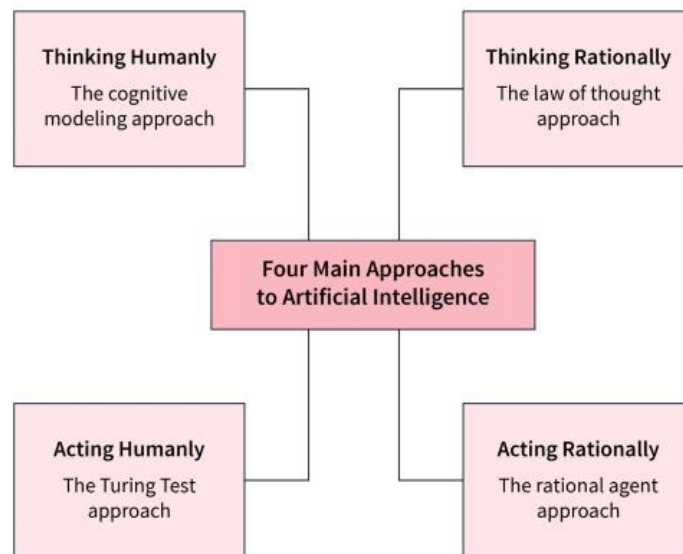


ĐỀ CƯƠNG ÔN TẬP THI CUỐI KỲ

I. *Khái niệm A.I.*

- A.I là lý thuyết và là sự phát triển của hệ thống máy tính có khả năng thực hiện các nhiệm vụ thông thường đòi hỏi sự can thiệp của trí thông minh của con người, chẳng hạn như thị giác nhận thức, nhận diện giọng nói, đưa ra quyết định và dịch ngôn ngữ, v.v..
- A.I có thể được chia thành 4 loại như sau:
 1. Hệ thống suy nghĩ (thông minh) như con người.
 2. Hệ thống suy nghĩ hợp lý.
 3. Hệ thống hành động (thông minh) như con người.
 4. Hệ thống hành động hợp lý.

A.I can be separated into four categories as following:



- Định nghĩa (1) và (2) liên quan tới các quá trình suy nghĩ và suy diễn.
- Định nghĩa (3) và (4) liên quan tới cách hành động.

- Định nghĩa (1) và (3) đánh giá mức độ thành công (sự thông minh) theo tiêu chuẩn con người.
- Định nghĩa (2) và (4) đánh giá mức độ thành công (sự thông minh) theo tiêu chuẩn của sự hợp lý.

1. *Hành động như con người (Acting humanly: Turing test).*

- Máy tính cần có những tính năng sau:
 - Xử lý ngôn ngữ tự nhiên: để giao tiếp thành công qua tiếng Anh.
 - Biểu diễn tri thức: để lưu trữ những gì nó biết và nghe được.
 - Lý luận tự động: sử dụng tri thức, thông tin được lưu trữ để trả lời câu hỏi và rút ra kết luận mới.
 - Học máy: để thích nghi với hoàn cảnh mới và phát hiện, suy diễn các mẫu mới.
- Để vượt qua bài kiểm tra Turing test, máy tính cần:
 - Thị giác máy tính: nhận biết đồ vật.
 - Robot: thao tác với đồ vật và di chuyển.
- Turing đã đề xuất các thành phần quan trọng của A.I như sau:
 - Tri thức.
 - Suy diễn.
 - Hiểu ngôn ngữ.
 - Học máy.

2. *Suy nghĩ như con người: mô hình nhận thức (Thinking humanly: cognitive modeling).*

- Đòi hỏi những lý thuyết khoa học về hoạt động bên trong của não:
 - Dự đoán và kiểm tra hành vi của đối tượng con người (nhận thức khoa học).
 - Nhận dạng trực tiếp từ dữ liệu thần kinh (nhận thức khoa học thần kinh).

3. *Suy nghĩ hợp lý: các luật suy nghĩ.*

- Aristotle đã cố gắng hệ thống hóa “tư duy đúng đắn” như sau:
 - Sử dụng “luật tư duy” để đưa ra kết luận đúng đắn bằng logic.
 - Ví dụ: “Socrates là một người đàn ông, tất cả đàn ông đều phải chết; do đó Socrates phải chết.
- Mối liên hệ giữa Toán học và Triết học dẫn tới A.I.
- Tuy nhiên, chúng ta sẽ có các trở ngại:
 - Không phải mọi hành vi thông minh có thể được mô tả dưới dạng logic, ký hiệu.
 - Có sự khác biệt lớn giữa khả năng giải quyết vấn đề “trong nguyên tắc” và thực hiện điều đó trong thực tế.

