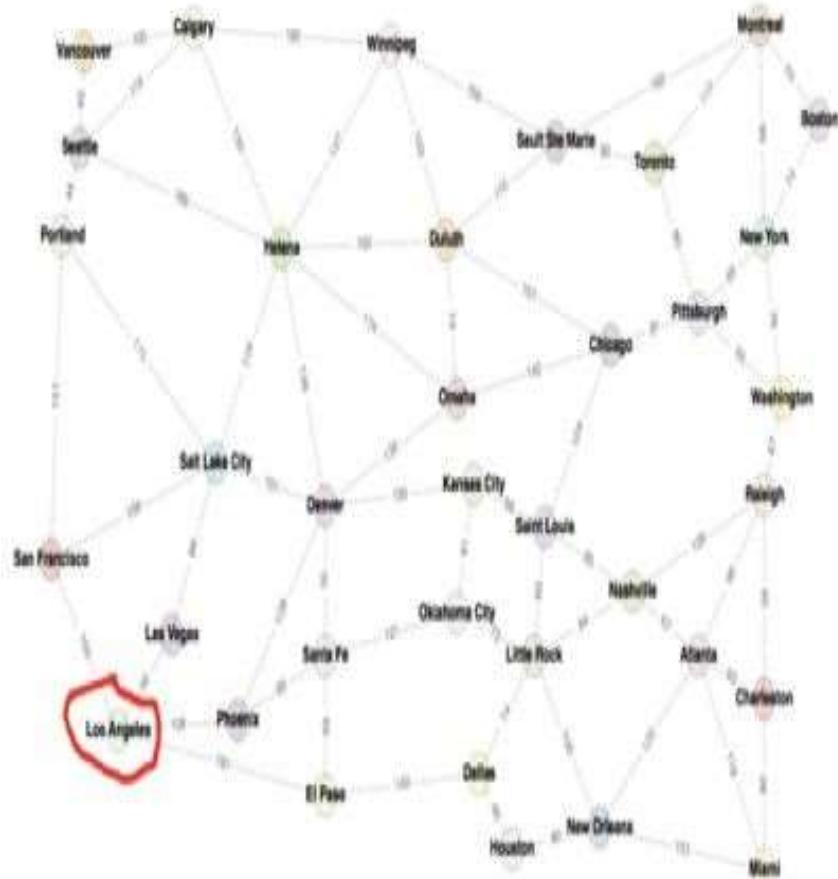
# Tìm kiếm chiều sâu có giới hạn

---



# Tìm kiếm theo chiều sâu có giới hạn

- Giả thiết nếu biết trước bài toán thì không duyệt hết độ sâu trong thuật toán tìm kiếm theo chiều sâu.
  - Chọn độ sâu  $L$ , để phát triển thuật toán tìm kiếm theo chiều sâu (các đỉnh ở độ sâu  $L$ , không chứa đỉnh con, không chứa đỉnh cháu)
  - Ý tưởng: Qua thực nghiệm chỉ ra rằng không có một thành phố nào tới thành phố khác với độ sâu vượt quá 36, nên  $L = 36$



# Tìm kiếm theo chiều sâu có giới hạn

---

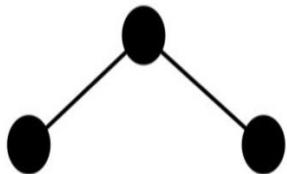
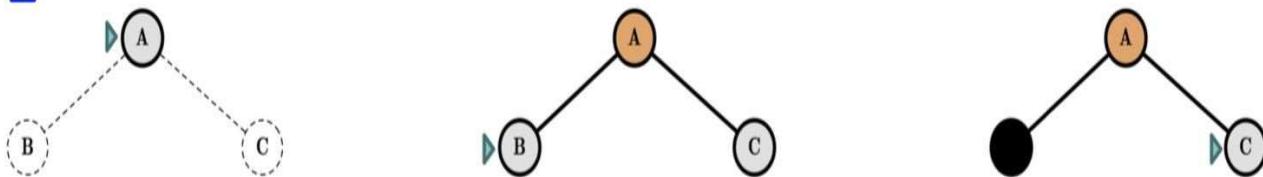
**Limit = 0**



# Tìm kiếm theo chiều sâu có giới hạn

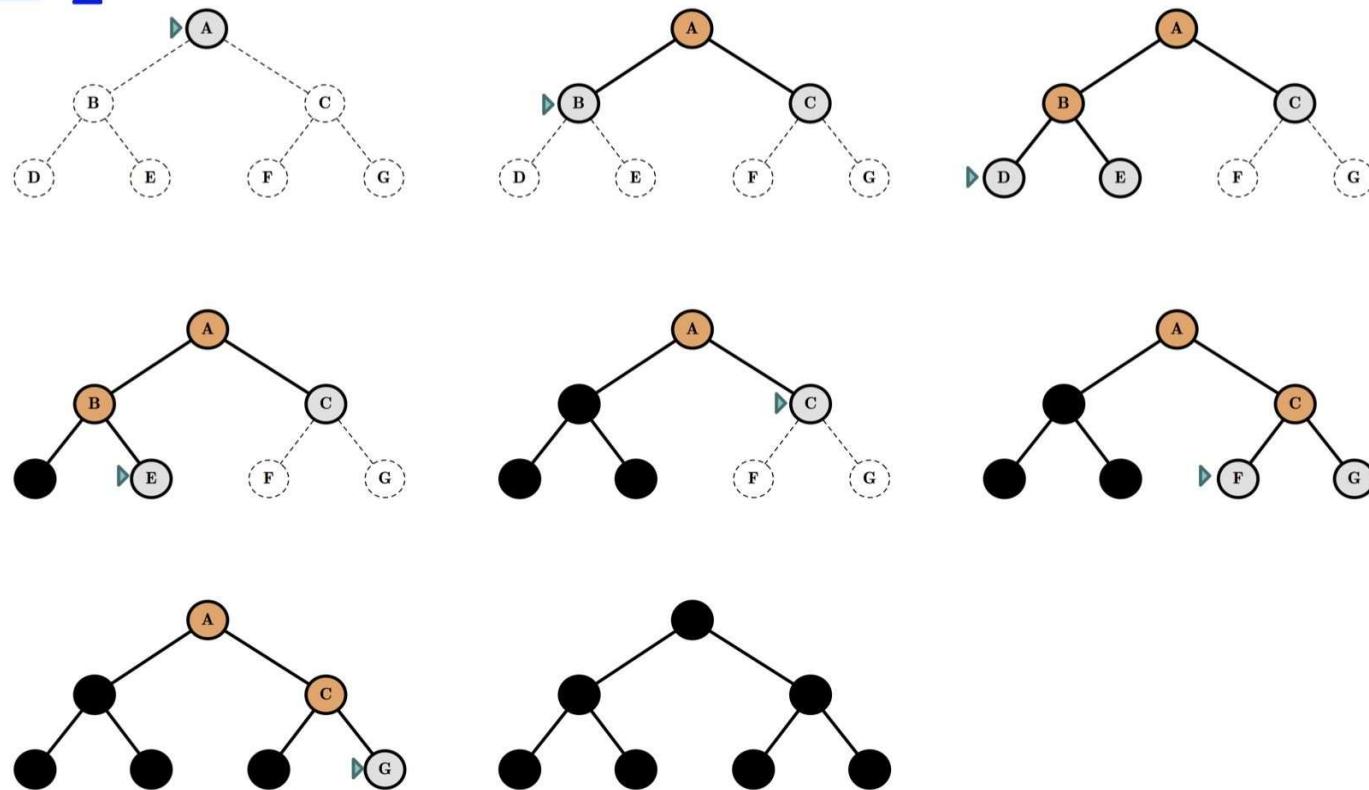
---

**Limit = 1**



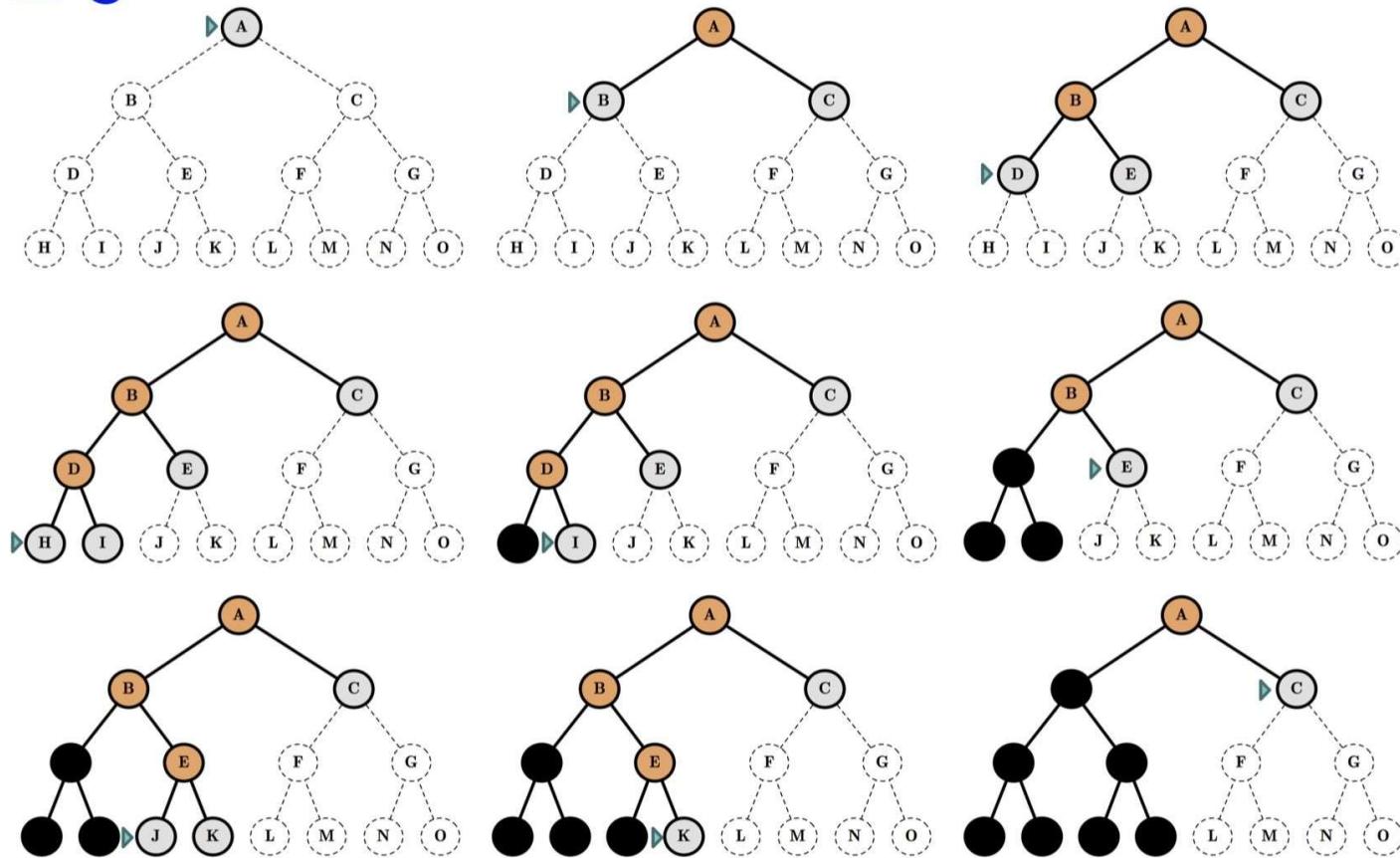
# Tìm kiếm theo chiều sâu có giới hạn

**Limit = 2**



# Tìm kiếm theo chiều sâu có giới hạn

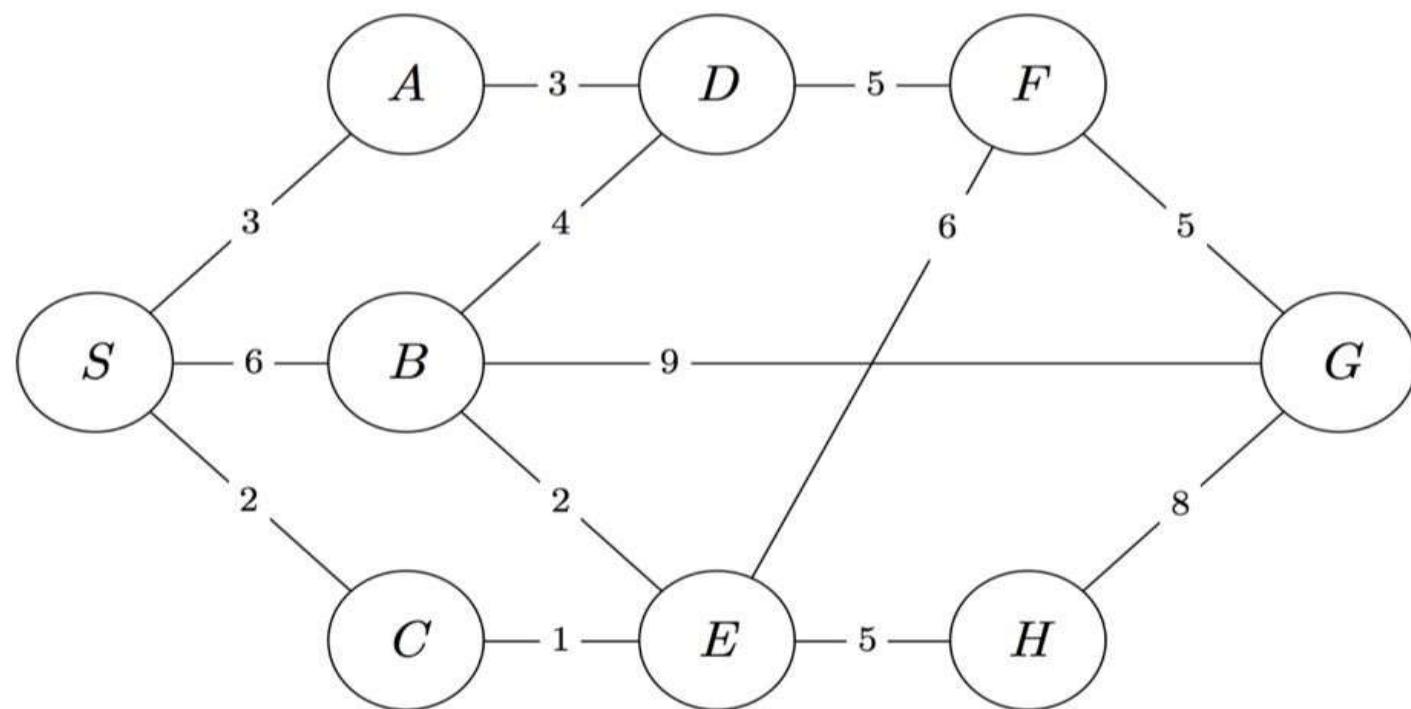
**Limit = 3**



# Bài tập 1

---

- Hãy xây dựng thứ tự khám phá và đường đi của các đỉnh sử ký tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu và tìm kiếm đều giá.



## Bài tập 2

---

- Hãy viết chương trình với các thuật toán tìm kiếm theo chiều rộng, thuật toán tìm kiếm theo chiều sâu.

---

## Tìm kiếm đều giá (Uniform-cost search)



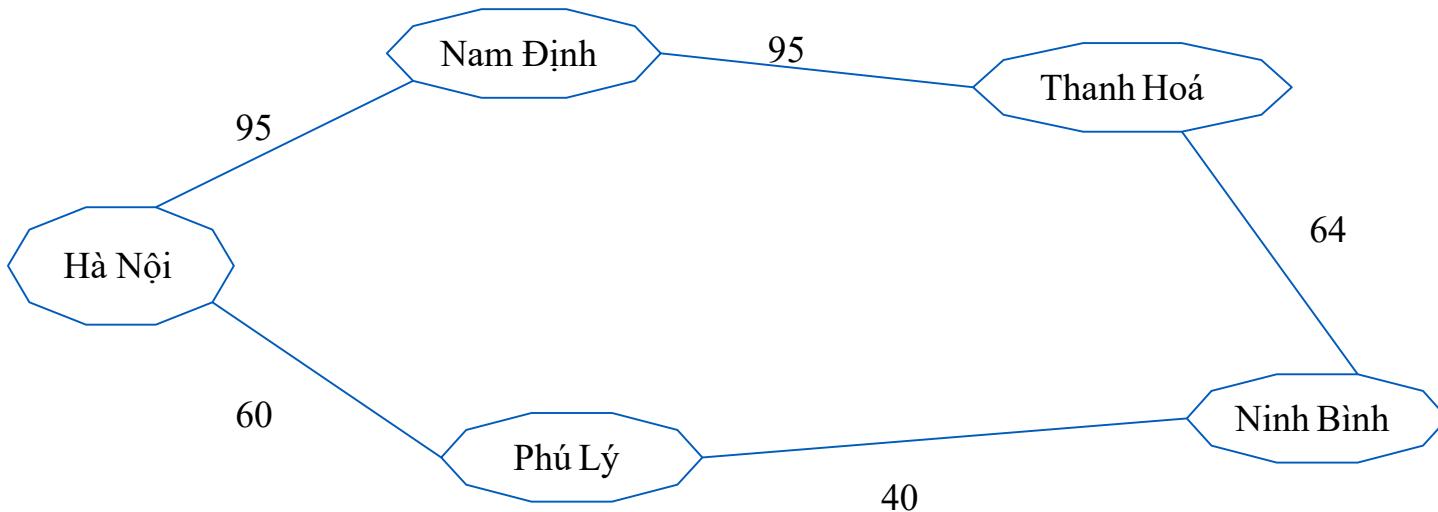
# Tìm kiếm đều giá (Uniform-cost search)

---

- Tương như tìm kiếm theo chiều rộng nếu các đỉnh có “giá” như nhau
- Ý tưởng: xét đỉnh có “giá” nhỏ nhất trước
- Cài đặt: sử dụng hàng đợi có sự ưu tiên về “giá”
- Cải tiến BFS: ưu tiên chi phí, không ưu tiên độ sâu.
- Mỗi đỉnh n viếng thăm có hàm chi phí bé nhất  $g(n)$

# Tìm kiếm đều giá (Uniform-cost search)

- Mỗi đỉnh n viếng thăm có hàm chi phí bé nhất  $g(n)$



- Từ Hà Nội đến Thanh Hoá. Sử dụng BFS, thì cho đường đi Hà Nội - Nam Định - Thanh Hoá, nhưng sử dụng UCS, thì cho đường đi Hà Nội - Phú Lý - Ninh Bình - Thanh Hoá với khoảng cách ngắn hơn.

# Tìm kiếm đều giá (Uniform-cost search)

---

- Mỗi đỉnh n viếng thăm có hàm chi phí bé nhất  $g(n)$
- Thuật toán tìm kiếm đều giá được mô tả bởi thủ tục:

```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

Cập nhật “giá”

# Tìm kiếm đều giá

- Hãy làm rõ quá trình biến đổi của danh sách **frontier** ?

frontier = {A(0)}

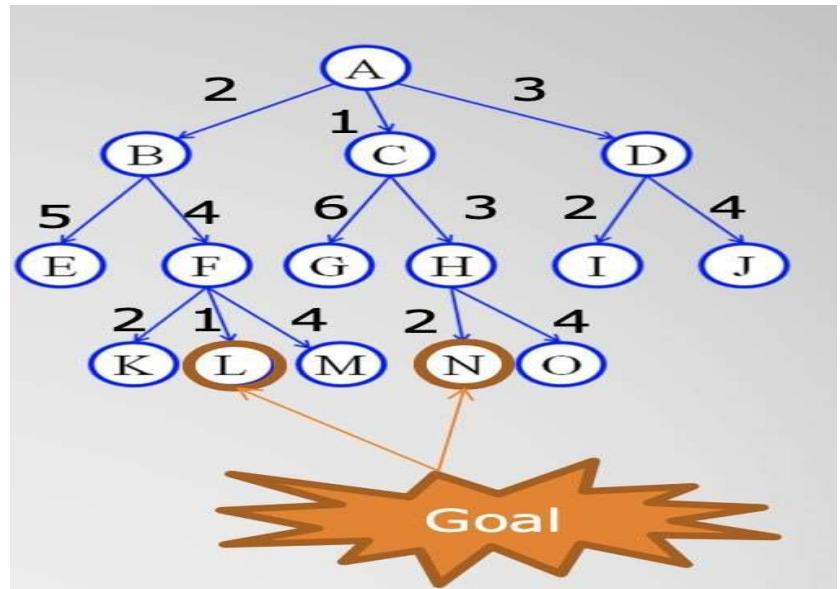
frontier = {B(2), C(1),D(3)}

frontier = {B(2), G(1+6),H(1+3),D(3)}

frontier =

{E(2+5),F(2+4),G(7),H(4),D(3)}

frontier = ...