4. *Hành động hợp lý: tác nhân hợp lý (Acting rationally: rational agent).*

- Agent hợp lý là Agent có thể hành động để đạt được kết quả tốt nhất ngay cả khi không có sự chắc chắn.
- Hành vi hợp lý: làm điều đúng đắn.
- Không nhất thiết bao gồm suy nghĩ nhưng suy nghĩ nên là một phần để phục vụ cho hành động hợp lý.

5. Tác nhân hợp lý: *Rational Agents*.

- Một tác nhân (Agent) là một thực thể có khả năng nhận thức và hành động.
- Một cách khái quát, một tác tử có thể được biểu diễn dưới một hàm ánh xạ: từ quá trình (lịch sử) nhận thức đến hành động: $f: P^* \rightarrow A$.
- Đối với một tập (lớp) các môi trường và nhiệm vụ, chúng ta cần tìm ra Agent (hoặc một lớp các Agent) có hiệu suất tốt nhất.
- *Lưu ý*: Các giới hạn về tính toán (của máy tính) không cho phép đạt được sự hợp lý hoàn hảo (tối ưu). → Mục tiêu: thiết kế chương trình máy tính tối ưu đối với các tài nguyên máy tính hiện có.

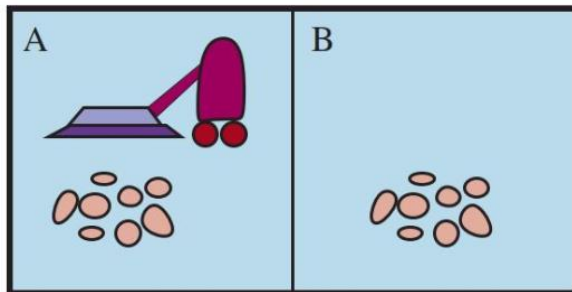
II. Tác nhân thông minh (*Intelligent Agents*).

1. Agents and Environments.

- Agents là bất cứ cái gì (con người, người máy, software robots, các bộ ổn nhiệt, v.v...) có khả năng *cảm nhận* (*nhận biết*) *môi trường* xung quanh nó thông qua *các bộ phận cảm biến* (*sensors*) và *hành động* phù hợp theo môi trường đó thông qua *các bộ phận hoạt động* (*actuators*).
- Human Agents:
 - o Các bộ phận cảm biến: tai, mắt, da và một số cơ quan nội tạng khác.
 - o Các bộ phận hoạt động: tay, chân, miệng và một số bộ phận khác.
- Robot Agents:
 - o Các bộ phận cảm biến: các máy quay, bộ phận cảm biến tín hiệu hồng ngoại.
 - o Các bộ phận hoạt động: các loại động cơ (motors).
- Hàm Agents: là hàm ánh xạ từ lịch sử nhận thức tới các hành động:

$$f: P^* \rightarrow A$$

- Chương trình Agents: hoạt động (chạy) dựa trên kiến trúc thực tế của hàm f :
 - o Agents = Kiến trúc + Chương trình.
- Ví dụ: Thế giới của máy hút bụi.



- Các nhận thức: vị trí và mức độ sạch sẽ. Ví dụ: [A, Bẩn], [B, Sạch].
- Các hành động: Di chuyển (máy hút bụi) sang trái, sang phải, hút bụi hoặc không làm gì cả.
- Hàm Agents:

- Nếu ô hiện tại bị bẩn, di chuyển sang ô bên phải và hút bụi, nếu không thì di chuyển sang ô vuông khác.