```
function UNIFORM-COST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n)$  */
    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

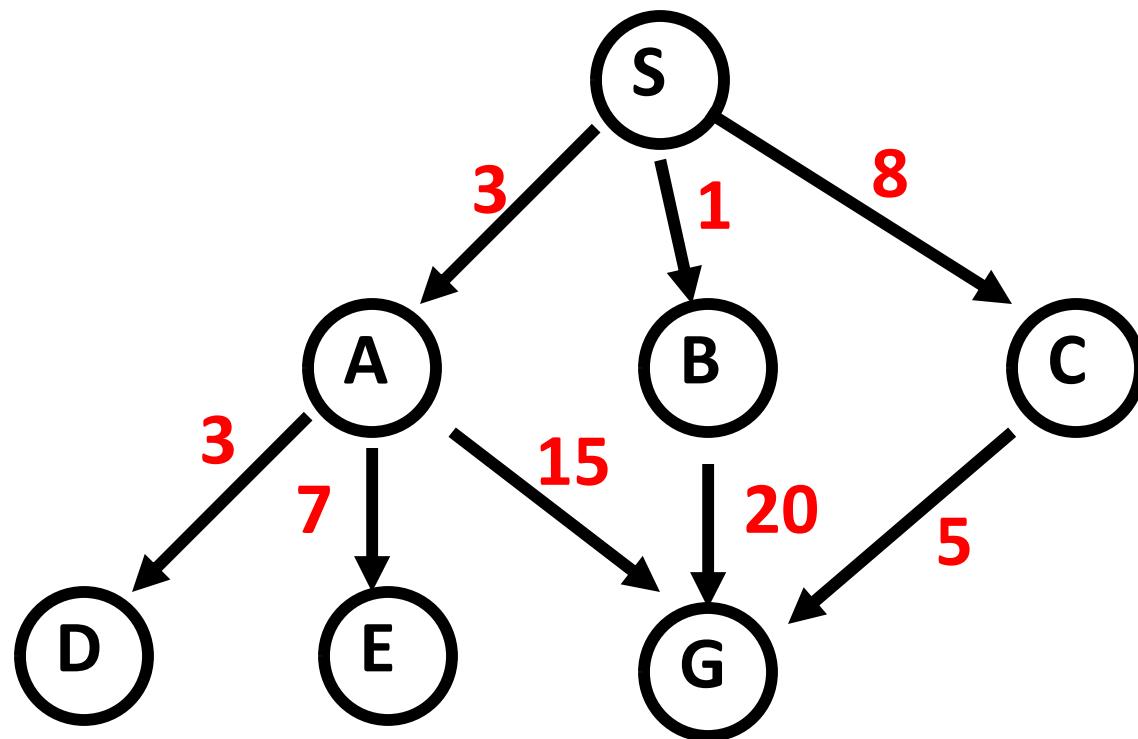
        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

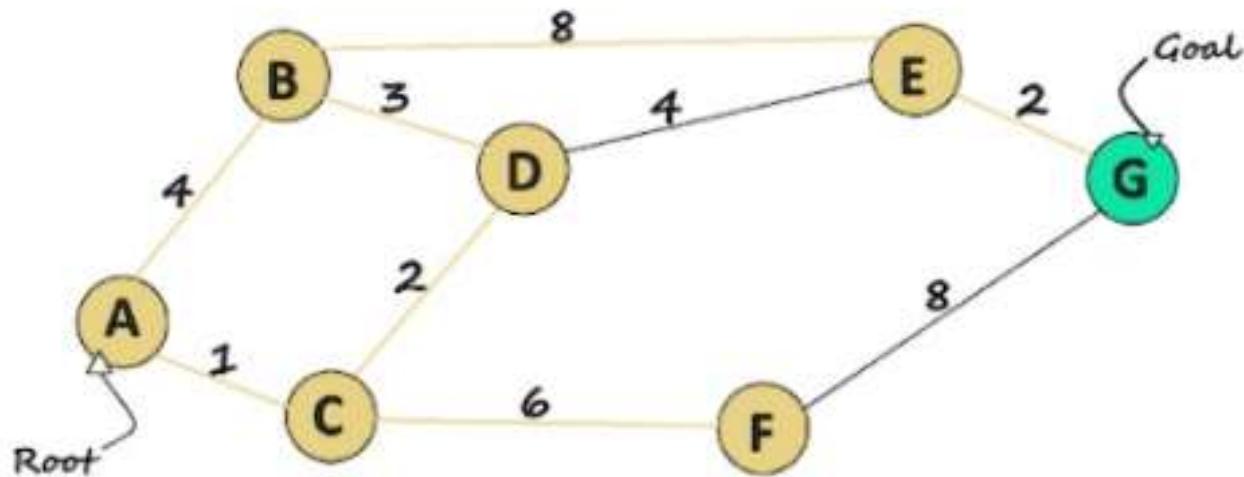
# Uninformed Search

---



# Uninformed Search

---



# Breadth-First Search

Expanded node	Nodes list
	$\{ S^0 \}$
$S^0$	$\{ A^3 \ B^1 \ C^8 \}$
$A^3$	$\{ B^1 \ C^8 \ D^6 \ E^{10} \ G^{18} \}$
$B^1$	$\{ C^8 \ D^6 \ E^{10} \ G^{18} \ G^{21} \}$
$C^8$	$\{ D^6 \ E^{10} \ G^{18} \ G^{21} \ G^{13} \}$
$D^6$	$\{ E^{10} \ G^{18} \ G^{21} \ G^{13} \}$
$E^{10}$	$\{ G^{18} \ G^{21} \ G^{13} \}$
$G^{18}$	$\{ G^{21} \ G^{13} \}$

Solution path found is  $S \ A \ G$  , cost 18

Number of nodes expanded (including goal node) = 7

# Depth-First Search

Expanded node	Nodes list
	{ S <sup>0</sup> }
S <sup>0</sup>	{ A <sup>3</sup> B <sup>1</sup> C <sup>8</sup> }
A <sup>3</sup>	{ D <sup>6</sup> E <sup>10</sup> G <sup>18</sup> B <sup>1</sup> C <sup>8</sup> }
D <sup>6</sup>	{ E <sup>10</sup> G <sup>18</sup> B <sup>1</sup> C <sup>8</sup> }
E <sup>10</sup>	{ G <sup>18</sup> B <sup>1</sup> C <sup>8</sup> }
G <sup>18</sup>	{ B <sup>1</sup> C <sup>8</sup> }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

# Uniform-Cost Search

Expanded node	Nodes list
	{ S <sup>0</sup> }
S <sup>0</sup>	{ B <sup>1</sup> A <sup>3</sup> C <sup>8</sup> }
B <sup>1</sup>	{ A <sup>3</sup> C <sup>8</sup> G <sup>21</sup> }
A <sup>3</sup>	{ D <sup>6</sup> C <sup>8</sup> E <sup>10</sup> G <sup>18</sup> G <sup>21</sup> }
D <sup>6</sup>	{ C <sup>8</sup> E <sup>10</sup> G <sup>18</sup> G <sup>1</sup> }
C <sup>8</sup>	{ E <sup>10</sup> G <sup>13</sup> G <sup>18</sup> G <sup>21</sup> }
E <sup>10</sup>	{ G <sup>13</sup> G <sup>18</sup> G <sup>21</sup> }
G <sup>13</sup>	{ G <sup>18</sup> G <sup>21</sup> }

Solution path found is S B G, cost 13

Number of nodes expanded (including goal node) = 7

# **TRÍ TUỆ NHÂN TẠO**

Chương 3: Kỹ thuật tìm kiếm có sử dụng thông tin

# Chương 3: Kỹ thuật tìm kiếm có sử dụng thông tin

## Nội dung

- Giới thiệu
- Tìm kiếm tham lam tốt nhất
- Tìm kiếm A\*
- Bài tập
- Tìm kiếm leo đồi

# Giới thiệu

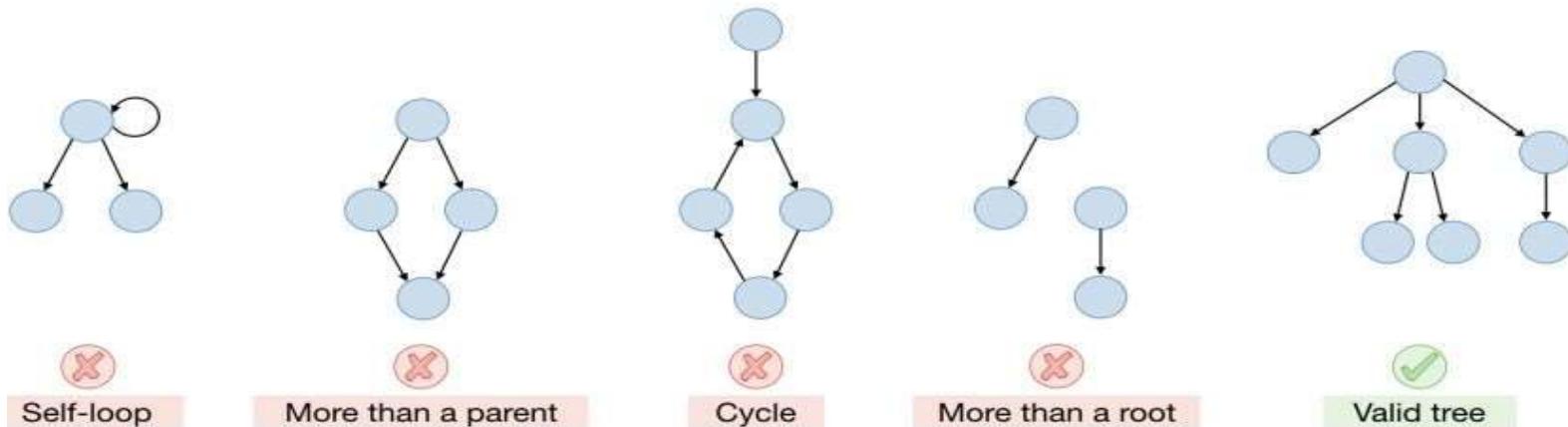
---

- **Tối ưu tìm kiếm** (Search Optimization):
  - Trạng thái s
  - Hành động a từ trạng thái s dẫn đến trạng thái Suss(s,a)
  - Mục tiêu xây dựng chuỗi hành động ( $a_1, a_2, a_n, \dots$ ) từ trạng thái ban đầu đến trạng thái kết thúc.
  - **Vấn đề:** Cần tìm đường đi với chi phí bé nhất

# Giới thiệu

- **Cây tìm kiếm** (Tree search):

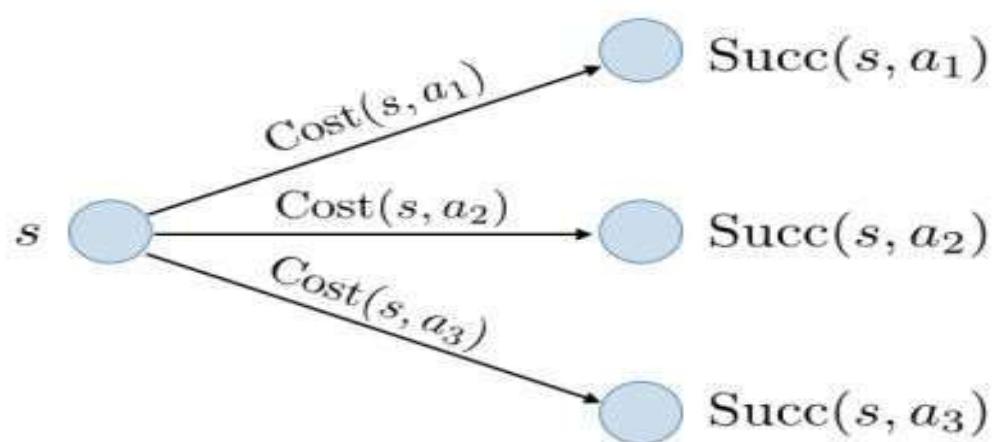
- Khám phá tất cả các trạng thái và hành động có thể có
- Kỹ thuật tìm với không gian bộ nhớ phù hợp ( như DFS  $\mathbf{O}(b \cdot m)$ )
- Hạn chế về thời gian thực thi thường khai triển theo cấp số nhân (như BFS và DFD  $\mathbf{O}(b^d)$ ).



# Giới thiệu

---

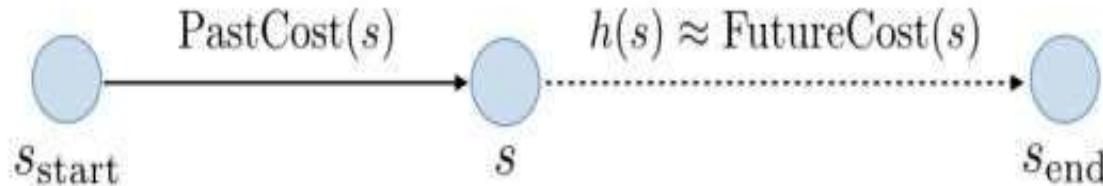
- **Vấn đề tìm kiếm** (Search problem) được định nghĩa:
  - Trạng thái ban đầu  $s$
  - Tập hành động từ trạng thái  $s$ : **Action** ( $s$ )
  - Chi phí thực hiện hành động  $a$  từ trạng thái  $s$ : **Cost**( $s,a$ )
  - Kết quả thực hiện trạng thái  $s$  bởi hành động  $a$ : **Succ**( $s,a$ )
  - Trạng thái kết thúc  $g$



# Giới thiệu

---

- Kỹ thuật tìm kiếm mù:
  - Kỹ thuật BFS và DFS kém hiệu quả.
- Khắc phục:
  - Tìm trạng thái đang hướng tới trạng thái đích.
  - Tìm **hàm** đo khoảng cách đến trạng thái đích.
- Hàm kinh nghiệm (Heuristic function)
  - Hàm heuristic là hàm  $h$  tại trạng thái  $s$  và được ước lượng tới trạng thái đích  $h(s) = \mathbf{FutureCost}(s)$



# Giới thiệu

---

- Kỹ thuật tìm kiếm có thông tin
  - Xây dựng hàm đánh giá có vai trò then chốt.
  - Tính hiệu quả của thuật toán phụ thuộc vào hàm đánh giá.
- Hàm đánh giá trong bài toán tìm đường đi:
  - Hàm đánh giá là đường chim bay.
  - Hàm đánh giá là khoảng cách thực giữa hai địa điểm.
  - Hàm đánh giá là khoảng cách thực và trọng số bổ sung trên đường đi.
- **Tóm lại: Hàm đánh giá tùy thuộc vào vấn đề cần giải.**

# Giới thiệu

---

- Kỹ thuật tìm kiếm có thông tin được dạy
  - **Tìm kiếm ăn tham tốt nhất đầu tiên (Greedy best-first search).**
  - **Tìm kiếm A\* (A star search).**
  - Thuật toán leo đồi (Hill-climbing search).

# Heuristic

---

- **Heuristic là gì?**
  - Heuristic là những tri thức được rút tóm từ những kinh nghiệm, “trực giác” của con người.
  - Heuristic có thể là những tri thức “đúng” hay “sai”.
  - Heuristic là những meta knowledge và “thường đúng”.
- **Heuristic dùng để làm gì?**
  - Trong những bài toán tìm kiếm trên không gian trạng thái, có 2 trường hợp cần đến heuristic:
    1. Vấn đề có thể không có nghiệm chính xác do các mệnh đề không phát biểu chặt chẽ hay thiếu dữ liệu để khẳng định kết quả.
    2. Vấn đề có nghiệm chính xác nhưng phí tổn tính toán để tìm ra nghiệm là quá lớn (hệ quả của bùng nổ tổ hợp)
  - Heuristic giúp tìm kiếm đạt kết quả với chi phí thấp hơn

# Heuristic (tt)

---

- **Heuristic dùng như thế nào trong SSS?**
  - Tìm kiếm trên không gian trạng thái theo chiều nào? Sâu hay rộng?
  - Tìm theo Heuristic : Heuristic định hướng quá trình tìm kiếm theo hướng mà “nó” cho rằng khả năng đạt tới nghiệm là cao nhất. Không “sâu” cũng không “rộng”
- **Kết quả của tìm kiếm với Heuristic**
  - Việc tìm kiếm theo định hướng của heuristic có kết quả tốt hay xấu tùy theo heuristic “đúng” hay “sai”.
  - Heuristic có khả năng bỏ xót nghiệm
  - Heuristic càng tốt càng dẫn đến kết quả nhanh và tốt.
    - **Làm sao tìm được Heuristic tốt???**

# Tìm kiếm tham lam tốt nhất đầu tiên (Greedy best-first search)

# Tìm kiếm tham lam tốt nhất đầu tiên (Greedy best-first search)

# Tìm kiếm tham lam tốt nhất đầu tiên

---

- Ý tưởng:
  - sử dụng hàm đánh giá.
  - sử dụng tìm kiếm theo chiều rộng.
- Hàm đánh giá  $h(u)$ 
  - Hàm ước lượng đến trạng thái kết thúc.
- Tìm kiếm tham lam tốt nhất đầu tiên
  - Các trạng thái được phát dựa vào hàm đánh giá.
  - Các trạng thái không được phát sinh lần lượt theo BFS.
- Hướng cài đặt
  - Dùng hàng đợi có sự ưu tiên dựa vào hàm đánh giá.

# Tìm kiếm tham lam tốt nhất đầu tiên

---

- Ý tưởng:
  - sử dụng hàm đánh giá.
  - sử dụng tìm kiếm theo chiều rộng.
- Hàm đánh giá  $h(u)$ 
  - Hàm ước lượng đến trạng thái kết thúc.
- Tìm kiếm tham lam tốt nhất đầu tiên
  - Các trạng thái được phát dựa vào hàm đánh giá.
  - Các trạng thái không được phát sinh lần lượt theo BFS.
- Hướng cài đặt
  - Dùng hàng đợi có sự ưu tiên dựa vào hàm đánh giá.

# Tìm kiếm tham lam tốt nhất đầu tiên

- Ý tưởng:  $h(u)$  và Queue

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

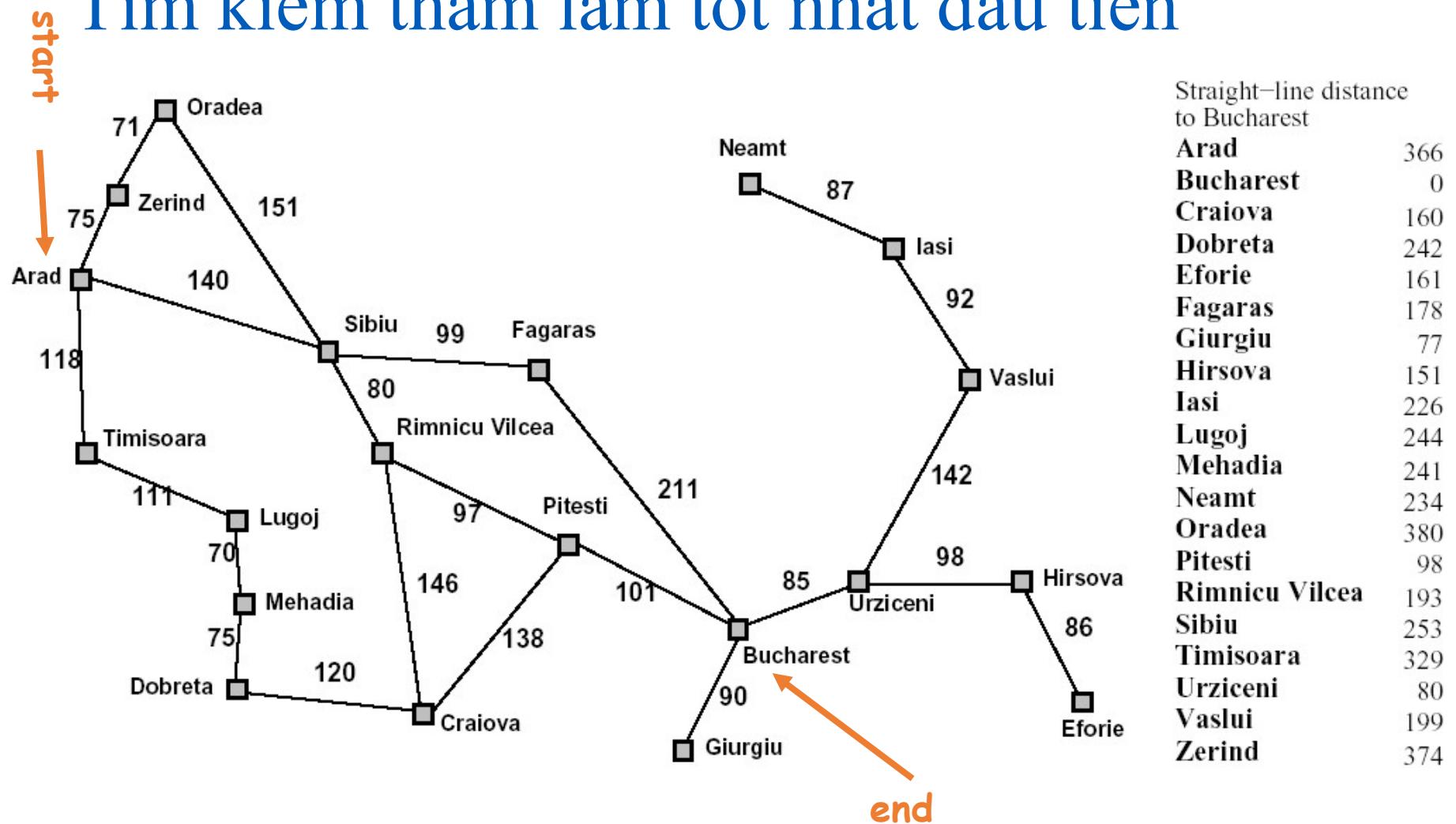
        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

khoảng cách ước lượng đến đích

Cập nhật

# Tìm kiếm tham lam tốt nhất đầu tiên



# Tìm kiếm tham lam tốt nhất đầu tiên

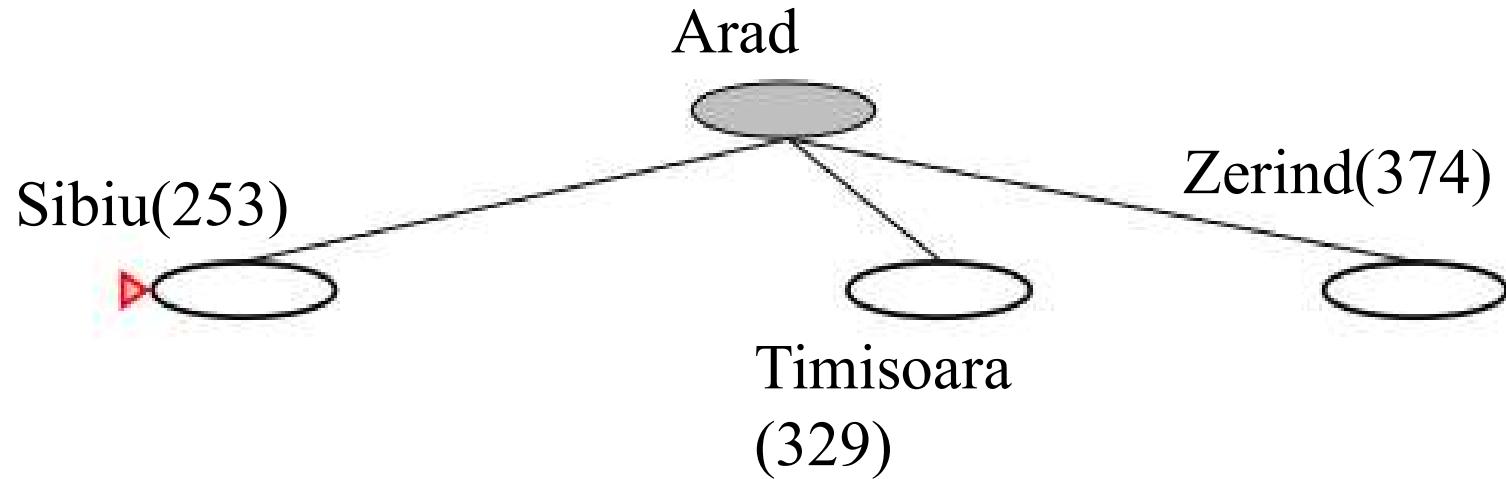
---

Arad (366)  