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

2. Rational Agents.

- Agents cần phải cố gắng đạt hiệu suất tốt để “làm đúng việc cần làm”, dựa trên những gì nó nhận thức (nhận biết) được và dựa trên những hành động mà nó có thể thực hiện.
- Một hành động đúng (hợp lý) là hành động giúp cho Agents đạt được thành công cao nhất đối với mục tiêu đặt ra.
- Đánh giá hiệu quả hoạt động: là tiêu chuẩn để đánh giá mức độ thành công trong hoạt động của một Agents.
 - Ví dụ: Tiêu chí đánh giá hiệu quả hoạt động của một Agent máy hút bụi có thể là: mức độ làm sạch, thời gian hút bụi, mức độ điện năng tiêu thụ, mức độ tiếng ồn gây ra, v.v...
- Nên thiết kế thước đo hiệu suất theo những gì người ta thực sự mong muốn ở môi trường chứ không phải cách một Agent thể hiện như thế nào.
- Với một chuỗi nhận thức có được, một Agent hợp lý cần phải lựa chọn một hành động giúp cực đại hóa tiêu chí đánh giá hiệu quả hoạt động của Agent đó. Dựa trên các thông tin được cung cấp bởi chuỗi nhận thức và các tri thức được sở hữu bởi các Agent đó.
- Sự hợp lý ≠ Sự thông suốt mọi thứ.
 - Sự thông suốt mọi thứ = Biết tất cả mọi thứ, với tri thức vô hạn.
 - Vì các nhận thức có thể không cung cấp tất cả các thông tin liên quan.
- Các Agent có thể thực hiện các hành động nhằm thay đổi các nhận thức trong tương lai, với mục đích thu được các thông tin hữu ích (ví dụ: thu thập thông tin, khám phá tri thức).
- Agent có tính tự trị là một Agent mà các hành động của nó được quyết định bởi chính kinh nghiệm của Agent đó (cùng với khả năng học và thích nghi).

III. Môi trường công việc – PEAS

1. PEAS:

- Performance measure: Tiêu chí đánh giá hiệu quả hoạt động.
- Environment: Môi trường xung quanh.
- Actuators: Các bộ phận hành động.
- Sensors: Các bộ phận cảm biến.
- Để thiết kế một Agent thông minh (hợp lý), trước tiên cần phải xác định (thiết lập) các giá trị của các thành phần của PEAS.
- Ví dụ: ***Design an Automated taxi driver.***
 - Performance measure: an toàn, nhanh, uy tín, chuyên đi thoải mái, tối ưu lợi nhuận, mức độ hài lòng của khách, v.v...
 - Environment: các con đường, các phương tiện khác, khách hàng, cảnh sát, người đi bộ, v.v...
 - Actuators: bánh lái, chân ga, phanh, đèn tín hiệu, còi xe, v.v...
 - Sensors: máy quay, đồng hồ đo tốc độ, GPS, các bộ cảm biến động cơ, v.v...
- Ví dụ: ***Medical Diagnosis System.***
 - Performance measure: mức độ sức khỏe của các bệnh nhân, tối ưu chi phí khám bệnh, các vụ kiện cáo, v.v...
 - Environment: bệnh nhân, bệnh viện, bác sĩ, nhân viên y tế, v.v...
 - Actuators: hiển thị trên màn hình các câu hỏi, các xét nghiệm, chẩn đoán, các điều trị, chỉ dẫn, v.v...
 - Sensors: bàn phím để nhập vào các thông tin về triệu chứng, các trả lời của bệnh nhân đối với các câu hỏi, v.v...
- Ví dụ: ***Part-picking robots.***
 - Performance measure: tỷ lệ phần trăm các đồ vật được đặt vào đúng thùng các thùng yêu cầu.
 - Environment: dây chuyền chuyển động, các đồ vật, các thùng.
 - Actuators: cánh tay và bàn tay được kết nối.
 - Sensors: máy quay, các bộ cảm biến ở các góc độ (các hướng).
- Ví dụ: ***Interactive English tutor:***
 - Performance measure: tối đa hóa số điểm thi tiếng Anh của học viên.
 - Environment: một nhóm học viên.
 - Actuators: hiển thị màn hình (bài tập, gợi ý và sửa bài tập).
 - Sensors: bàn phím.
- Ví dụ: ***Spam emails filtering.***
 - Performance measure: khả năng lọc thư rác.
 - Environment: Mails server, mails client.
 - Actuators: đánh dấu thư rác, thông báo, v.v...
 - Sensors: nhận, phân tích nội dung mail.

IV. Các kiểu môi trường.

1. Fully Observable (vs. partially observable).

- Các bộ cảm biến của một Agent cho phép nó truy cập tới trạng thái đầy đủ của môi trường tại mỗi thời điểm.

Fully Observable Task Environment
Tic Tac Toe/Noughts & Crosses/X's and O's



Copyright 2019 @ www.perchingtree.com

Partially Observable Task Environment
Cards Game /Hidden Hand Poker



Copyright 2019 @ www.perchingtree.com

2. Deterministic (vs. stochastic).

- Trạng thái tiếp theo của môi trường được xác định hoàn toàn dựa trên trạng thái hiện tại và hành động của Agent (tại trạng thái hiện tại này).
- Nếu một môi trường là xác định, ngoại trừ đối với các hành động của Agent khác, thì gọi là môi trường chiến lược.
- Examples:
 - o Chess: sẽ chỉ có một số bước đi cho quân cờ tại trạng thái hiện tại và những bước đi này có thể xác định được.
 - o Self – Driving Cars: hành động của ô tô tự lái không là duy nhất, nó diễn biến – thay đổi theo thời gian.

3. Episodic (vs. sequential).

- Kinh nghiệm của Agent được chia thành các giai đoạn (chương/hồi – episodes).
- Mỗi giai đoạn bao gồm việc nhận thức của Agent và hành động mà nó thực hiện.
- Ở mỗi giai đoạn, việc lựa chọn hành động để thực hiện chỉ phụ thuộc vào giai đoạn đó (không phụ thuộc vào giai đoạn khác).
- Trong môi trường tuần tự, *quyết định hiện tại có thể ảnh hưởng đến tất cả các quyết định trong tương lai, chẳng hạn như là Chess và Taxi driving.*

4. Static (vs. dynamic).

- Môi trường không thay đổi trong khi Agent cân nhắc (xem nên đưa ra hành động nào):
 - o Taxi driving là dynamic: Những chiếc xe khác và taxi vẫn tiếp tục di chuyển trong khi thuật toán lái xe quyết định việc cần làm tiếp theo.
 - o Crossword puzzles là static.
- Môi trường bán động (semi – dynamic) là môi trường khi thời gian trôi qua thì nó (môi trường) không thay đổi, nhưng hiệu quả của Agent thì thay đổi.
 - o Chess khi chơi với bộ đếm giờ cho các bước đi là semi – dynamic.

5. Discrete (vs. continuous).

- Tập các nhận thức và các hành động là hữu hạn, được định nghĩa phân biệt rõ ràng).

- Example:
 - o Chess có một tập hợp các nhận thức và hành động riêng biệt.
 - o Taxi – driving là một tập có trạng thái liên tục và liên tục trong thời gian thực.

6. *Single agent (vs. multi – agent).*

- Một Agent hoạt động độc lập (không phụ thuộc/liên hệ với các Agent khác) trong một môi trường.
- Example:
 - o Một Agent giải trò chơi Crossword – puzzle rõ ràng là một môi trường mà Agent hoạt động độc lập.
 - o Trong khi đó, khi Agent chơi Chess – tức là một môi trường 2 – Agent.

Các kiểu môi trường – Ví dụ

	Chơi cờ tính giờ	Chơi cờ không tính giờ	Lái xe taxi
Quan sát đầy đủ?	có	có	không
Xác định?	chiến lược	chiến lược	không
Phân đoạn?	không	không	không
Tĩnh?	bán động	có	không
Rời rạc?	có	có	không
Tác tử đơn?	không	không	không

- Kiểu của môi trường có ảnh hưởng tới quyết định đối với việc thiết kế Agent.
- Môi trường trong thực tế thường có các đặc điểm: chỉ có thể quan sát được một phần, ngẫu nhiên, liên tiếp, thay đổi (động), liên tục và đa tác tử.

V. *Agent functions and programs:*

1. *Agent functions.*

- Một Agent được xác định hoàn toàn bằng một hàm ánh xạ Agent từ nhận thức tới hành động.
- Công việc của AI là thiết kế chương trình Agent thực hiện hàm ánh xạ Agent từ nhận thức tới hành động.
- Hàm một Agent là một hàm hợp lý.
- Mục tiêu : tìm cách triển khai hàm Agent hợp lý một cách chính xác.