- Assume that we want to use greedy search to solve the problem of travelling from Arad to Bucharest.
- The initial state=Arad

# Tìm kiếm tham lam tốt nhất đầu tiên

---



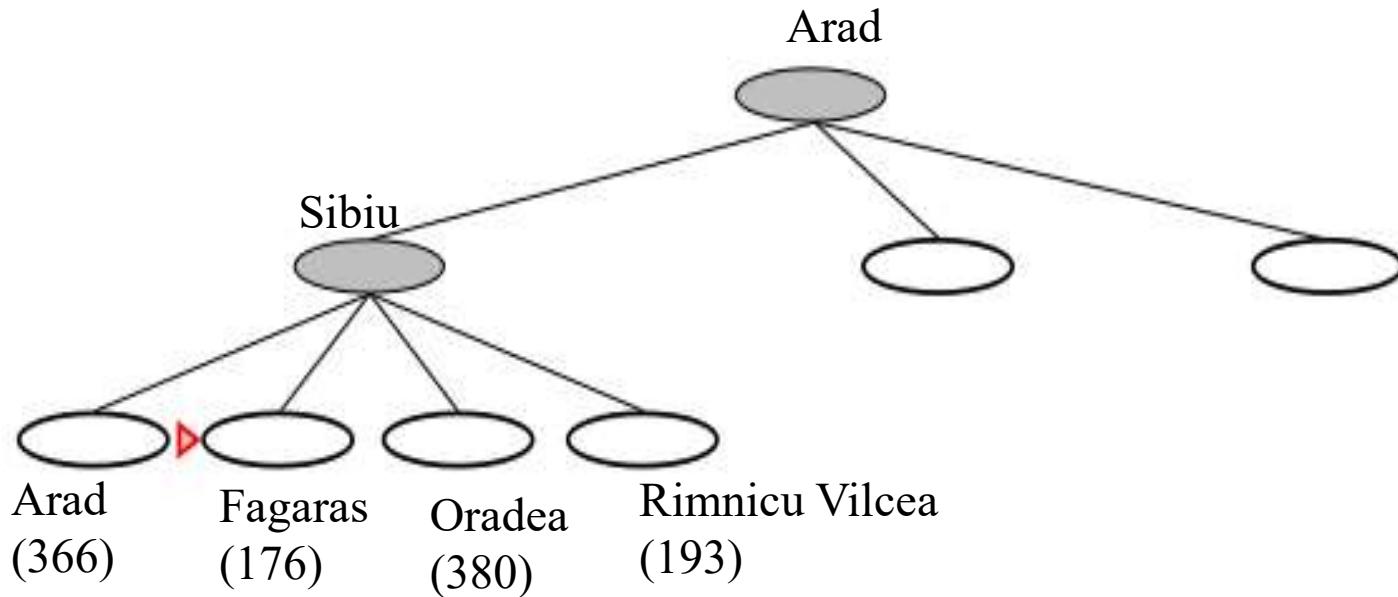
The first expansion step produces:

Sibiu, Timisoara and Zerind

Greedy best-first will select Sibiu.

# Tìm kiếm tham lam tốt nhất đầu tiên

---



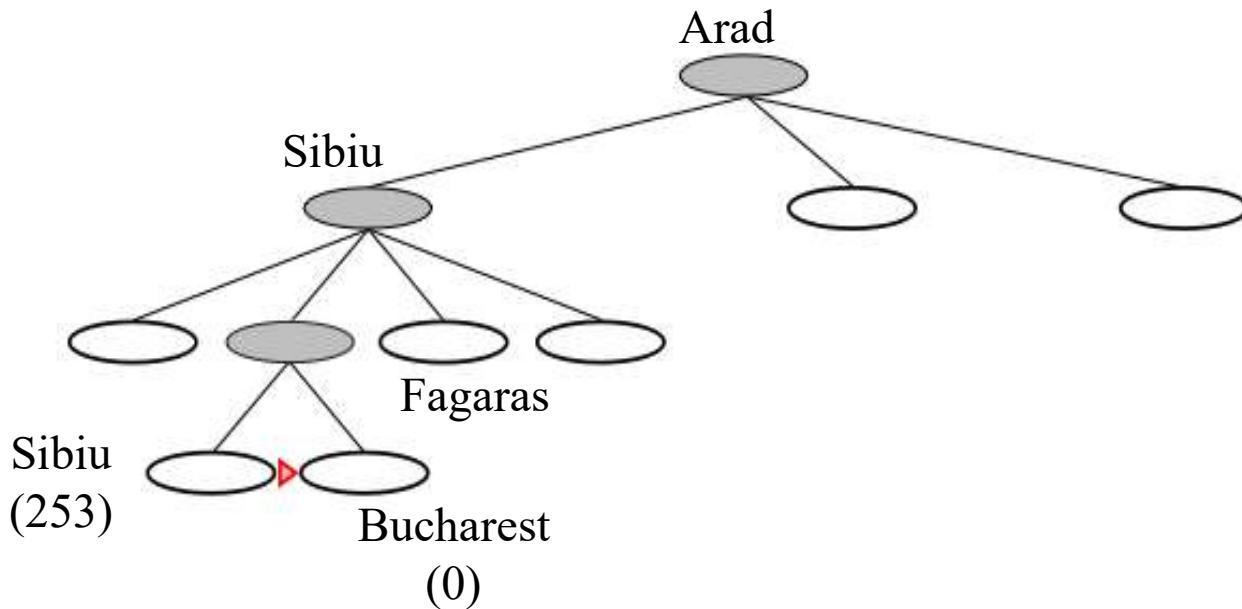
If Sibiu is expanded we get:

Arad, Fagaras, Oradea and Rimnicu Vilcea

Greedy best-first search will select: Fagaras

# Tìm kiếm tham lam tốt nhất đầu tiên

---



If Fagaras is expanded we get:

Sibiu and Bucharest

Goal reached !!

Yet not optimal (see Arad, Sibiu, Rimnicu Vilcea, Pitesti)

# Tìm kiếm tham lam tốt nhất đầu tiên

- Quá trình biến đổi frontier ?

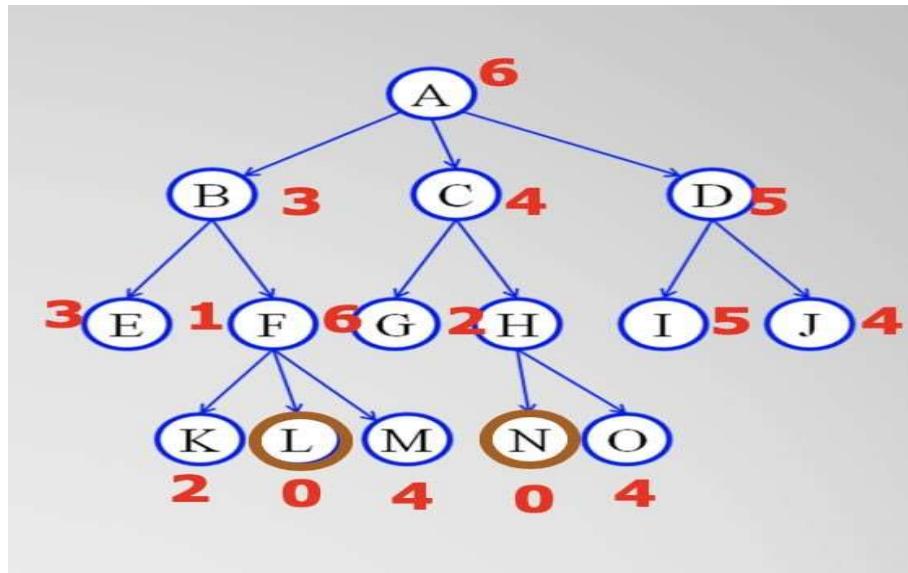
Frontier = {A(6)}

Frontier = {B(3), C(4),D(5)}

Frontier = {F(1),E(3), C(4), D(5)}

Frontier = {L(0),K(2),M(4),E(3),  
C(4), D(5)}{}

Frontier = ....



```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

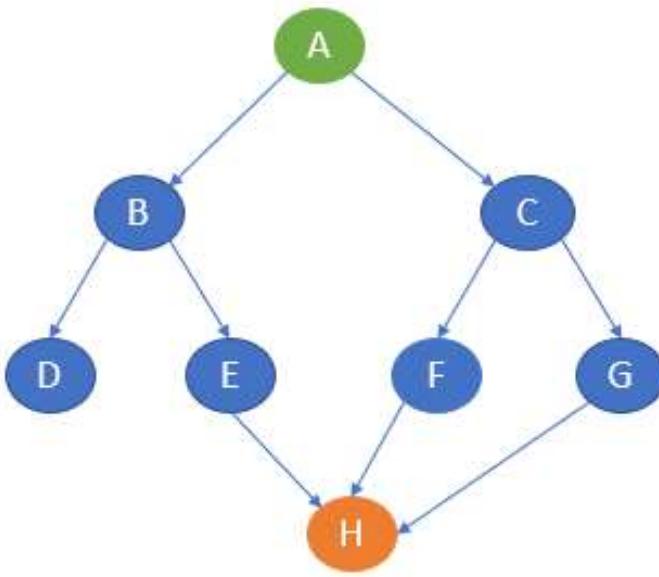
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

# Bài tập tại lớp



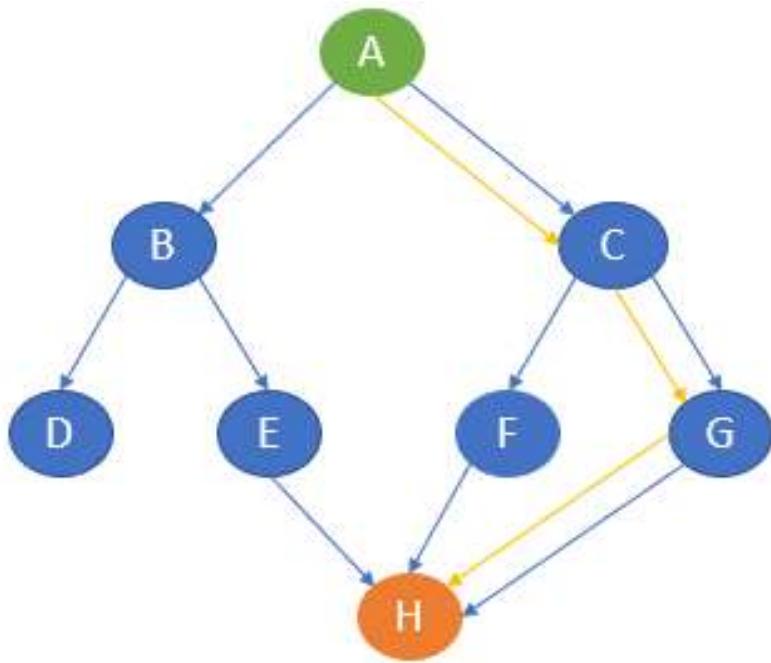
NODES	HEURISTICS
A	13
B	12
C	4
D	7
E	3
F	8
G	2
H	0

Tìm đường đi từ A-H

- 1) Tìm kiếm theo chiều rộng,
- 2) Tìm kiếm theo chiều sâu
- 3) Tìm kiếm tham lam tốt nhất

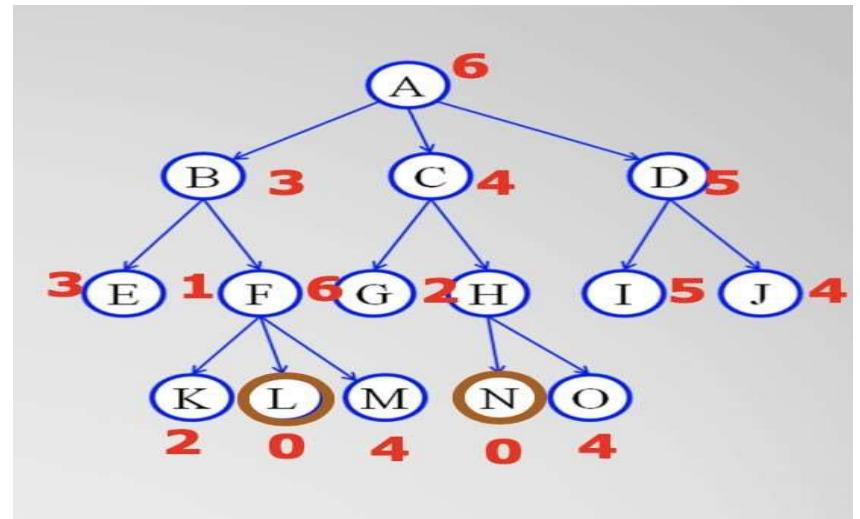
# Tìm kiếm tham lam tốt nhất đầu tiên

---



# Demo(10 phút)

```
1 import heapq
2 from TreeNode import Tree
3 def update_frontier(frontier, new_node):
4     for idx, n in enumerate(array_of_node):
5         if n == new_node:
6             if frontier[idx].goal_cost > new_node.goal_cost:
7                 frontier[idx] = new_node
8 def GBF_search(initial_state, goalTest):
9     frontier = []
10    explored = []
11    heapq.heapify(frontier)
12    heapq.heappush(frontier, initial_state)
13    while len(frontier) > 0:
14        state = heapq.heappop(frontier)
15        explored.append(state)
16        if state == goalTest:
17            return explored;
18        for neighbor in state.get_children():
19            if neighbor not in (frontier and explored):
20                heapq.heappush(frontier, neighbor)
21            elif neighbor in frontier:
22                update_frontier(frontier=frontier, new_node=neighbor)
23    return False
```



```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE : /* Cost  $f(n) = h(n)$  */
  frontier = Heap.new(initialState)
  explored = Set.new()

  while not frontier.isEmpty():
    state = frontier.deleteMin()
    explored.add(state)

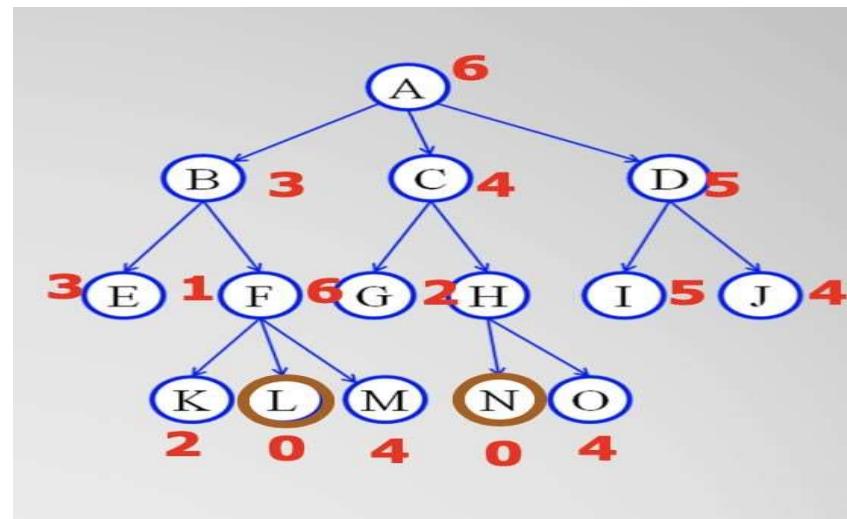
    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      if neighbor not in frontier ∪ explored:
        frontier.insert(neighbor)
      else if neighbor in frontier:
        frontier.decreaseKey(neighbor)

  return FAILURE
```

# Demo(7 phút)

```
24  if __name__ == '__main__':
25      A = Tree("A",6)
26      B = Tree("B",3)
27      C = Tree("C",4)
28      D = Tree("D",5)
29      E = Tree("E",3)
30      F = Tree("F",1)
31      G = Tree("G",6)
32      H = Tree("H",2)
33      I = Tree("I",5)
34      J = Tree("J",4)
35      K = Tree("K",2)
36      L = Tree("L",0)
37      M = Tree("M",4)
38      N = Tree("N",0)
39      O = Tree("O",4)
40      A.add_child(B)
41      A.add_child(C)
42      A.add_child(D)
43      B.add_child(E)
44      B.add_child(F)
45      C.add_child(G)
46      C.add_child(H)
47      D.add_child(I)
48      D.add_child(J)
49      F.add_child(K)
50      F.add_child(L)
51      F.add_child(M)
52      H.add_child(N)
53      H.add_child(O)
54      result = GBF_search(A,L)
55      if result:
56          s = 'explored: '
57          for i in result:
58              s+=i.data + " "
59              print(s)
60      else:
61          print('404 not Found!')
```



# Tìm kiếm A \*

## (A Star search )

# Tìm kiếm A \*

---

- Tìm kiếm đường đi tốt nhất từ trạng thái đầu đến trạng thái đích
- Ý tưởng:
  - sử dụng hàm kinh nghiệm chấp nhận được
  - kỹ thuật tìm kiếm **BFS**
- Khám phá trạng thái s
  - $g(u)$  - độ dài từ trạng thái đầu đến trạng thái u
  - $h(u)$  - hàm heuristic
- Hàm  $h(u)$  chấp nhận được
  - $h(u) \leq h^*(u)$ ,
  - $h^*(u)$  - giá trị thực tế từ u đến trạng thái đích.
- Để tăng hiệu quả tìm kiếm:  $f(u) = g(u) + h(u)$ 
  - $f(u)$  - lượng giá từ trạng thái đầu tới đích qua trạng thái u

# Tìm kiếm A \*

---

- Tìm kiếm đường đi tốt nhất từ trạng thái đầu đến trạng thái đích
- Ý tưởng:
  - sử dụng hàm kinh nghiệm chấp nhận được
  - kỹ thuật tìm kiếm **BFS**
- Khám phá trạng thái s
  - $c(u)$  - độ dài từ trạng thái đầu đến trạng thái u
  - $h(u)$  - hàm heuristic
- Hàm  $h(u)$  chấp nhận được
  - $h(u) \leq h^*(u)$ ,
  - $h^*(u)$  - giá trị thực tế từ u đến trạng thái đích.
- Để tăng hiệu quả tìm kiếm:  $f(u) = c(u) + h(u)$ 
  - $f(u)$  - lượng giá từ trạng thái đầu tới đích qua trạng thái u

# Tìm kiếm A \*

---

- Kết hợp giữa hàm Heuristic và BFS.

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

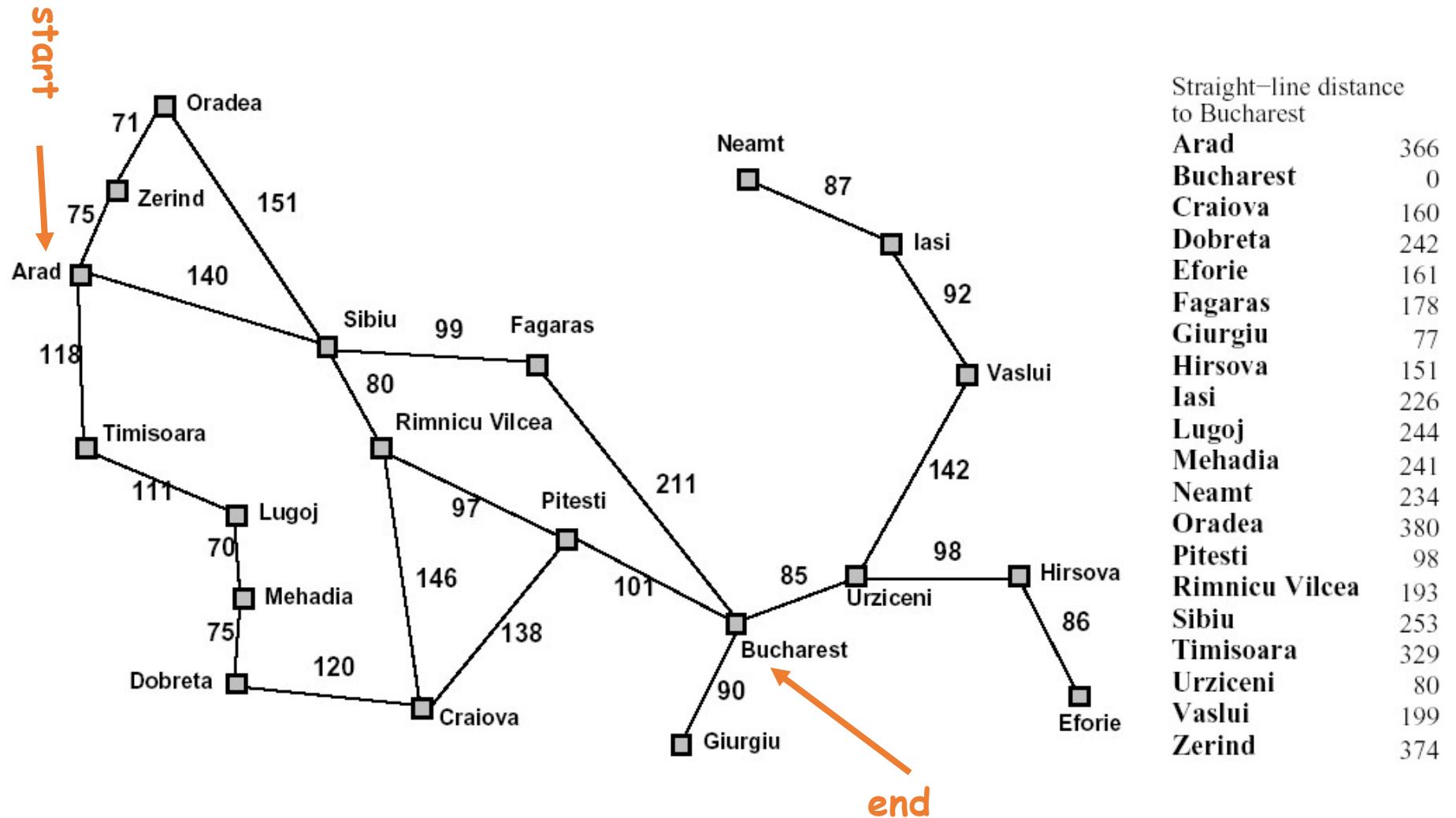
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier  $\cup$  explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

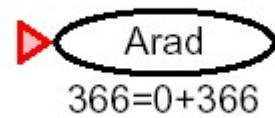
# Tìm kiếm A \*



# Tìm kiếm A \*

---

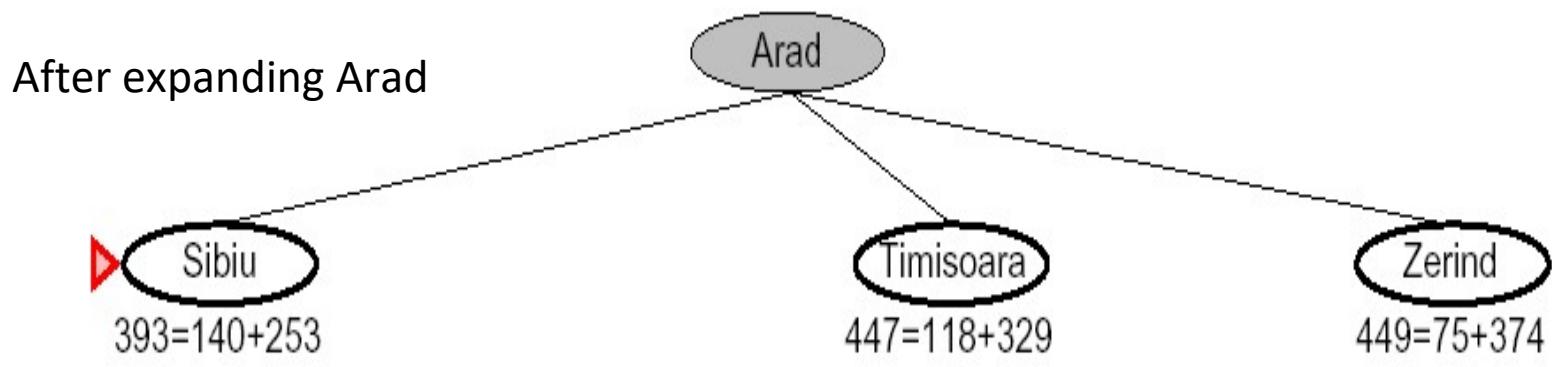
a) The initial state



Find Bucharest starting at Arad

$$f(\text{Arad}) = g(\text{Arad}, \text{Arad}) + h(\text{Arad}) = 0 + 366 = 366$$

# Tìm kiếm A \*



Expand Arad and determine  $f(n)$  for each node

$$f(\text{Sibiu}) = g(\text{Arad}, \text{Sibiu}) + h(\text{Sibiu}) = 140 + 253 = 393$$

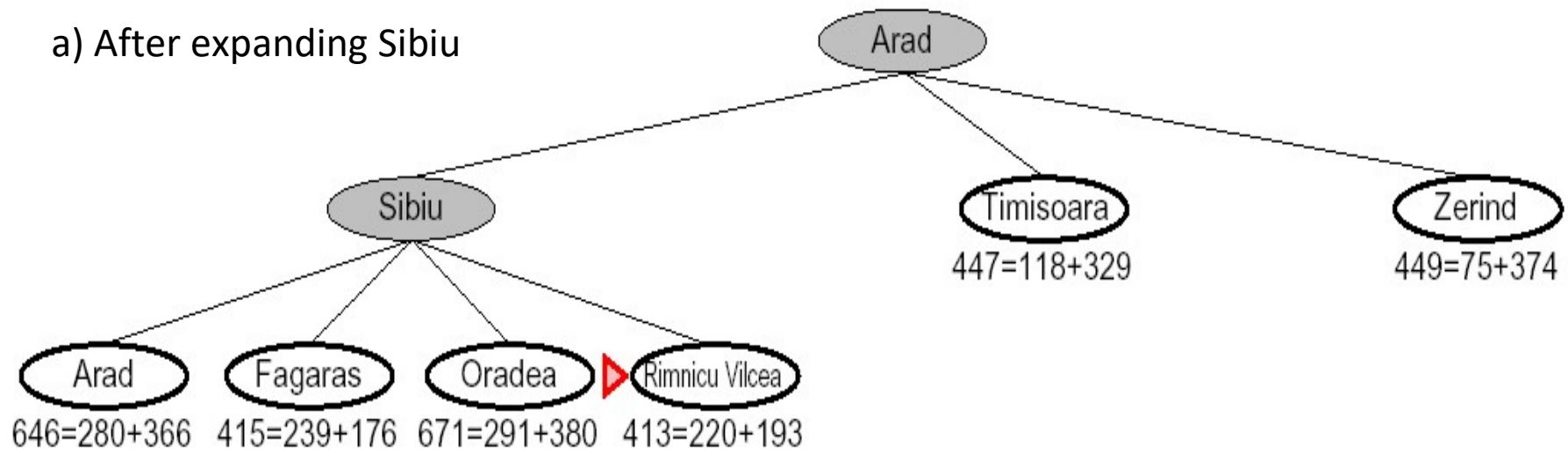
$$f(\text{Timisoara}) = g(\text{Arad}, \text{Timisoara}) + h(\text{Timisoara}) = 118 + 329 = 447$$

$$f(\text{Zerind}) = g(\text{Arad}, \text{Zerind}) + h(\text{Zerind}) = 75 + 374 = 449$$

Best choice is Sibiu

# Tìm kiếm A \*

a) After expanding Sibiu



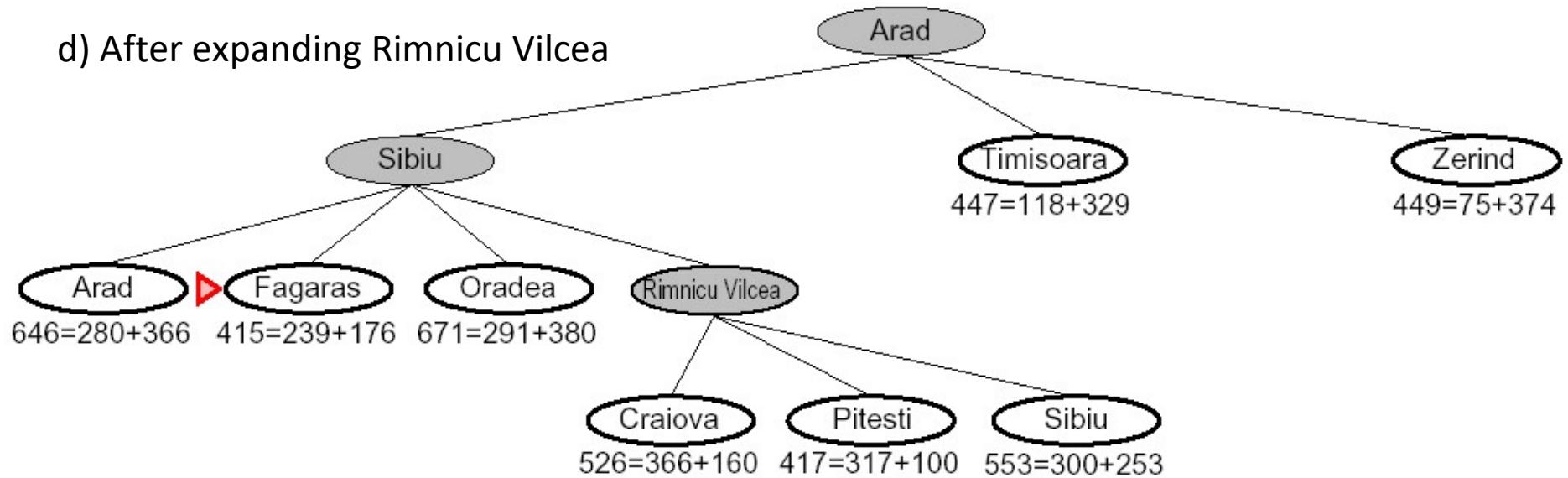
Expand Sibiu and determine  $f(n)$  for each node

- $f(\text{Arad}) = g(\text{Sibiu}, \text{Arad}) + h(\text{Arad}) = 280 + 366 = 646$
- $f(\text{Fagaras}) = g(\text{Sibiu}, \text{Fagaras}) + h(\text{Fagaras}) = 239 + 179 = 415$
- $f(\text{Oradea}) = g(\text{Sibiu}, \text{Oradea}) + h(\text{Oradea}) = 291 + 380 = 671$
- $f(\text{Rimnicu Vilcea}) = g(\text{Sibiu}, \text{Rimnicu Vilcea}) + h(\text{Rimnicu Vilcea}) = 220 + 192 = 413$

Best choice is Rimnicu Vilcea

# Tìm kiếm A \*

d) After expanding Rimnicu Vilcea



Expand Rimnicu Vilcea and determine  $f(n)$  for each node

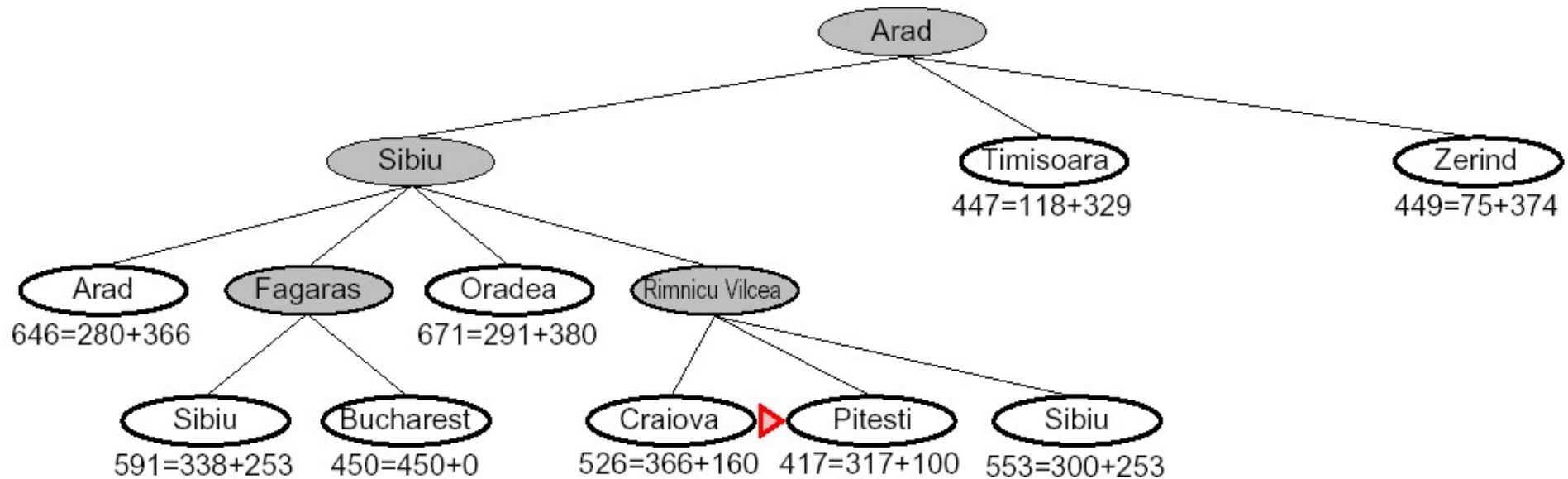
$$f(\text{Craiova}) = g(\text{Rimnicu Vilcea}, \text{Craiova}) + h(\text{Craiova}) = 360 + 160 = 526$$

$$f(\text{Pitesti}) = g(\text{Rimnicu Vilcea}, \text{Pitesti}) + h(\text{Pitesti}) = 317 + 100 = 417$$

$$f(\text{Sibiu}) = g(\text{Rimnicu Vilcea}, \text{Sibiu}) + h(\text{Sibiu}) = 300 + 253 = 553$$

Best choice is Fagaras

# Tìm kiếm A \*



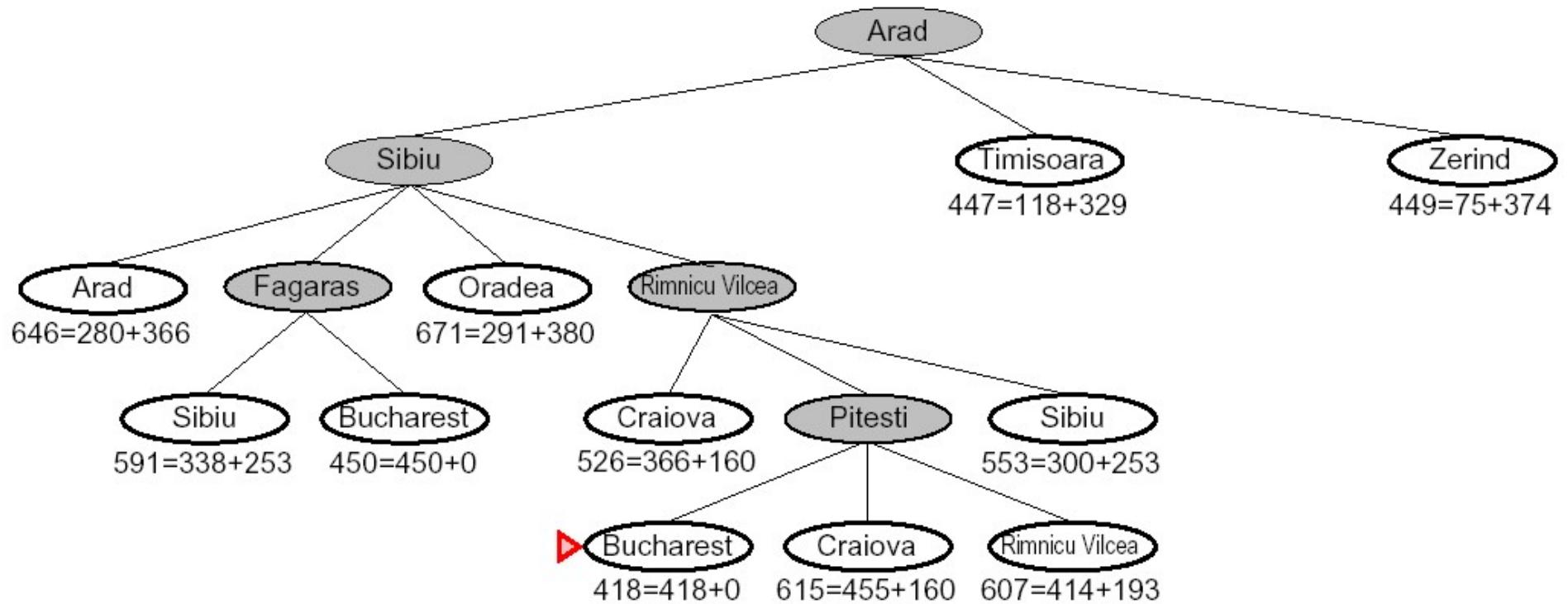
Expand Fagaras and determine  $f(n)$  for each node

$$f(\text{Sibiu}) = g(\text{Fagaras}, \text{Sibiu}) + h(\text{Sibiu}) = 338 + 253 = 591$$

$$f(\text{Bucharest}) = g(\text{Fagaras}, \text{Bucharest}) + h(\text{Bucharest}) = 450 + 0 = 450$$

Best choice is Pitesti !!!

# Tìm kiếm A \*



Expand Pitesti and determine  $f(n)$  for each node

$$f(\text{Bucharest}) = g(\text{Pitesti}, \text{Bucharest}) + h(\text{Bucharest}) = 418 + 0 = 418$$

Best choice is Bucharest !!!

Optimal solution (only if  $h(n)$  is admissible)

Note values along optimal path !!

# Tìm kiếm A\*

- Quá trình biến đổi frontier ?

frontier = A(0+6)

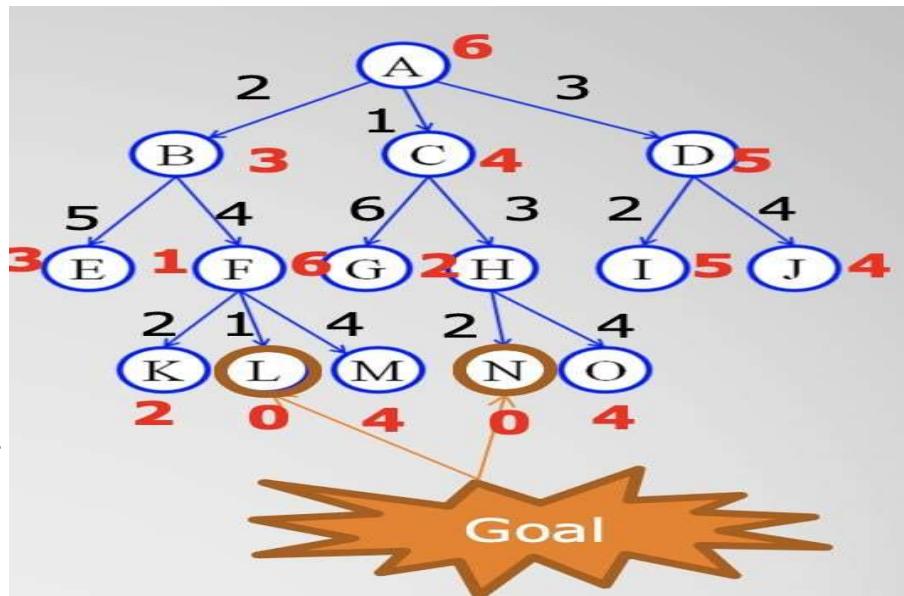
frontier = {B(2+3), C(1+4),D(3+5)}

frontier = {E(7+3), F(6+1),C(1+4),D(3+5)}

frontier = {E(10),  
F(7),G(7+6),H(4+2),D(8)}

frontier = {E(10),  
F(7),G(7+6),**N(6+0)**,O(8+4),D(8)}  
}

frontier = ...