2. Agent programs.

- Lấy nhận thức hiện tại làm đầu vào từ các cảm biến (bởi vì không có gì hơn những thứ có sẵn từ môi trường), và trả lại một hành động cho bộ hành động.
- Khác với hàm Agent lấy toàn bộ lịch sử nhận thức.
- Nếu hành động của Agent phụ thuộc vào toàn bộ chuỗi nhận thức, thì Agent phải nhớ tới nhận thức.

Table-Driven Agent

- To keep track of the percept sequence and then use it to index into a table of actions to decide what to do next.
- To build a rational agent in this way, one must construct a table that contains the appropriate action for every possible percept sequence

```
function TABLE-DRIVEN-AGENT (percept) returns action
static: percepts, a sequence, initially empty
       table, a table, indexed by percept sequences, initially fully specified

append percept to the end of percepts
action ← LOOKUP(percepts, table)
return action
```

- Cons :
 - Huge table.
 - Take a long time to build the table.
 - No autonomy.
 - Need a long time to learn the table entries.

Vacuum-cleaner agent

```
Function Reflex-Vacuum-Agent ([Location, status]) returns an action
If status = Dirty then return Suck
Else if location = A then return Right
Else if location = B then return Left
```

Four basic kinds of agent program:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

3. Simple reflex agents (Agent phản xạ đơn giản).

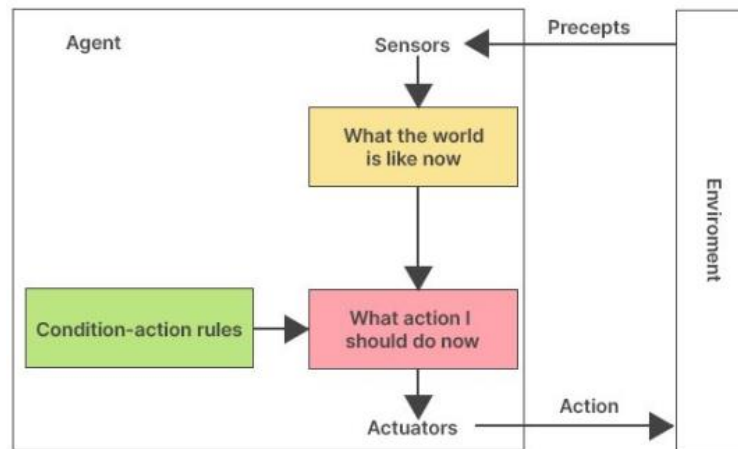
- Hành động theo một quy tắc (luật) có điều kiện phù hợp với trạng thái hiện tại (của môi trường).


```

function SIMPLE-REFLEX-AGENT(percept)
static: rules (tập các luật có dạng: điều kiện-hành động)

state  $\leftarrow$  INTERPRET-INPUT(percept)
rule  $\leftarrow$  RULE-MATCH(state, rules)
action  $\leftarrow$  RULE-ACTION[rule]
return action