```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost f(n) = g(n) + h(n) */
    frontier = Heap.new(initialState)
    explored = Set.new()
    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)
        if goalTest(state):
            return SUCCESS(state)
        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)
    return FAILURE
```

# Tìm kiếm A \*

- Ví dụ hàm Heuristic chấp nhận được của bài toán 8 số
  - $h_1(u)$  - số lượng ô bị sai.
  - $h_2(u)$ - tổng số lượng ô từ ô hiện tại đến trạng thái đích

7	2	4
5		6
8	3	1

Start State

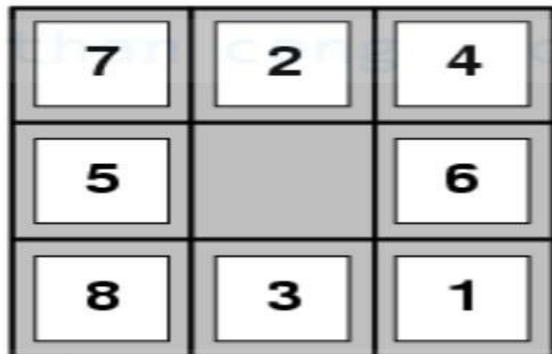
	1	2
3	4	5
6	7	8

Goal State

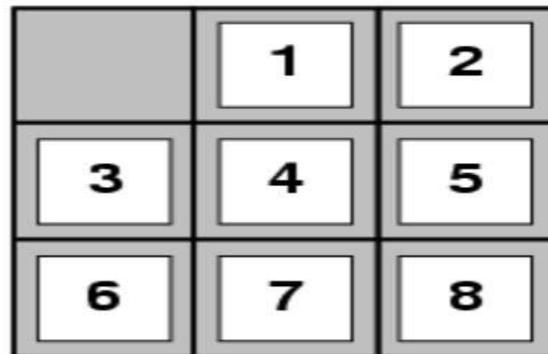
- $h_1(u) = ?$      $h_2(u) = ?$

# Tìm kiếm A \*

- Ví dụ hàm Heuristic chấp nhận được của bài toán 8 số



Start State



Goal State

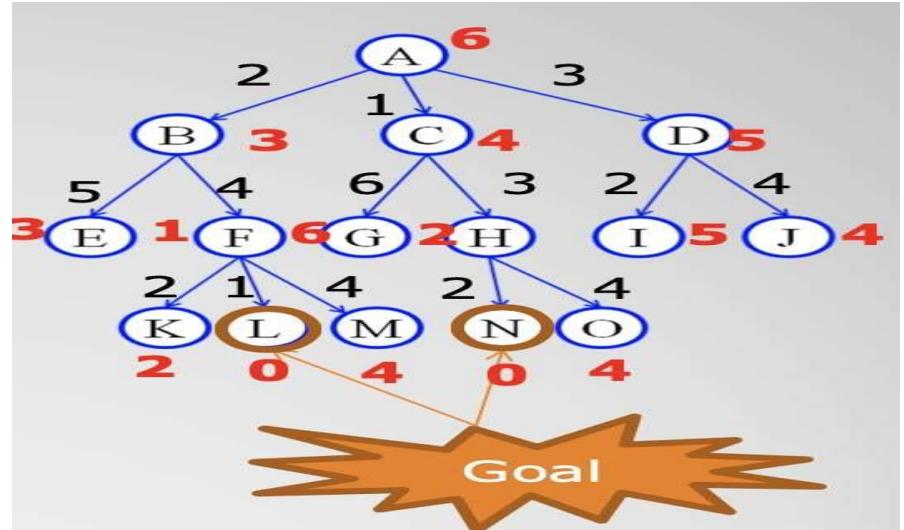
- $h_1(u) = 8$
- $h_2(u) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
- Nếu  $h_2(u) \geq h_1(u)$  với mọi  $u$ .  
 $\Rightarrow h_2(u)$  mạnh hơn  $h_1(u)$  có nghĩa  $h_2(u)$  **tốt hơn**  $h_1(u)$

# Demo(15 phút)

```

90 if __name__ == "__main__":
91     tree = Tree()
92     tree.add_nodes([
93         Node("A", 6),
94         Node("B", 3),
95         Node("C", 4),
96         Node("D", 5),
97         Node("E", 3),
98         Node("F", 1),
99         Node("G", 6),
100        Node("H", 2),
101        Node("I", 5),
102        Node("J", 4),
103        Node("K", 2),
104        Node("L", 0),
105        Node("M", 4),
106        Node("N", 0),
107        Node("O", 4)
108    ])
109    tree.add_edges([
110        ("A", "B", 2),
111        ("A", "C", 1),
112        ("A", "D", 3),
113        ("B", "E", 5),
114        ("B", "F", 4),
115        ("C", "G", 6),
116        ("C", "H", 3),
117        ("D", "I", 2),
118        ("D", "J", 4),
119        ("F", "K", 2),
120        ("F", "L", 1),
121        ("F", "M", 4),
122        ("H", "N", 2),
123        ("H", "O", 4),
124    ])
125    tree.nodes[0].cost = 0
126    #print(tree.edges)
127    result = A_Star(tree, tree.nodes[0], tree.nodes[14])
128    if result:
129        s = 'explored: '
130        for i in result:
131            s += i.label + " "
132            print(s);
133    else:
134        print('404 Not Found!')

```



```

function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE

```

# Demo(15 phút)

---

```
69 def update_cost(tree, current_node, prev_node):
70     if tree.get_edge(prev_node, current_node) is not None:
71         #print("OK")
72         if current_node.cost > prev_node.cost + tree.get_edge(prev_node, current_node)[2]:
73             current_node.cost = prev_node.cost + tree.get_edge(prev_node, current_node)[2]
74 def A_Star(tree, start, end):
75     frontier = [start]
76     heapq.heapify(frontier)
77     explored = []
78     while len(frontier) > 0:
79         state = heapq.heappop(frontier)
80         explored.append(state)
81         print(state)
82         if state == end:
83             return explored
84         for child in state.neighbors():
85             update_cost(tree, child, state)
86             if child.get_label() not in list(set(node.get_label() for node in frontier + explored)):
87                 heapq.heappush(frontier, child)
88     return False
```

# Demo(15 phút)

---

```
2 import heapq
3 class Node:
4     def __init__(self, label, goal_cost):
5         self.label = label
6         self.cost = 10000
7         self.goal_cost = goal_cost
8         self.save_cost = None
9         self.pr = []
10        self.chld = []
11    def __repr__(self):
12        return str(dict({
13            "label": self.label,
14            "cost" : self.cost,
15            "goal cost": self.goal_cost
16        }))
17    def __eq__(self, other):
18        return self.label == other.label
19    def __lt__(self, other):
20        if self.save_cost == 10000:
21            return self.goal_cost + self.cost < other.goal_cost + other.cost
22        else:
23            return self.cost < other.cost
24    def get_label(self):
25        return self.label
26    def neighbors(self):
27        return self.chld + self.pr
28
```

# Demo(15 phút)

---

```
29 class Tree:
30     def __init__(self):
31         self.nodes = []
32         self.edges = []
33     def add_nodes(self, data):
34         for node in data:
35             self.nodes.append(node)
36     def add_node(self, node):
37         self.nodes.append(node)
38
39     def get_index(self, node):
40         for i, n in enumerate(self.nodes):
41             if n.get_label() == node.get_label():
42                 return i
43         return -1
44     def add_edges(self, tuple_edges):
45         for t in tuple_edges:
46             start_label = t[0]
47             end_label = t[1]
48             w = t[2]
49             index_start_label = self.get_index(Node(start_label, None))
50             index_end_label = self.get_index(Node(end_label, None))
51
52             self.nodes[index_start_label].chld.append(self.nodes[index_end_label])
53             self.nodes[index_end_label].pr.append(self.nodes[index_start_label])
54             self.edges.append((self.nodes[index_start_label], self.nodes[index_end_label], t[2]))
55
56     def show_nodes(self):
57         return [node.get_data() for node in self.nodes]
58     def get_edge(self, start_node, end_node):
59         try:
60             return [edges for edges in self.edges if edges[0] == start_node
61                     and edges[1] == end_node][0]
62         except:
63             return None
```

# Bài tập 1

---

- Xây dựng chương trình cài đặt kỹ thuật tìm kiếm tham lam tốt nhất đầu tiên và kỹ thuật tìm kiếm A\*

-



Tìm kiếm leo đồi (Hill-climbing search )

# Tìm kiếm leo đồi

---

- Chọn một trạng thái tốt hơn trạng thái hiện hành để mở rộng. Nếu không, thuật toán phải dừng
- Nếu chỉ chọn một trạng thái tốt hơn: leo đồi đơn giản; trạng thái tốt nhất: leo đồi dốc đứng
- Sử dụng hàm  $H$  để biết trạng thái nào tốt hơn
- Khác với tìm kiếm sâu, leo đồi không lưu tất cả các con mà chỉ lưu đúng một trạng thái được chọn nếu có

# Tìm kiếm leo đồi

---

- Ý tưởng:
  - Sử dụng hàm đánh giá.
  - Sử dụng tìm kiếm theo chiều sâu.
- Hàm đánh giá  $h(u)$ 
  - Hàm ước lượng đến trạng thái kết thúc.
- Tìm kiếm leo đồi
  - Các trạng thái được phát dựa vào hàm đánh giá.
  - Các trạng thái không được phát sinh lần lượt theo DFS.
- Hướng cài đặt
  - Dùng ngăn xếp có sự ưu tiên dựa vào hàm đánh giá.

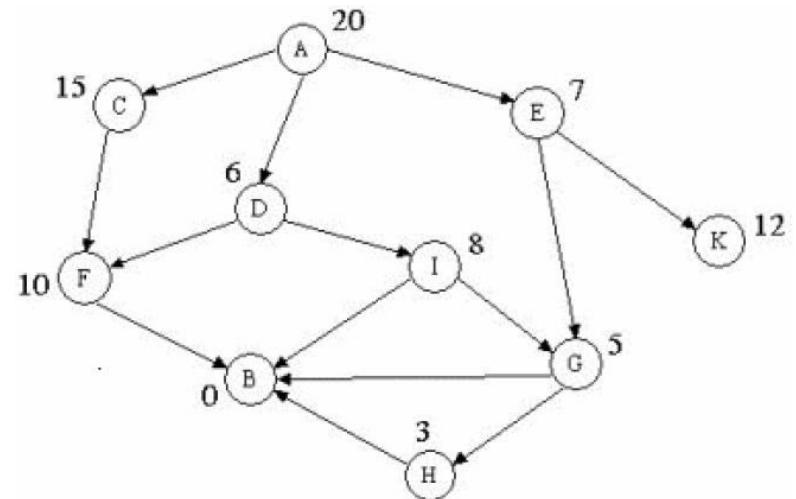
# Tìm kiếm leo đồi

---

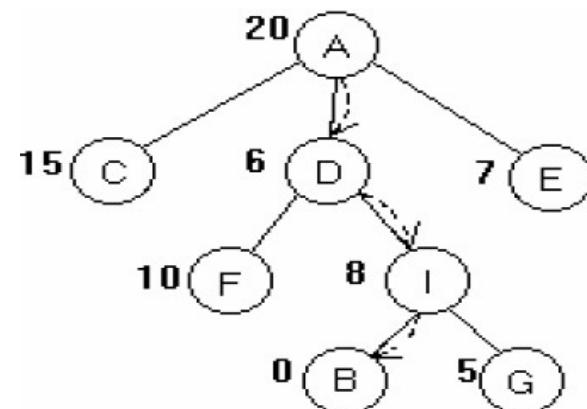
- Ý nghĩa:
  - Phương pháp này được thực hiện nhờ hàm đánh giá.
  - Khác với phương pháp tìm kiếm theo chiều sâu, khi phát triển đỉnh u, chọn trong số các đỉnh con của u, đỉnh nào có nhiều hứa hẹn nhất thì phát triển.

# Tìm kiếm leo đồi

- **Đầu vào:**
  - Trạng thái đầu là A,
  - Trạng thái kết thúc là B.
- **Thực hiện:**
  - A được xét  $\rightarrow$  C, D, E.
  - Chọn D, vì  $h(D) = 6$  (min), sinh ra F,I.
  - Trong số các đỉnh con của D, chọn I, vì  $h(I) = 8$ .
  - Với I được chọn, sinh ra B và G.
  - B là trạng thái kết thúc.

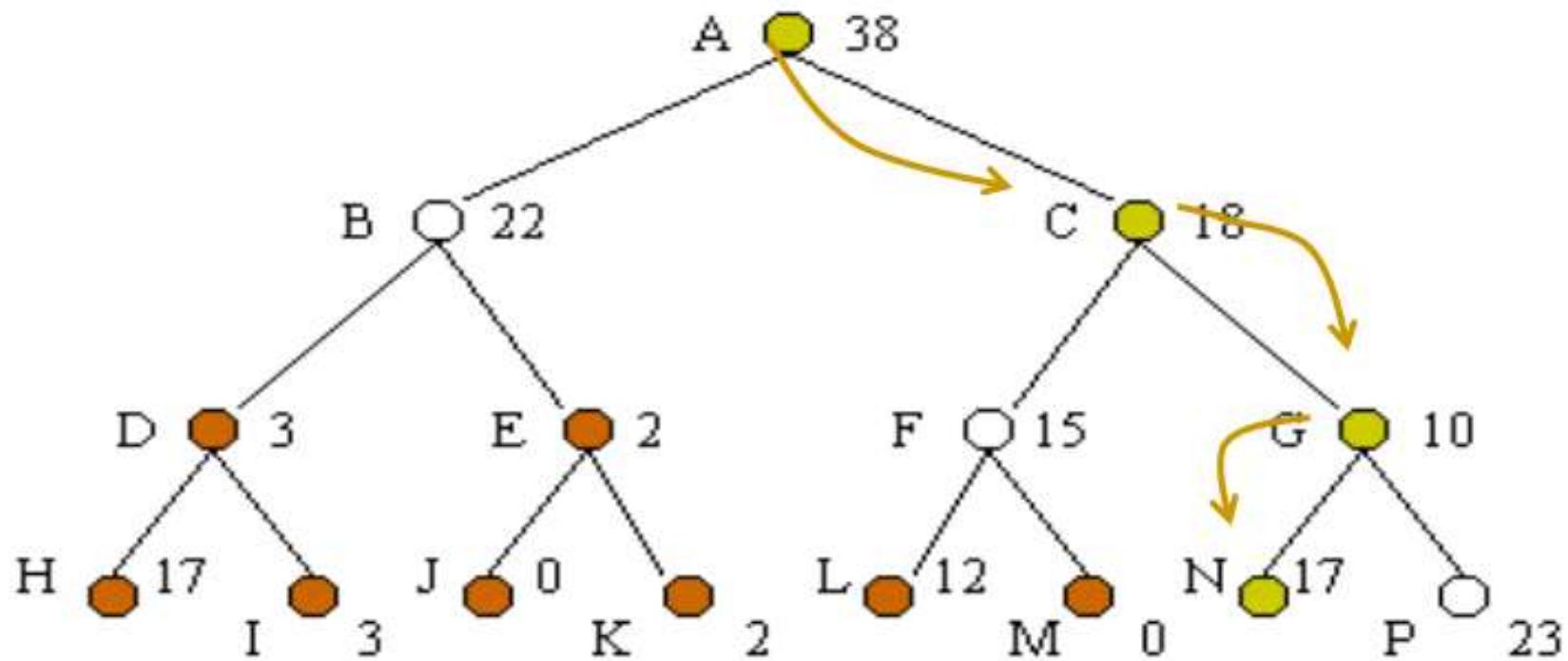


Xét không gian trạng thái sau



# Tìm kiếm leo đồi

---



# Cài đặt Hill-Climbing Search

---

**Procedure** Hill\_Climbing\_Search;

**Begin**

1. Khởi tạo danh sách **L** chỉ chứa trạng thái đầu;
2. **Loop do**
  1. **If** **L** rỗng **then** { thông báo thất bại; **stop**; }
  2. Loại trạng thái **u** đầu danh sách **L**;
  3. **If** **u** là trạng thái kết thúc **then** { thông báo thành công; **stop**; }
  4. **For** mỗi trạng thái **v** kề **u** đặt **v** vào **L** sao cho các phần tử được đưa vào đầu danh sách **L** có đánh giá giảm dần;
3. **End**;

# Cài đặt Hill-Climbing Search

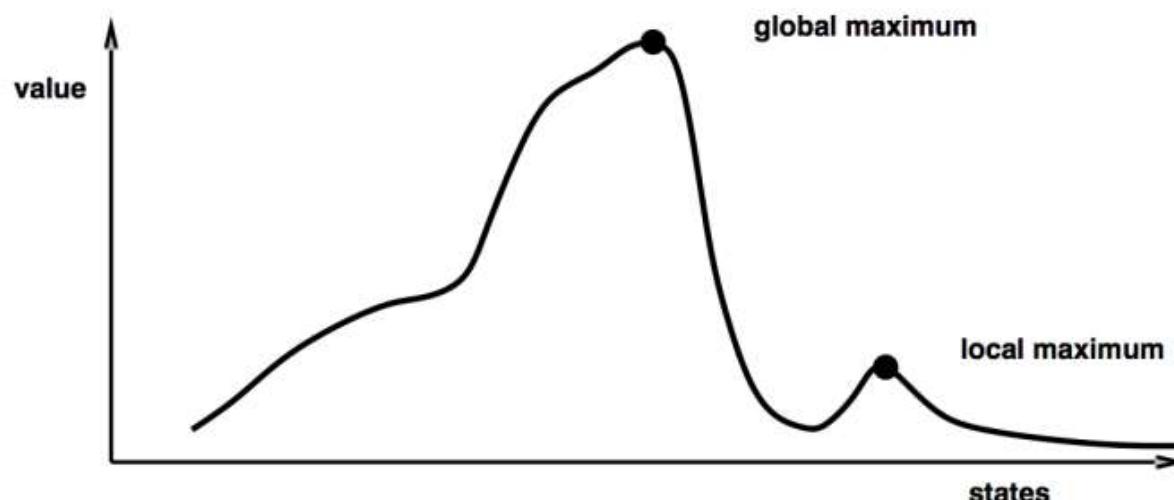
---

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if VALUE[neighbor]  $\leq$  VALUE[current] then return STATE[current]
    current  $\leftarrow$  neighbor
```

# Hill-Climbing Search

## Giới hạn

- Có khuynh hướng bị sa lầy ở những cực đại cục bộ
  - Lời giải tìm được không tối ưu
  - Không tìm được lời giải mặc dù có tồn tại lời giải
- Có thể gặp vòng lặp vô hạn do không lưu giữ thông tin về các trạng thái đã duyệt



## Bài tập 2

---

- Mỗi sinh viên chọn 01 kỹ thuật tìm kiếm bên dưới để nghiên cứu để hiểu kỹ thuật tìm kiếm đã chọn:
  - Tìm kiếm nhánh và cận (Branch and Bound)
  - Tìm kiếm cục bộ (Local search algorithms)
  - Tìm kiếm mô phỏng luyện kim (Simulated annealing search)
  - Thuật toán gen (Genetic algorithms)

-

Bài tập 3: Cho mê cung như bên dưới, tìm đường đi ngắn nhất theo thuật toán BFS, DFS, A\*.

---

```
maze = [
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', 'E', '*', '*', '*', ' ', '*', '*', '*', '0'],
    ['0', '*', '0', '*', '*', '0', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', ' ', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', ' ', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', '0', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', ' ', '*', '*', '*', '0'],
    ['0', '*', '0', '*', '*', '0', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', ' ', '*', '*', '*', '0'],
    ['0', '*', '*', '*', '*', '0', '*', '*', '*', '0'],
    ['0', '*', '*', 'X', '*', '0', '*', '*', '*', '0']
]
```

# **TRÍ TUỆ NHÂN TẠO**

Chương 4: Tìm kiếm có đối thủ

# Chương 4: Tìm kiếm có đối thủ

---

## Nội dung

- Giới thiệu
- Cây trò chơi
- Chiến lược Minimax
- Chiến lược cắt tỉa Alpha-Beta

# Giới thiệu

---

- Trò chơi:
  - Đặc tính thông minh của con người
  - Phiên bản “F1” của trí tuệ nhân tạo
- Trò chơi đối kháng phổ biến:
  - Cờ caro
  - Cờ tướng
  - Cờ vua
  - Cờ vây
- Kỹ thuật tìm kiếm mù, tìm kiếm có thông tin:
  - Không giải quyết được cho trò chơi đối kháng

# Giới thiệu

---

- Trò chơi cờ vua
  - Deep Blue được phát triển bởi IBM
  - Đại kiện tướng *Garry Kimovich Kasparov*
  - Năm 1997, Deep Blue 3.5 - 2.5 Kasparov
  - Bí quyết của DeepBlue
    - Tìm kiếm với độ sâu từ 8 hoặc 10 hoặc nhiều hơn.
    - Cứ 1s tính được  $2 \times 10^8$  nước đi (99.99% nước đi được xem là tồi)

# Giới thiệu

---

- Trò chơi cờ vây

- Lee Sedol - 18 lần vô địch thế giới
- AlphaGo của Google DeepMind

Nguồn Wikipedia.org

<b>AlphaGo Lee</b>	48 TPUs,distributed	3,739	<b>4:1 against Lee Sedol</b>	3/2016
AlphaGo Master	4 TPUs,single machine	4,858	60:0 against professional players;	5/2017
AlphaGo Zero (40 block)	4 TPUs,single machine	5,185	100:0 against AlphaGo Lee 89:11 against AlphaGo Master	10/2017
AlphaGo Zero (20 block)	4 TPUs,single machine	5,018	60:40 against AlphaGo Zero (20 block)	12/2017

# Cây trò chơi và tìm kiếm trên cây trò chơi

---

- Trong bài, nghiên cứu các trò chơi có hai người tham gia; như
  - cờ vua,
  - cờ ca rô,
  - cờ tướng...
- Người chơi là quân Trắng, đối thủ là quân Đen.
- **Mục tiêu:** **nghiên cứu giải thuật cho quân Trắng đi.**

# Cây trò chơi (Game tree)

---

- Cây trò chơi mô tả khả năng của trò chơi
- Mỗi đỉnh của cây trò chơi là một quyết định của người chơi
- Mỗi tầng của cây trò chơi ứng với khả năng của mỗi người chơi
- Kết quả của trò chơi - đường đi từ đỉnh gốc đến lá
- Độ sâu cây trò chơi - số lượt đi của hai người chơi

# Cây trò chơi

---

- Hai người chơi hiểu rõ luật chơi và quan sát đầy đủ tình thế, có cùng mức cỗ gắng thăng như nhau
- 2 người thay phiên đưa ra các nước đi tuân theo một luật nào đó
- Các luật trên là như nhau cho cả 2 người.
- Tìm kiếm trên cây trò chơi:
  - Xác định nước đi đủ tốt, để sau nước đi đó, mọi nước đi dẫn đến trạng thái kết thúc.

# Cây trò chơi

---

- Vấn đề tìm kiếm ở đây **khó khăn** hơn với việc tìm kiếm trong các bài trước, vì:
  - Ở trong vấn đề này, có đối thủ, nên không biết đối thủ sẽ đi như thế nào.
  - Nếu có thể tổng quát, cũng sẽ rất khó vì không gian tìm kiếm quá rộng.
  - Nhìn chung, không thể tìm được lời giải tối ưu, chỉ tìm được lời giải xấp xỉ.

# Cây trò chơi

---

- Giải pháp: trong trò chơi, có thể coi như tìm kiếm trong không gian trạng thái, mỗi trạng thái là một tình thế của trò chơi. Có thể tóm tắt giải pháp:
  - Trạng thái ban đầu là sự sắp xếp các quân cờ trong lúc đầu của cuộc chơi.
  - Các nước đi hợp lệ là các toán tử.
  - Các trạng thái kết thúc là các tình thế mà cuộc chơi dừng, thường đã xác định, có thể thông qua hàm kết quả.
  - Có thể biểu diễn không gian trạng thái trên cây trò chơi.

# Cây trò chơi

---

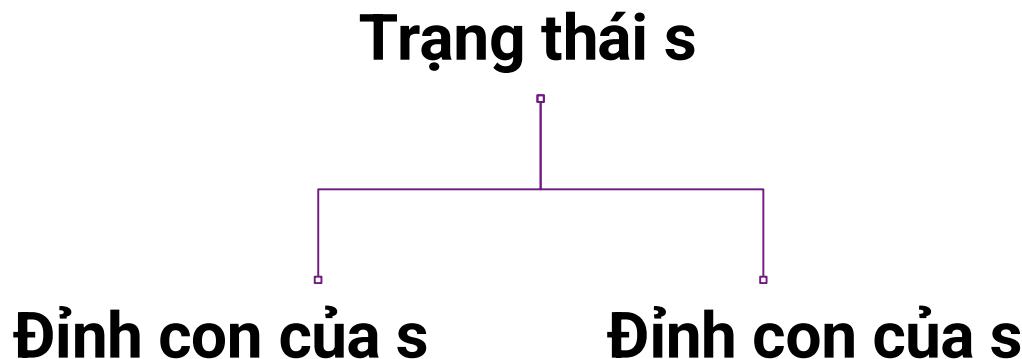
- Cây trò chơi:
  - $s_{start}$ : trạng thái ban đầu của trò chơi
  - **Actions(s)**: tập hành động - nước đi hợp lệ với trạng thái s.
  - **Suss(s,a)**: trạng thái trò chơi sinh ra từ s bởi hành động a.
  - **IsEnd(s)**-trạng thái kết thúc- tình thế mà trò chơi dừng, đặc trưng bởi hàm lợi ít: **Utility(s)**
- Xây dựng cây trò chơi:
  - Gốc của cây ứng với trạng thái s
  - Trạng thái con ứng với nước đi hợp lệ

# Chiến lược Minimax

---

- Ý tưởng:

- Tại thời điểm hiện tại mong muốn xây dựng cách đi tốt nhất cho trạng thái s (xem như gốc của cây trò chơi).
- Xây dựng nước đi hợp lệ cho trạng thái s
- **Nước đi hợp lệ tốt nhất:** đỉnh con có giá trị tốt nhất



# Chiến lược Minimax

---

- **Đỉnh con có giá trị tốt:**
  - Đỉnh con phải có giá trị
  - Giá trị được tính từ đỉnh lá tới các đỉnh con.
- Cây trò chơi sự luân phiên giữa 2 người chơi:
  - Người chơi max: tối đa hóa hàm lợi ít
  - Người chơi min: tối thiểu hóa hàm lợi ít

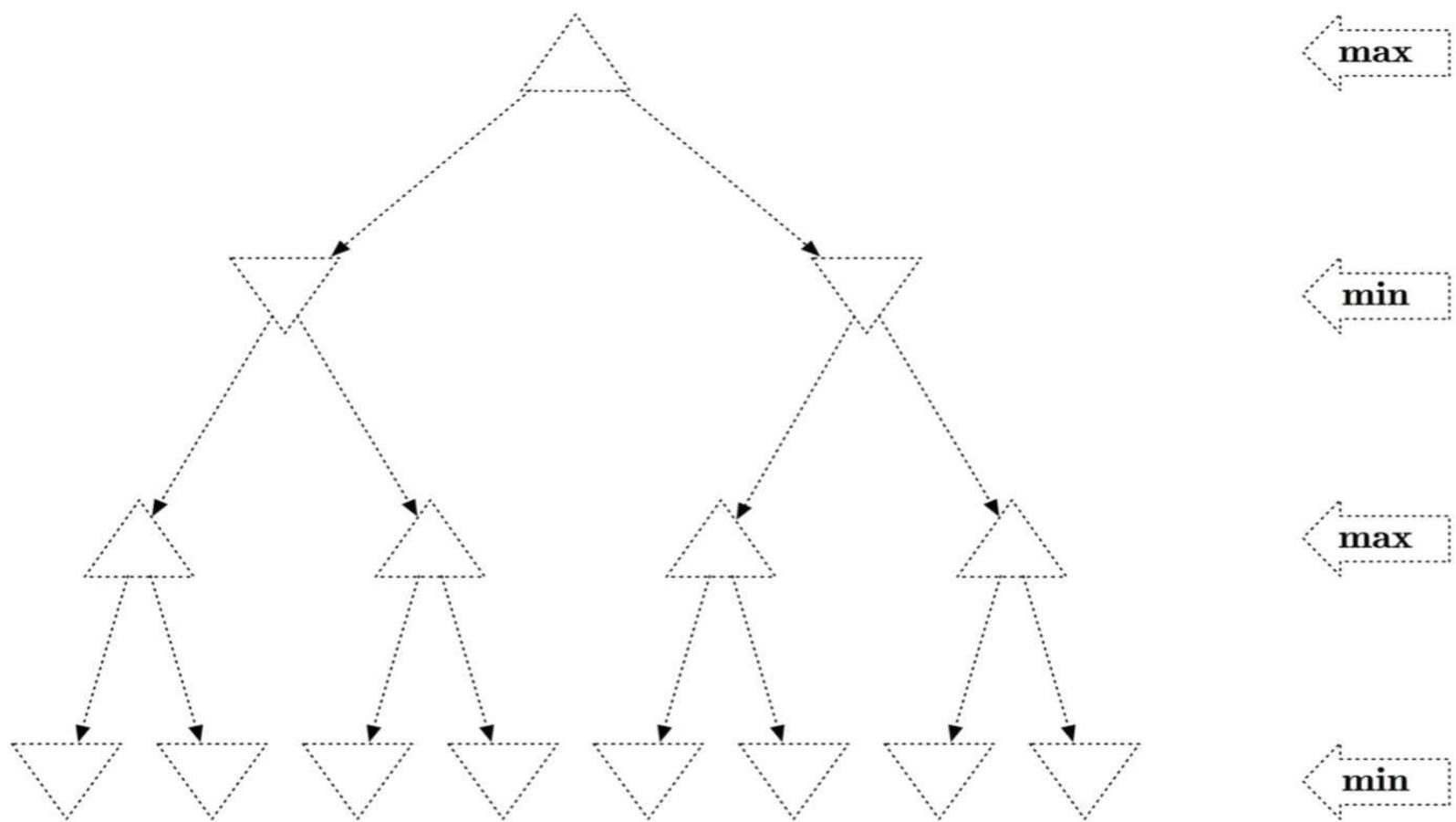
# Chiến lược Minimax

---

- Cách tính điểm cho các đỉnh trên cây trò chơi:
  - Để xác định giá trị các đỉnh có gốc là u, ta đi từ mức thấp nhất đến u.
  - Cây trò chơi sẽ được phân thành các lớp Min-Max xen kẽ lẫn nhau:
    - Lớp Min: lấy giá trị nhỏ nhất của các node con
    - Lớp Max: lấy giá trị lớn nhất của các node con

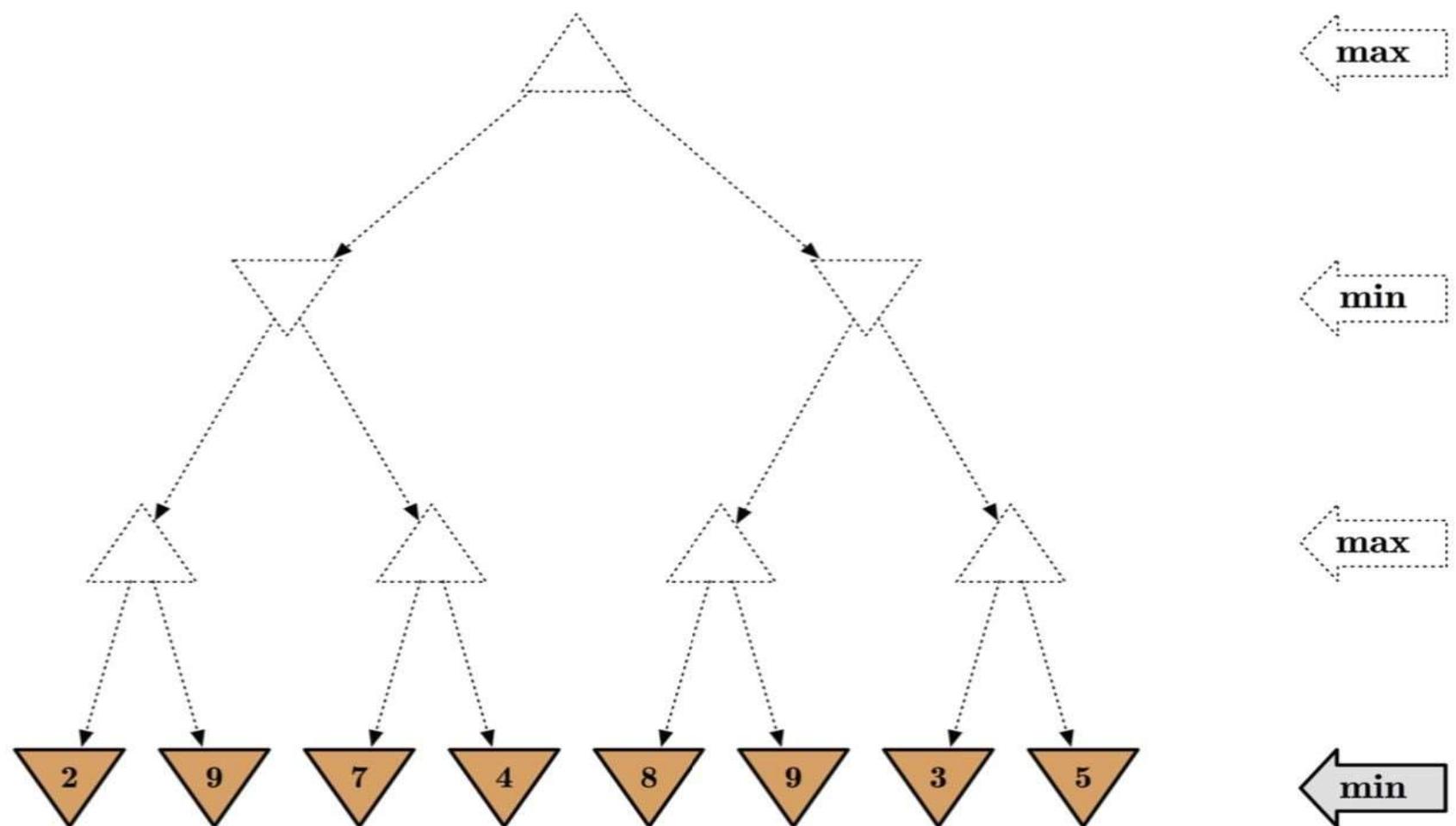
# Minh họa Minimax

---



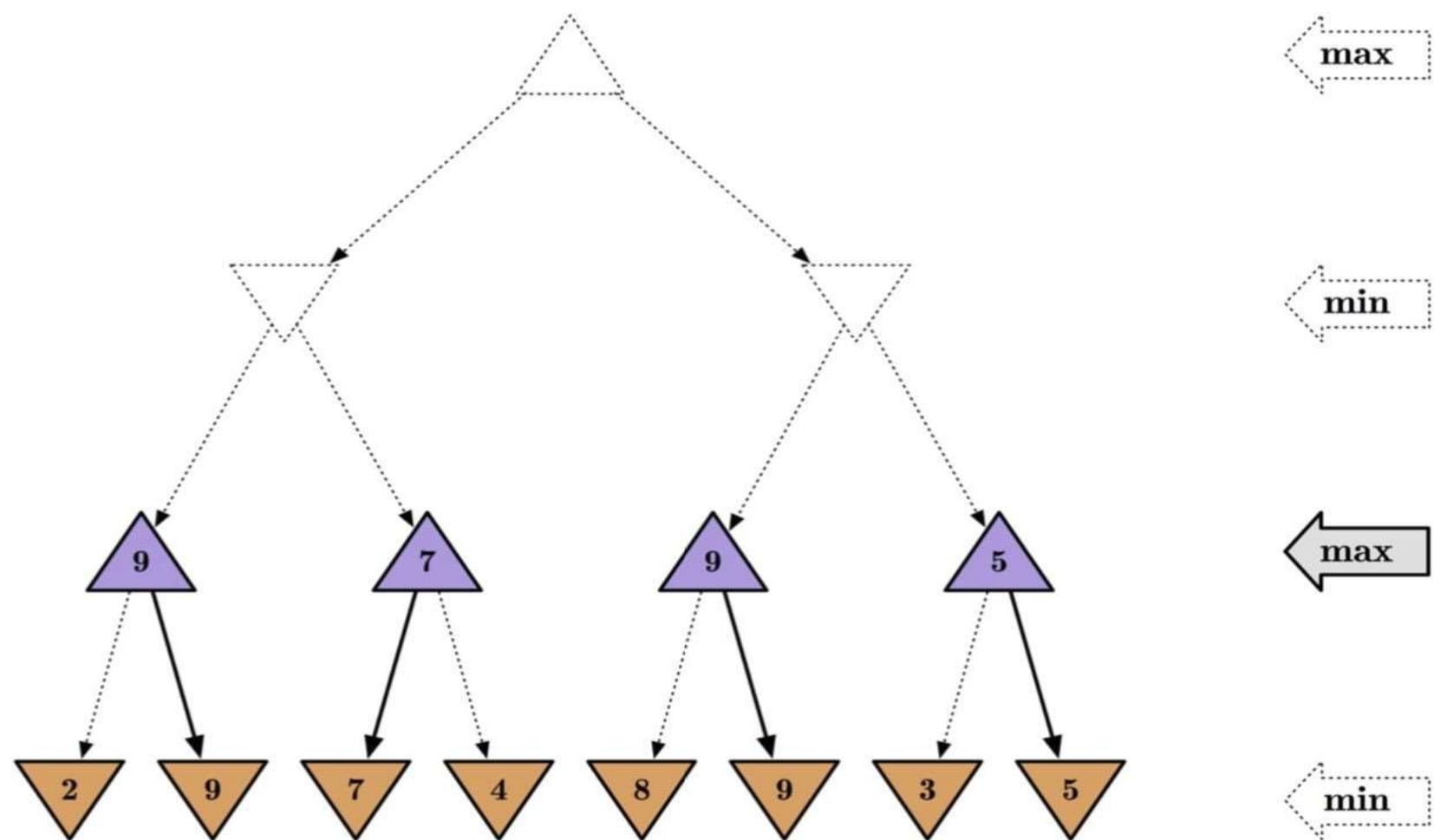
# Minh họa Minimax

---

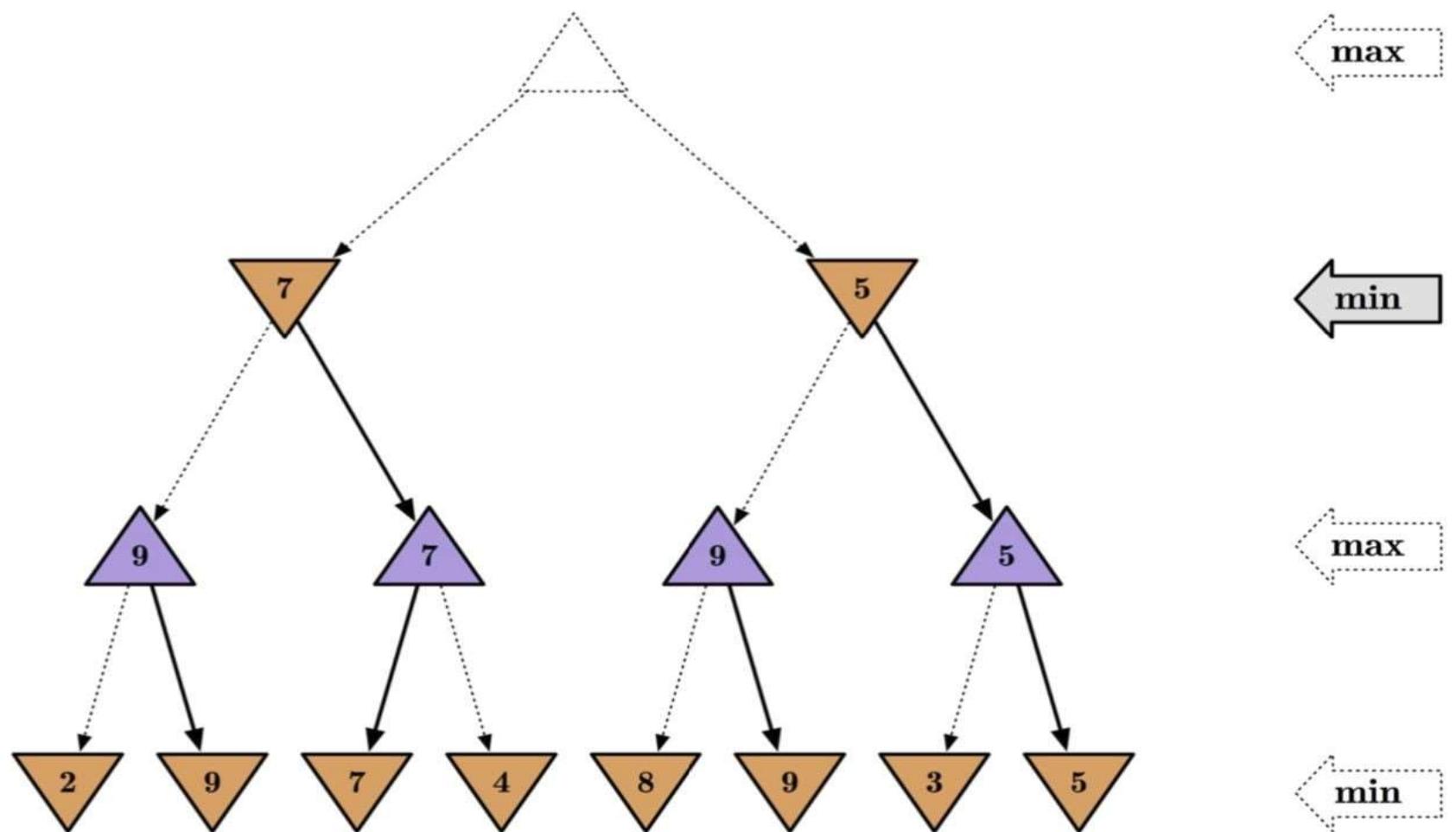


# Minh họa Minimax

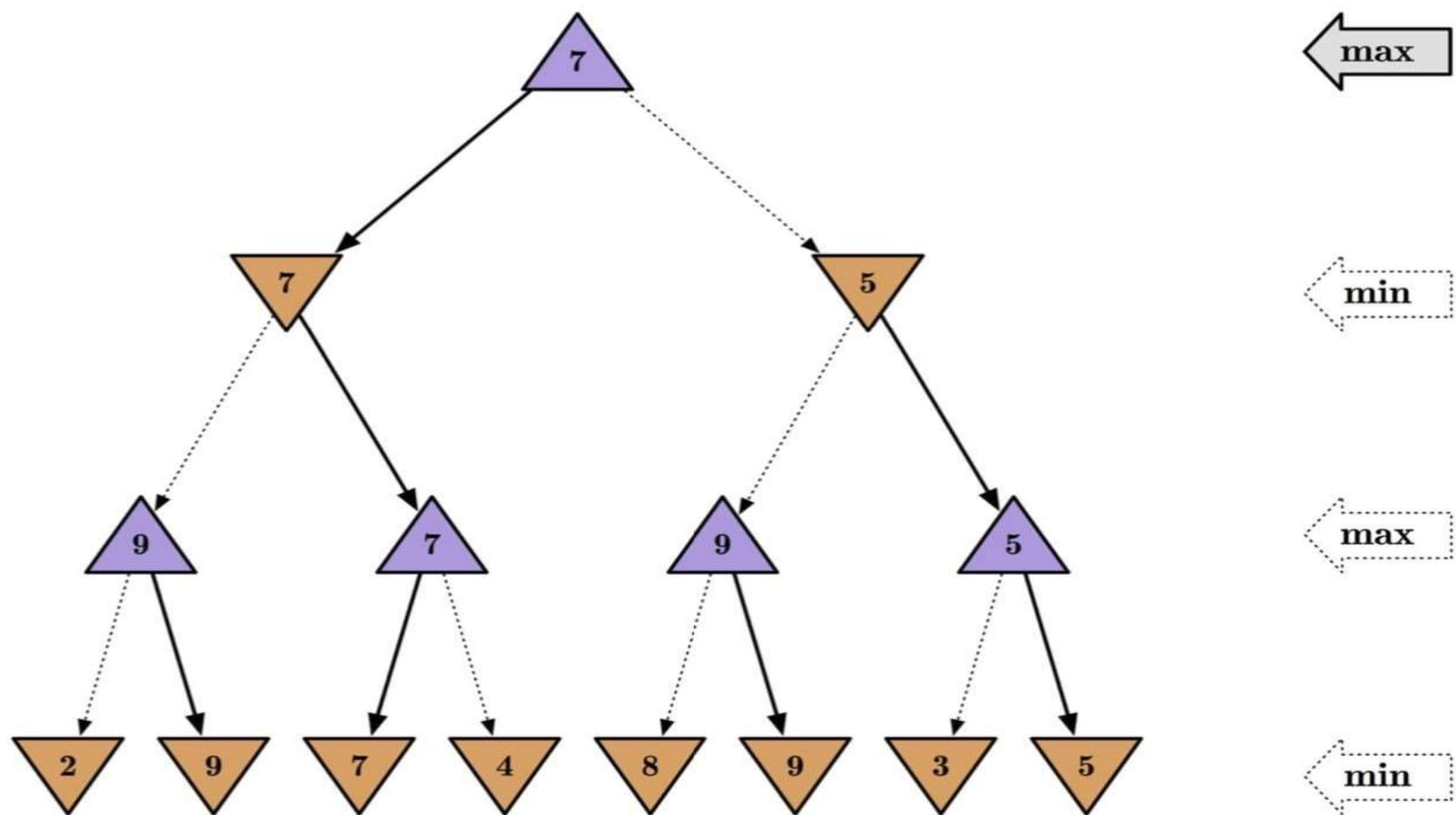
---



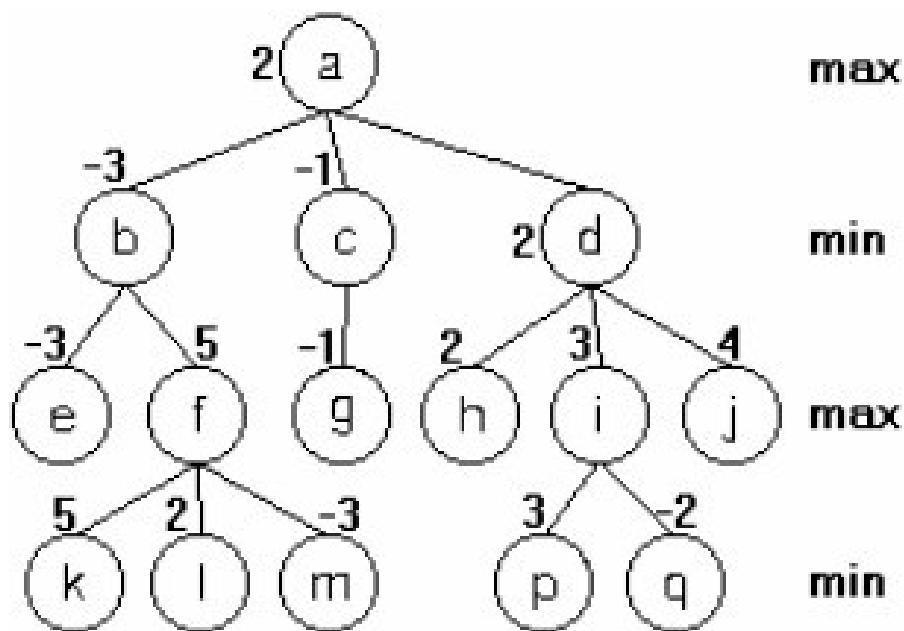
# Minh họa Minimax



# Minh họa Minimax



# Chiến lược Minimax



Gán giá trị cho các đỉnh của cây trò chơi.