```



4. Model – based reflex agents (Agent phản xạ dựa trên mô hình).

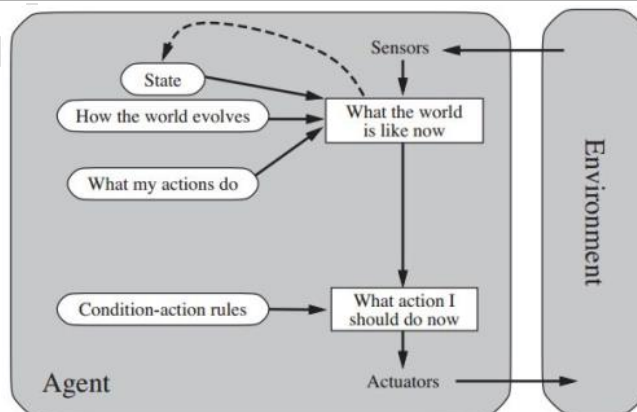
- Sử dụng một mô hình nội bộ để giám sát trạng thái hiện tại của môi trường.
- Lựa chọn hành động: giống như đối với Agent phản xạ đơn giản.

```

function REFLEX-AGENT-WITH-STATE(percept)
static: state (mô tả trạng thái hiện tại của môi trường)
         rules (tập các luật có dạng: điều kiện-hành động)
         action (hành động gần nhất)

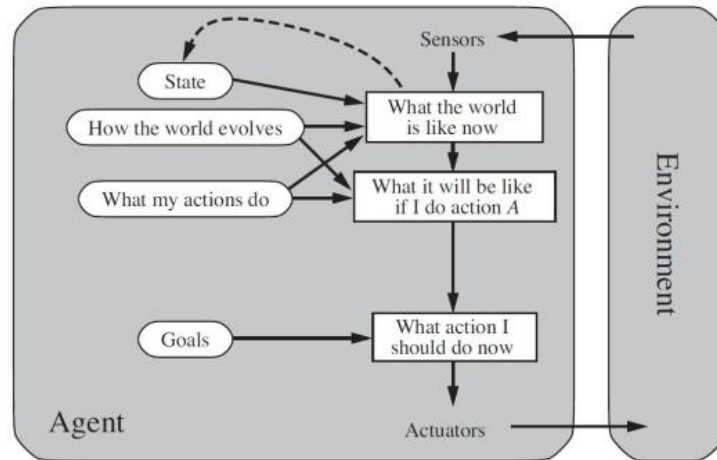
state  $\leftarrow$  UPDATE-STATE(state, action, percept)
rule  $\leftarrow$  RULE-MATCH(state, rules)
action  $\leftarrow$  RULE-ACTION[rule]
return action

```



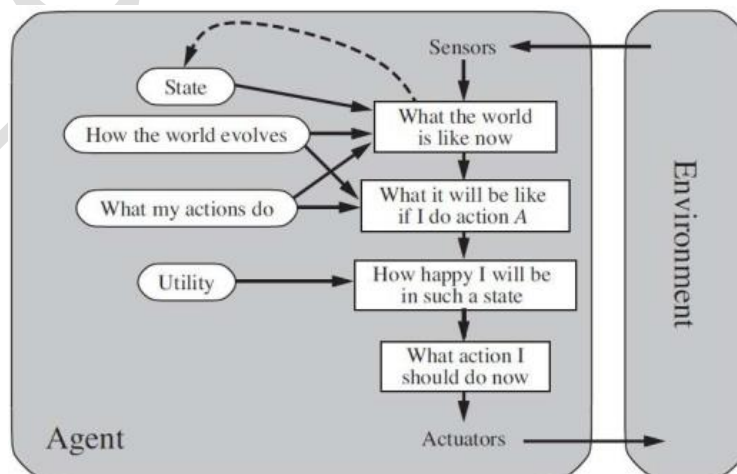
5. Goal – based agent (Agent dựa trên mục tiêu).

- Biết về trạng thái hiện tại của môi trường: chưa đủ → cần biết thêm thông tin về mục tiêu.
 - Trạng thái hiện tại của môi trường: Ở một ngã tư, taxi có thể rẽ phải, rẽ trái, đi thẳng.
 - Thông tin của mục tiêu: Xe taxi cần đi đến đích của hành khách.



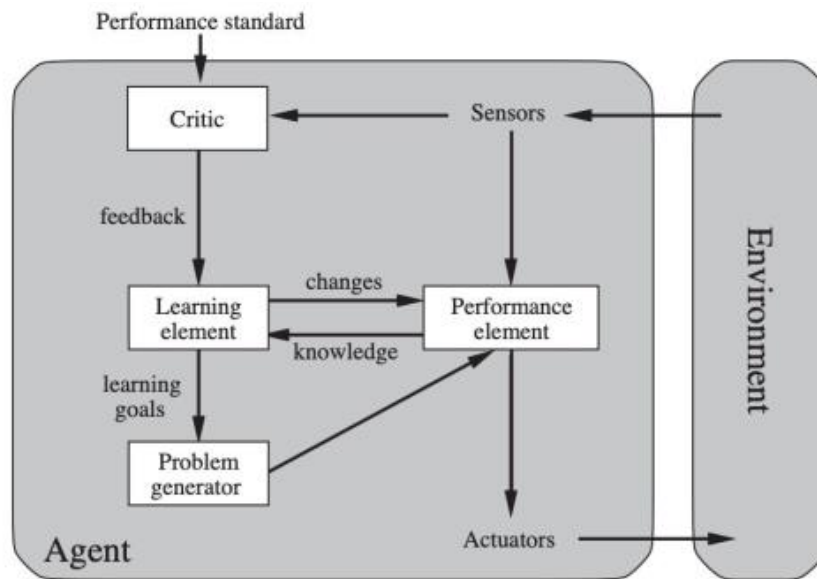
6. Utility – based agents.