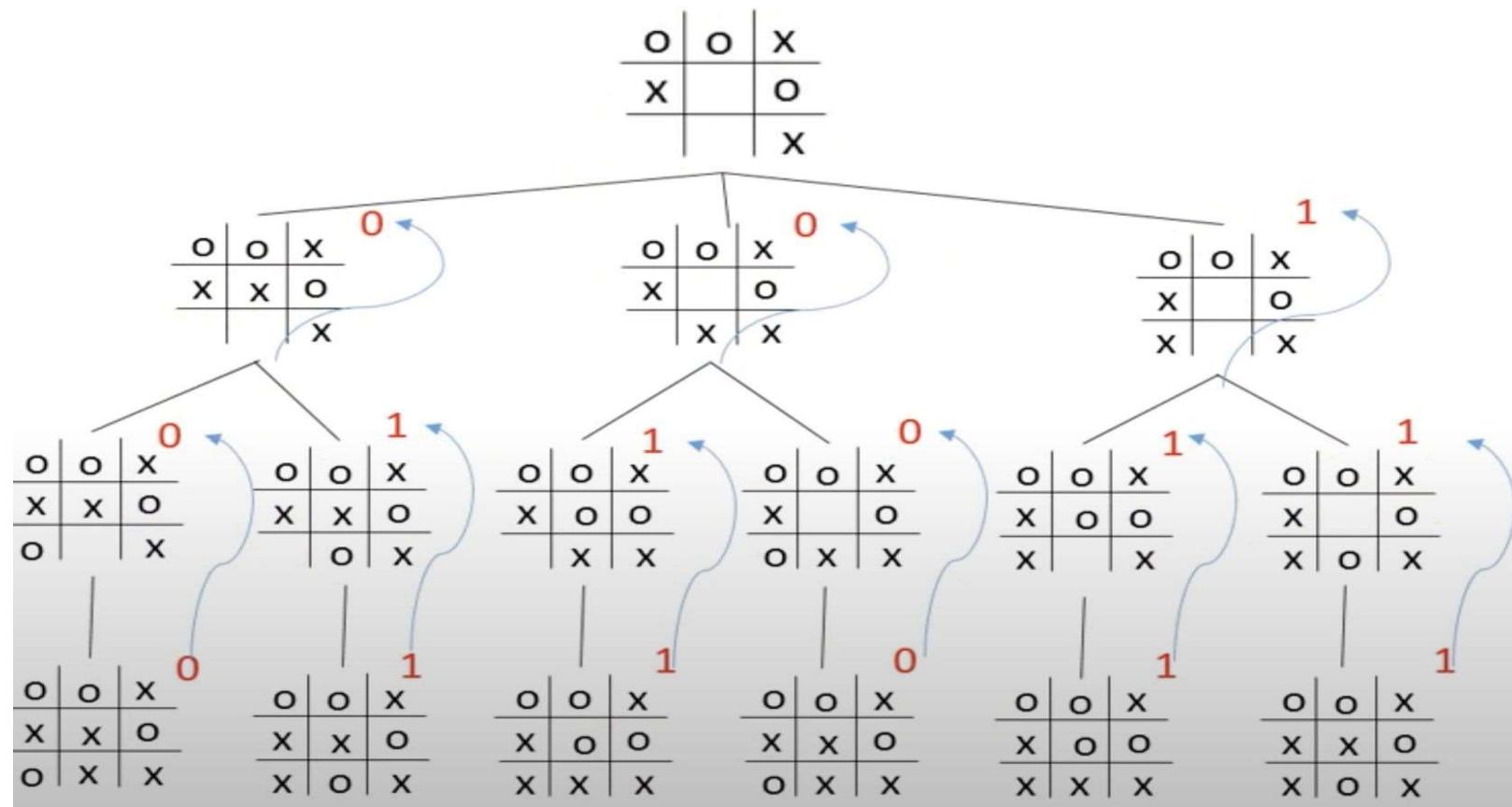
Ví dụ:

đỉnh f là đỉnh Trắng, giá của đỉnh f =  $\max(5, 2, -3) = 5$

đỉnh d là đỉnh Đen, giá của đỉnh d =  $\min(2, 3, 4) = 2$

# Minh họa “Tic Tac Toe”

---



# MiniMax với độ sâu giới hạn

---

- Minimax như đã xét buộc phải có toàn bộ không gian trạng thái đã được triển khai để có thể gán trị cho các nút lá và tính ngược lại  
→ Không khả thi vì không gian bài toán lớn
- Giải pháp: **Giới hạn** không gian trạng thái theo nột **độ sâu** nào đó và giới hạn các **node con** theo một quy tắc nào đó

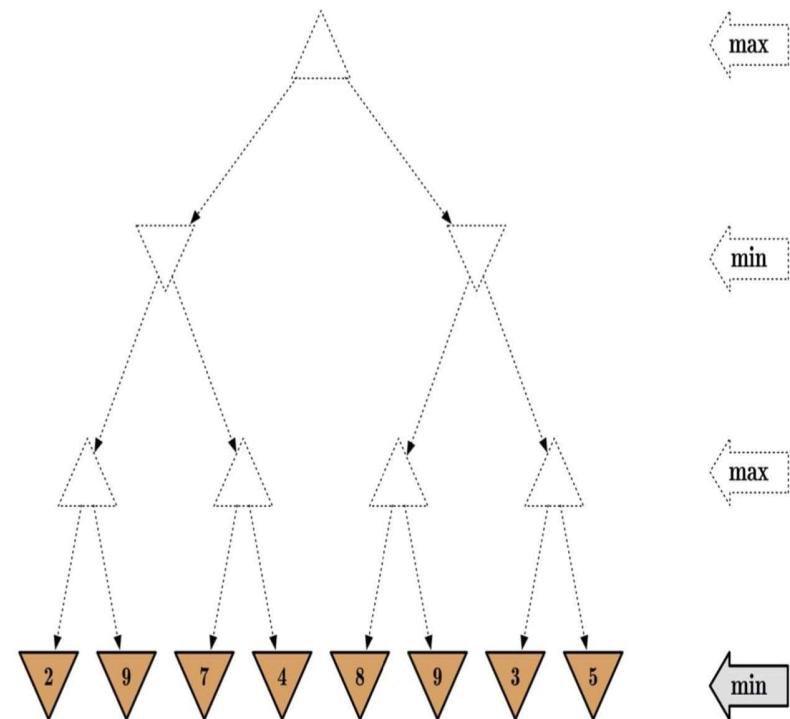
# Thuật toán MiniMax

---

```
function minimax (node, depth, maximizingPlayer)
    if node is “End Node” or depth = 0
        return value(node)
    if maximizingPlayer
        Max=-∞
        foreach child of node
            Max= max(Max, minimax(child, depth-1, False))
        return Max
    else
        Min=∞
        foreach child of node
            Min := min(Min, minimax(child, depth-1, True))
        return Min
```

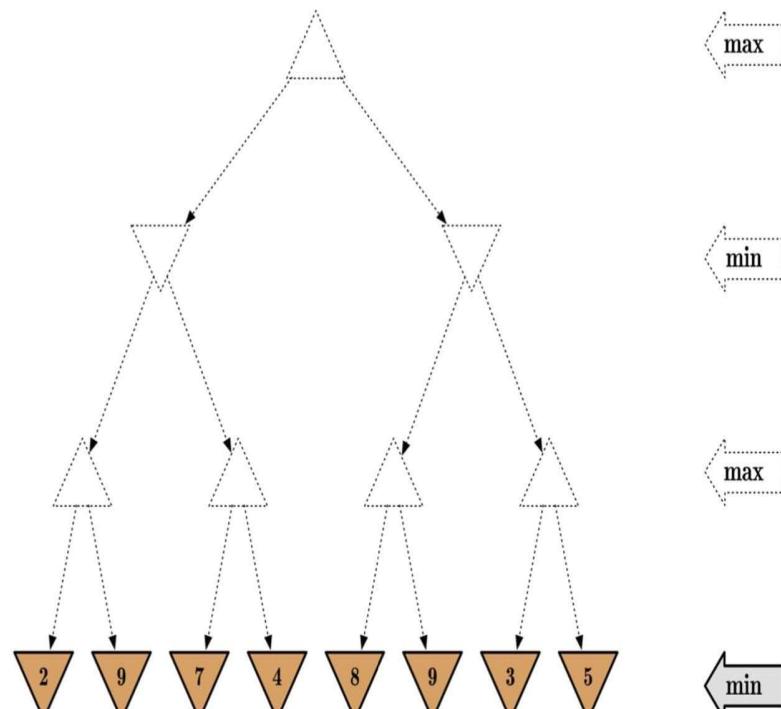
# Demo chiến lược MiniMax (10 phút)

```
class Tree:  
    def __init__(self, data, cost=100000):  
        self.data = data  
        self.cost = cost  
        self.children = []  
        self.parent = None  
    def add_child(self, child):  
        child.parent = self  
        self.children.append(child)  
    def get_data(self):  
        return self.data  
    def get_children(self):  
        return self.children  
    def get_parent(self):  
        return self.parent  
    def __lt__(self, other):  
        return self.cost < other.cost
```



# Demo chiến lược MiniMax (10 phút)

```
if __name__ == "__main__":
    A = Tree("A")           B.add_child(E)
    B = Tree("B")           C.add_child(F)
    C = Tree("C")           C.add_child(G)
    D = Tree("D")           D.add_child(H)
    E = Tree("E")           D.add_child(I)
    F = Tree("F")           E.add_child(J)
    G = Tree("G")           E.add_child(K)
    H = Tree("H")           F.add_child(M)
    I = Tree("I")           F.add_child(N)
    J = Tree("J")           G.add_child(L)
    K = Tree("K")           G.add_child(Z)
    L = Tree("L")           H.value = 2
    M = Tree("M")           I.value = 9
    N = Tree("N")           J.value = 7
    Z = Tree("Z")           K.value = 4
    A.add_child(B)          M.value = 8
    A.add_child(C)          N.value = 9
    B.add_child(D)          L.value = 3
                            Z.value = 5
                            Minimax_Search(A)
                            print(A.value)
```



# Demo chiến lược MiniMax (10 phút)

---

## Chiến lược người chơi Max

```
from TreeNode import Tree  
  
def MaxValue(node):  
    if len(node.children) == 0:  
        return node  
    node.value = -100000  
    for child in node.children:  
        temp = MinValue(child)  
        if temp.value > node.value:  
            node.value = temp.value  
    return node
```

## Chiến lược cho người chơi cho Min

```
def MinValue(node):  
    if len(node.children) == 0:  
        return node  
    node.value = 100000  
    for child in node.children:  
        temp = MaxValue(child)  
        if temp.value < node.value:  
            node.value = temp.value  
    return node
```

## Chiến lược Max

```
def Minimax_Search(state):  
    MaxValue(state)
```

# Chiến lược Minimax

---

- Thành công của Minimax
  - Cần xác định giá trị của tất cả các nút lá trên cây trò chơi
  - Để chọn trạng thái “tốt nhất” thì phải tính ngược từ nút lá đến trạng thái đó.
- Vấn đề: trong trò chơi thực tế, chúng ta bị hạn chế về mặt thời gian nên không thể tìm kiếm tất cả các lá
  - Để thiết thực và chạy trong một khoảng thời gian nhất định, Minimax chỉ tìm kiếm theo giới hạn độ sâu hoặc giới hạn các node theo một quy tắc nào đó.
  - Tạo nên sự khác biệt

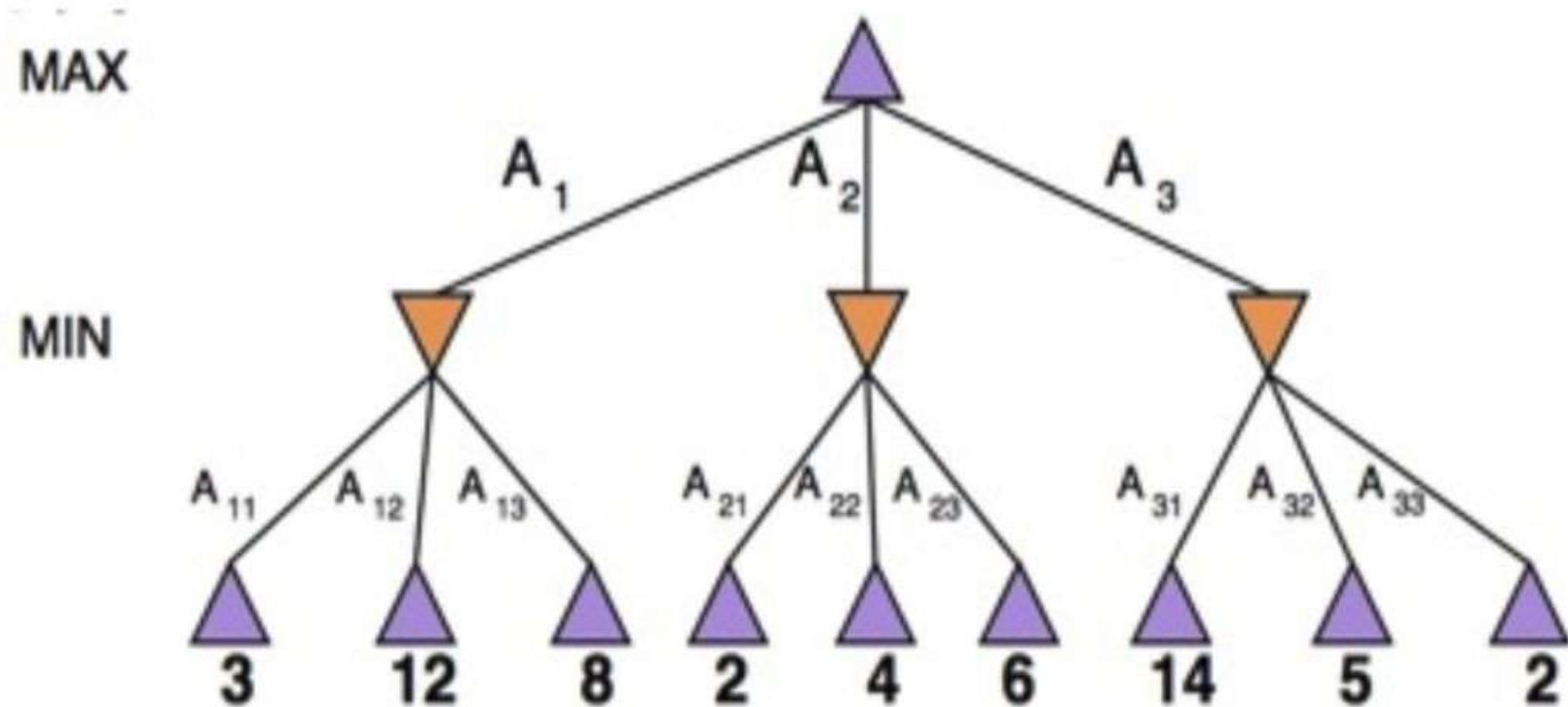
# Chiến lược Minimax

---

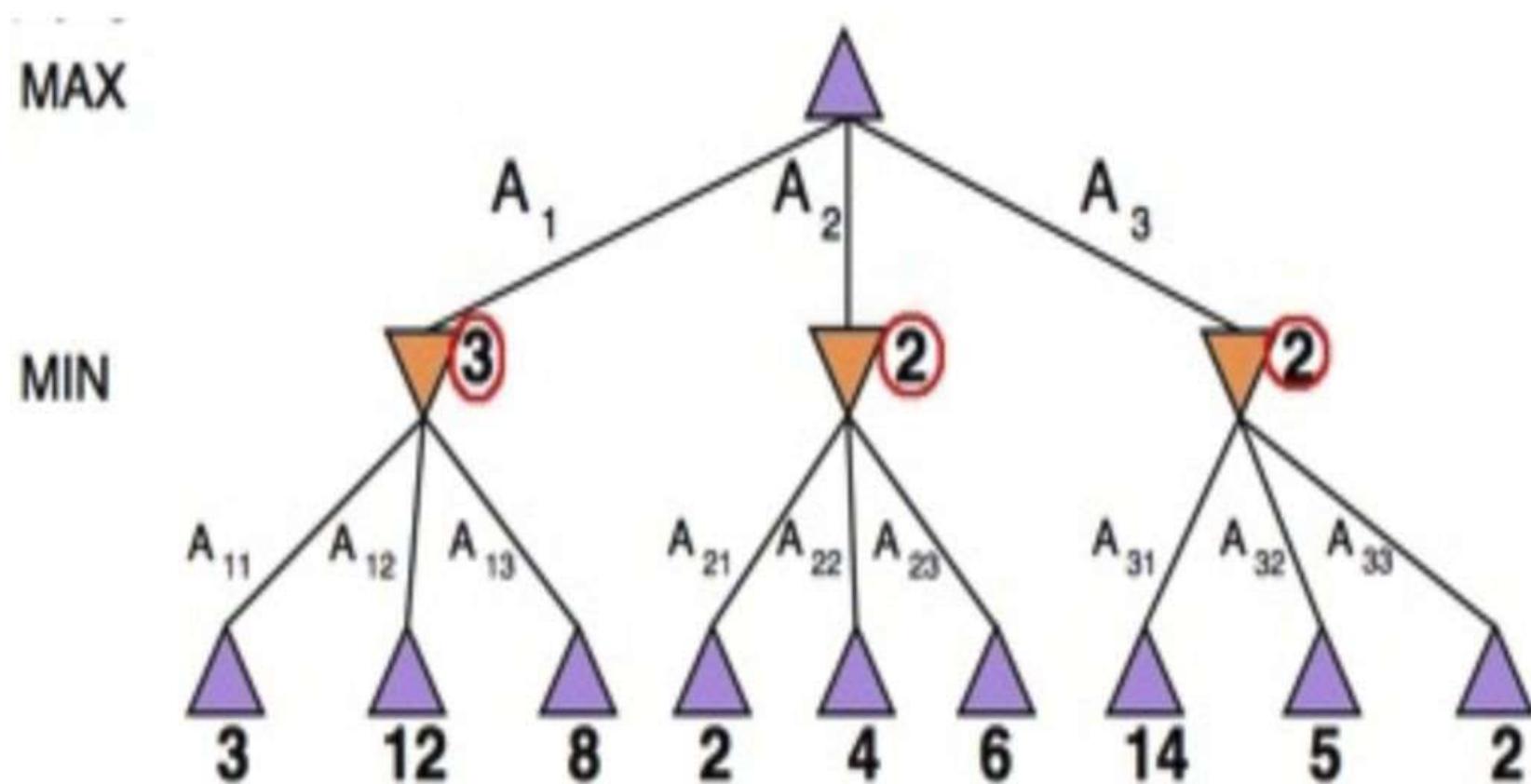
- Trong việc tìm kiếm Minimax, cho dù giới hạn độ sâu thì số đỉnh con của một đỉnh bất kỳ vẫn rất lớn trong thực tế
- Ví dụ:
  - Cờ vua: số nhánh trung bình khoảng 35.
  - Vậy nếu cần suy nghĩ độ sâu là 4 thì phải mất ít nhất  $35^4 = 1.5 \cdot 10^6$  số phép đánh giá.
- → Cần một phương pháp **làm giảm số nhánh**
- **Giải pháp: phương pháp cắt tỉa alpha-beta**
  - Mục đích làm giảm số nhánh trong cây tìm kiếm và
  - Không làm ảnh hưởng đến sự đánh giá của đỉnh đang xét.
  - Loại bỏ những phần lớn các nhánh của cây.

# Chiến lược cắt tỉa Alpha-Beta

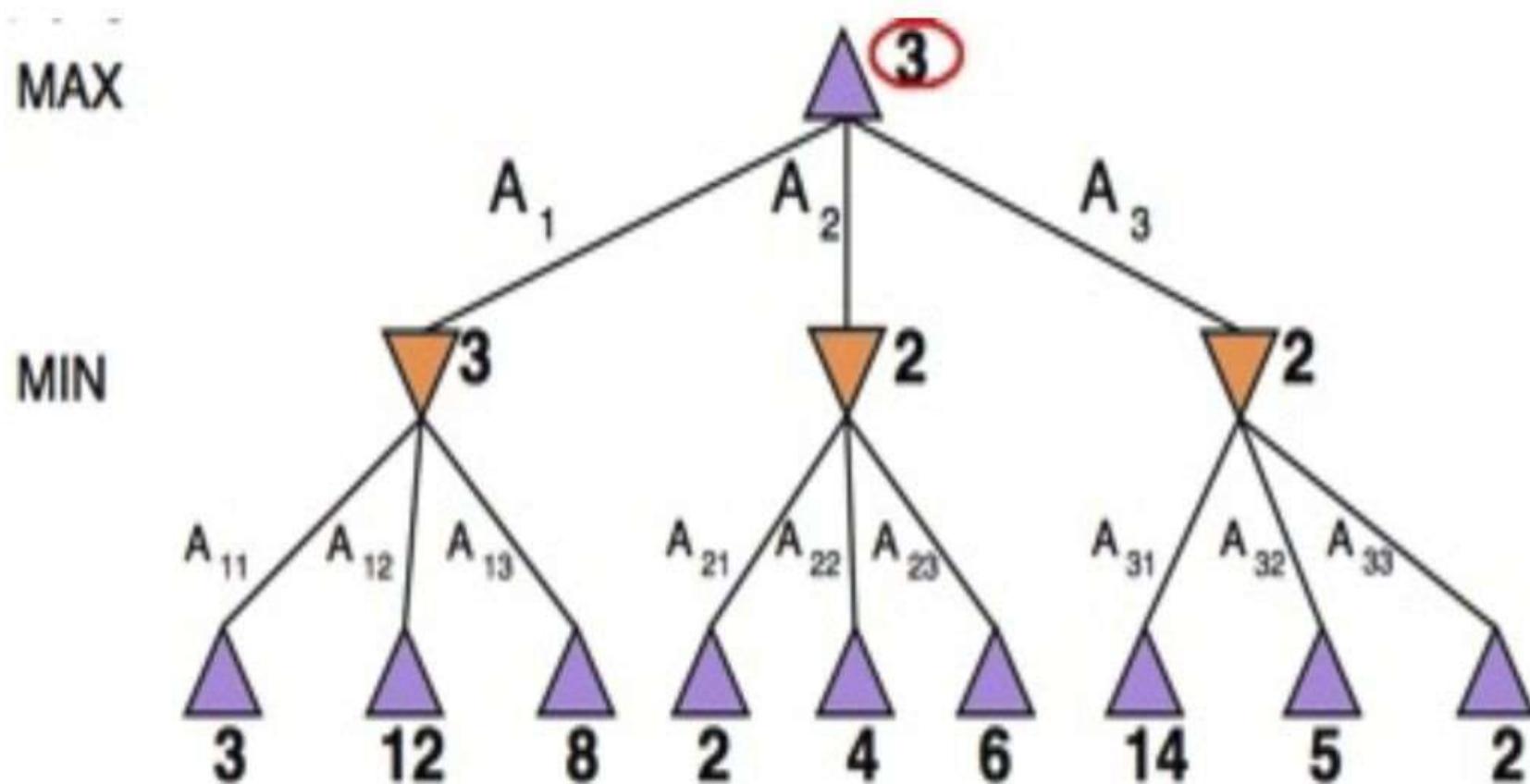
---



# Chiến lược cắt tỉa Alpha-Beta



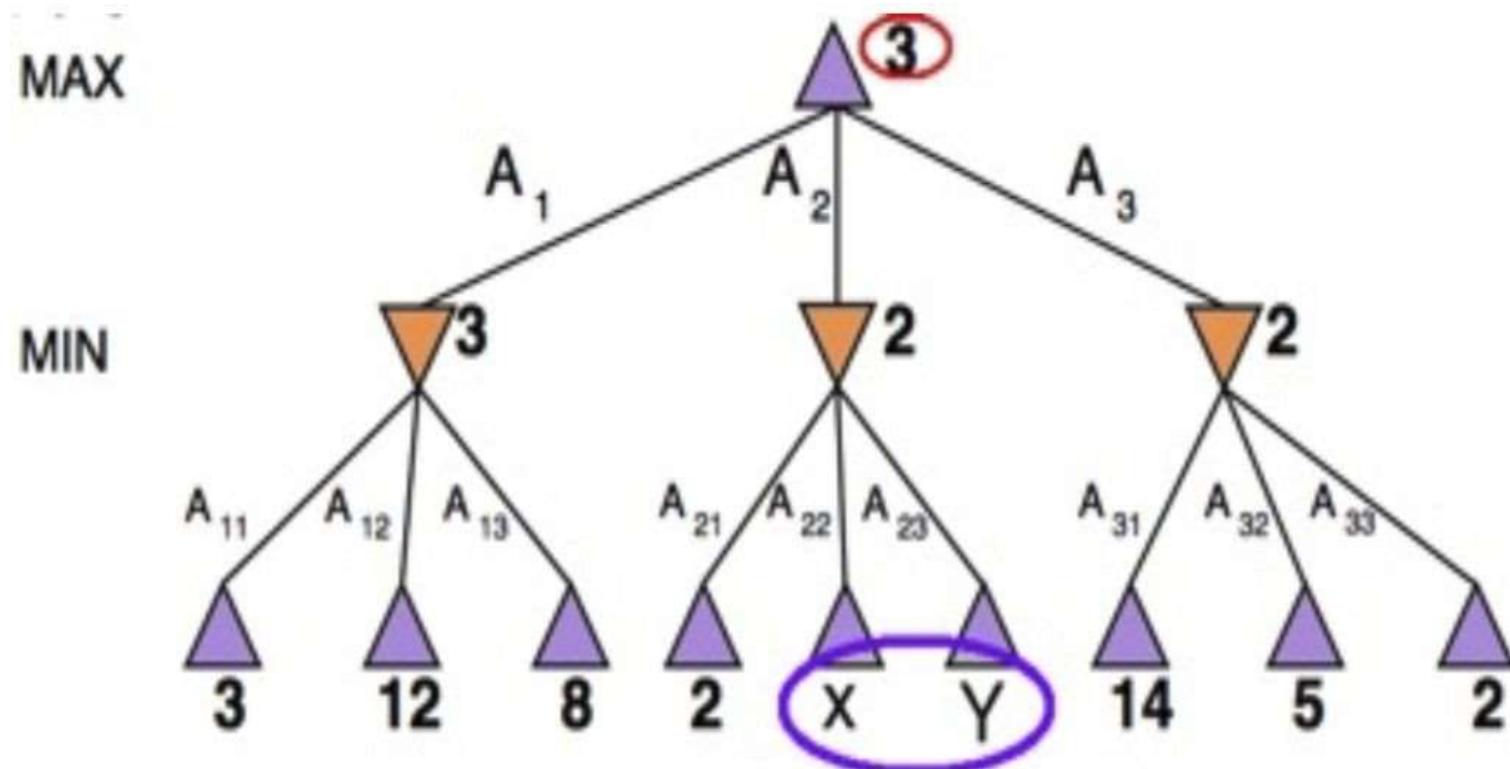
# Chiến lược cắt tỉa Alpha-Beta



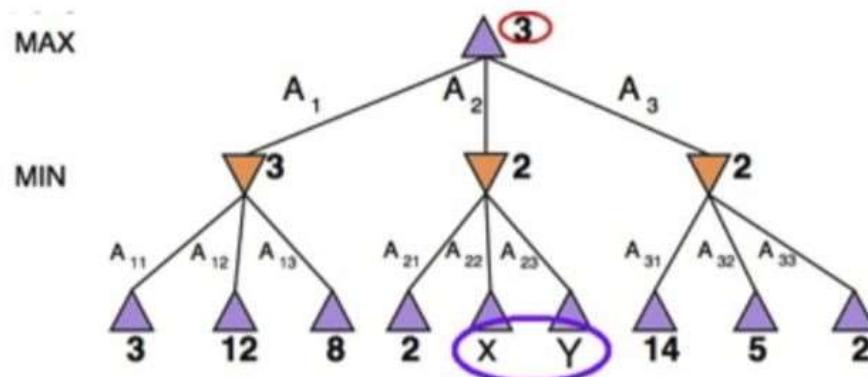
# Chiến lược cắt tỉa Alpha-Beta

---

- Nên cắt bỏ nhánh không cần thiết



# Chiến lược cắt tỉa Alpha-Beta

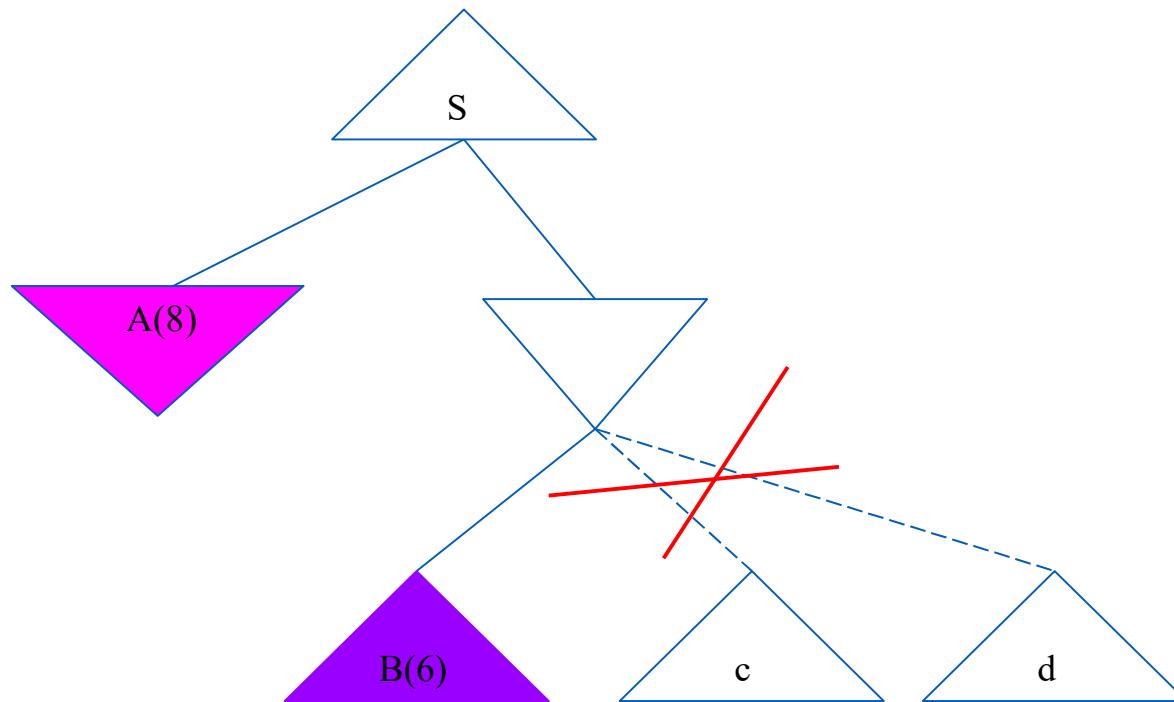


$$\begin{aligned} \text{Minimax}(\text{root}) &= \max(\min(3, 12, 8), \min(2, X, Y), \min(14, 5, 2)) \\ &= \max(3, \min(2, X, Y), 2) \\ &= \max(3, Z, 2) \quad \text{where } Z = \min(2, X, Y) \leq 2 \\ &= 3 \end{aligned}$$

# Chiến lược cắt tỉa Alpha-Beta

---

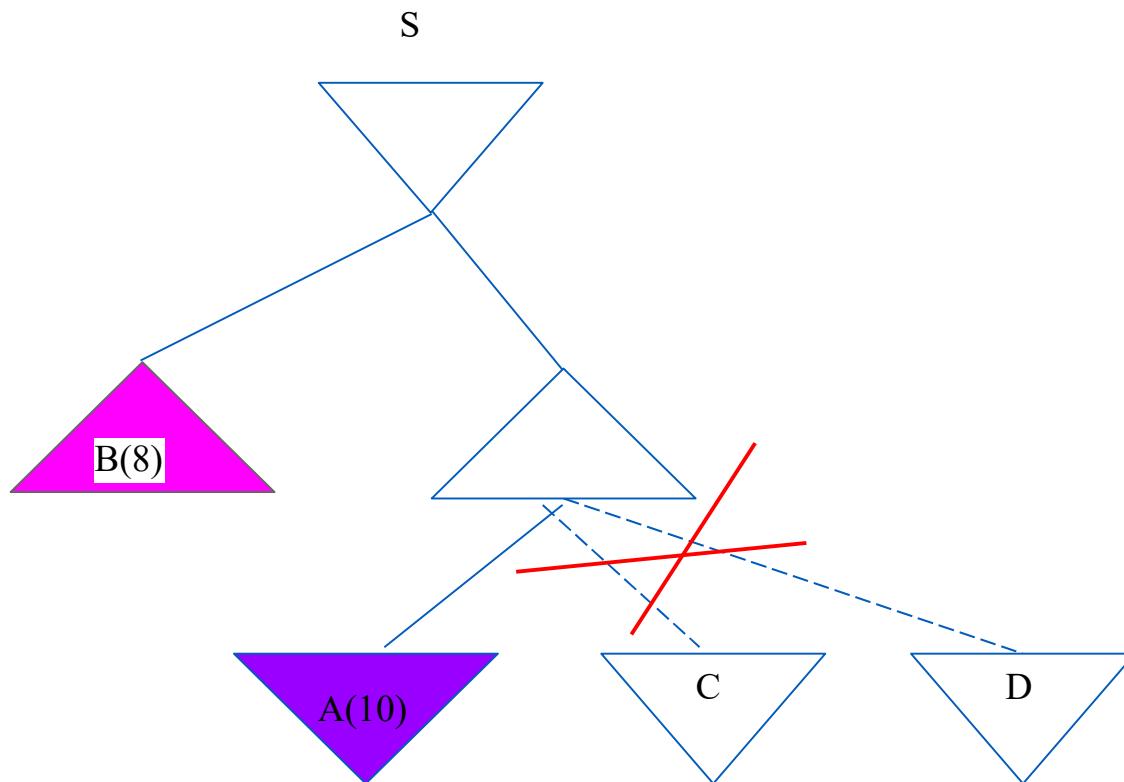
- Tham số Alpha: giá trị lớn nhất trong các đỉnh con của tầng Max



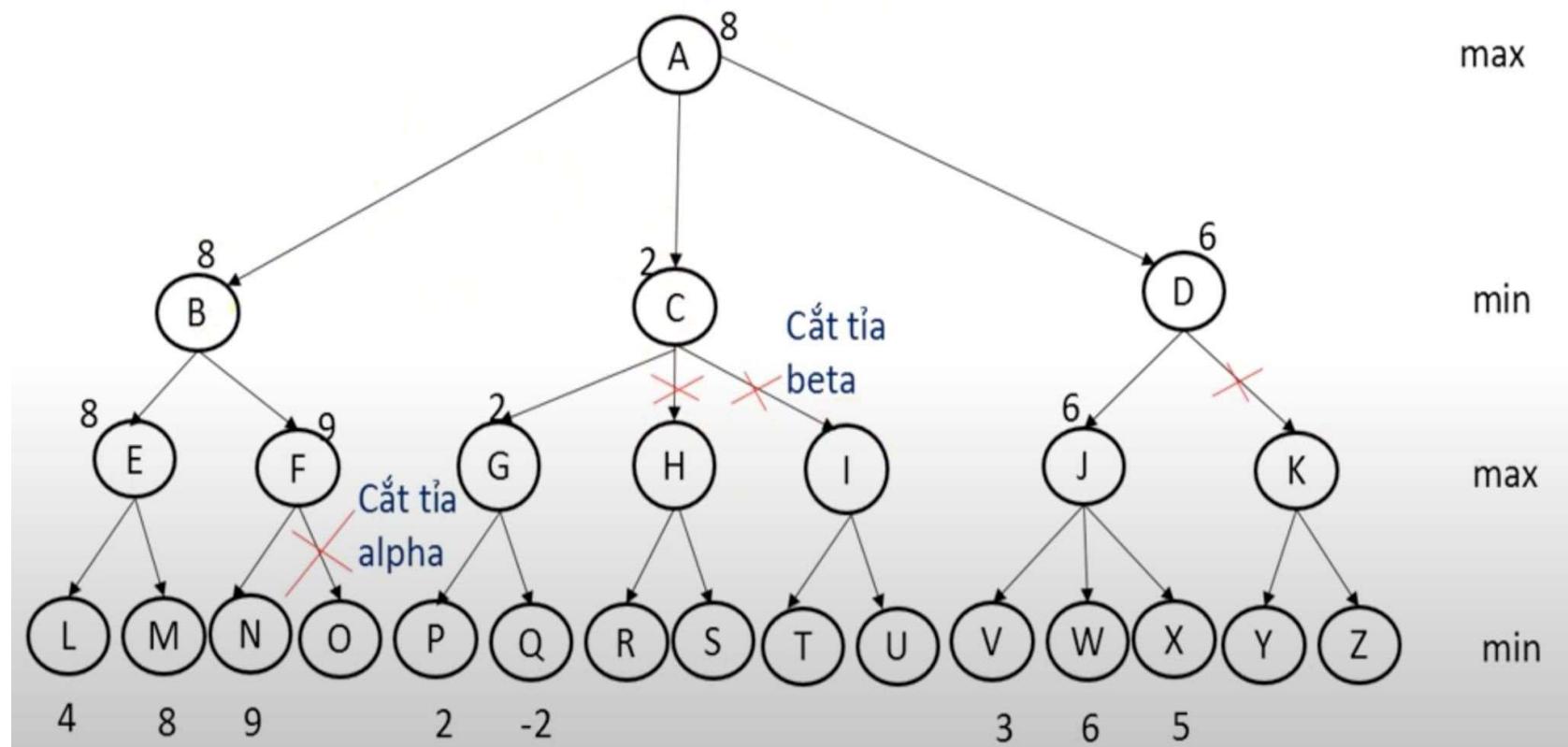
# Chiến lược cắt tia Alpha-Beta

---

- Tham số Beta- giá trị nhỏ nhất trong các đỉnh con của tầng Min



# Minh họa cắt tỉa alpha-beta



# Cắt tỉa Alpha beta- Thuật Toán 1

```
function minimax (node, depth, maximizingPlayer)
    return alphabeta(node, depth, -  $\infty$ ,  $\infty$ ,maximizingPlayer)
```

```
function alphabeta (node, depth, a, b, maximizingPlayer)
    if node is “End Node” or depth = 0
        return value(node)
    if maximizingPlayer
        foreach child of node
            a= max(a, alphabeta (child, depth-1, a, b, False))
            if a $\geq$ b break;
        return a
    else
        foreach child of node
            b := min(b, alphabeta (child, depth-1, a, b, True))
            if a $\geq$ b break;
        return b
```

# Cắt tỉa Alpha beta – Thuật Toán 2

```
function minimax (node, depth)
    return alphabeta(node, depth, -∞, ∞)
```

```
function alphabeta (node, depth, a, b)
    if node is “End Node” or depth = 0
        return value(node)
    foreach child of node
        a= max(a, -alphabeta (child, depth-1, -b, -a))
        if a>=b break;
    return a
```

# Demo cắt tỉa alpha-beta

---

```
1 from TreeNode import Tree
2 def MaxValue(node, alpha, beta):
3     if len(node.children) == 0:
4         return node
5     node.value = -100000
6     for child in node.children:
7         temp = MinValue(child, alpha, beta)
8         if temp.value > node.value:
9             node.value = temp.value
10        if child.value >= beta:
11            return child
12        if child.value > alpha:
13            alpha = child.value
14    return node
```

# Demo cắt tỉa alpha-beta

---

```
15 def MinValue(node, alpha, beta):
16     if len(node.children) == 0:
17         return node
18     node.value = 100000
19     for child in node.children:
20         temp = MaxValue(child, alpha, beta)
21         if temp.value < node.value:
22             node.value = temp.value
23         if child.value <= alpha:
24             return child
25         if child.value < beta:
26             beta = child.value
27     return node
28
29 def Alpha_Beta_Search(state):
30     MaxValue(state, -100000, 100000)
31     . .
32     . .
33     . .
```

# Demo cắt tỉa alpha-beta

---

```
30 if __name__ == "__main__":
31     A = Tree("A")
32     B = Tree("B")
33     C = Tree("C")
34     D = Tree("D")
35     E = Tree("E")
36     F = Tree("F")
37     G = Tree("G")
38     H = Tree("H")
39     I = Tree("I")
40     J = Tree("J")
41     K = Tree("K")
42     L = Tree("L")
43     M = Tree("M")
44     N = Tree("N")
45     Z = Tree("Z")
46     A.add_child(B)
47     A.add_child(C)
48     B.add_child(D)
49     B.add_child(E)
50     C.add_child(F)
51     C.add_child(G)
52     D.add_child(H)
53     D.add_child(I)
54     E.add_child(J)
55     E.add_child(K)
56     F.add_child(M)
57     F.add_child(N)
58     G.add_child(L)
59     G.add_child(Z)
60     H.value = 2
61     I.value = 9
62     J.value = 7
63     K.value = 4
64     M.value = 8
65     N.value = 9
66     L.value = 3
67     Z.value = 5
68     Alpha_Beta_Search(A)
69     print(A.value)
```