- Trong nhiều môi trường, thông tin về các mục tiêu không đủ để đánh giá hiệu quả của các hành động.
 - Có rất nhiều chuỗi các hành động cho phép taxi đi đến đích (tức là đạt đến mục tiêu).
 - Nhưng: chuỗi nào hoạt động nhanh hơn, an toàn hơn, đáng tin cậy hơn, chi phí thấp hơn?
- Cần sự đánh giá lợi ích đối với Agent.
- Hàm lợi ích (utility function)
 - Ánh xạ từ chuỗi các trạng thái của môi trường tới một số giá trị thực (thể hiện mức lợi ích đối với Agents).



7. *Learning Agents.*

- Khả năng học cho phép các Agent cải thiện hiệu quả hoạt động của nó.
- 4 thành phần cấu tạo nên một Agent có khả năng học:
 - o Thành phần hành động: đảm nhiệm việc lựa chọn các hành động.
 - o Thành phần đánh giá: đánh giá hiệu quả hoạt động.
 - o Thành phần học: giúp cải thiện hiệu quả hoạt động – dựa trên các đánh giá, để thay đổi (cải thiện) thành phần hành động.
 - o Thành phần sản sinh kinh nghiệm: có nhiệm vụ đề xuất các hành động giúp sản sinh ra (dẫn đến) các kinh nghiệm mới.



VI. *Solving problems by searching.*

1. *Problem – solving agents.*

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

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

inputs: *percept*, a percept

static: *seq*, an action sequence, initially empty
state, some description of the current world state
goal, a goal, initially null
problem, a problem formulation

state \leftarrow UPDATE-STATE(*state*, *percept*)

if *seq* is empty **then do**

goal \leftarrow FORMULATE-GOAL(*state*)

problem \leftarrow FORMULATE-PROBLEM(*state*, *goal*)

seq \leftarrow SEARCH(*problem*)

action \leftarrow FIRST(*seq*)

seq \leftarrow REST(*seq*)

return *action*

- Example:

- On holiday in Vietnam; at the moment in Nha Trang.
- Flight leaves tomorrow in Hochiminh city.
- Formulate goal: be in Hochiminh city.
- Formulate problem:
 - States: Various provinces.
 - Actions: drive between provinces.
 - Find solutions: sequence of provinces.

■ Một người du lịch đang trong chuyến đi du lịch ở Rumani

- Anh ta hiện thời đang Arad
- Ngày mai, anh ta có chuyến bay khởi hành từ Bucharest
- Bây giờ, anh ta cần di chuyển (lái xe) từ Arad đến Bucharest

■ Phát biểu **mục tiêu**:

- Cần phải có mặt ở Bucharest

■ Phát biểu **bài toán**:

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

■ **Tìm kiếm** giải pháp:

- Chuỗi các thành phố cần đi qua, ví dụ: Arad, Sibiu, Fagaras, Bucharest

2. Problem types.

a. Deterministic, fully observable. → Bài toán trạng thái đơn.

- Agent biết chính xác trạng thái tiếp theo mà nó sẽ chuyển qua.
- Giải pháp của bài toán: một chuỗi hành động.

b. No – observable. → Bài toán thiếu cảm nhận.

- Agent có thể không biết là nó đang ở trạng thái nào.
- Giải pháp của bài toán: một chuỗi hành động.

c. Nondeterministic and/or partially observable. → Bài toán có sự kiện ngẫu nhiên.

- Các nhận thức cung cấp các thông tin mới về trạng thái hiện tại.
- Giải pháp của bài toán: một kế hoạch (chính sách).
- Thường kết hợp đan xen giữa: tìm kiếm và thực hiện.

d. Unknown state space. → Bài toán thăm dò.

→ Xem lại các ví dụ: Vacuum world, 8 – puzzle.

3. Tree search algorithms.

- Ý tưởng:

- Khám phá (xét) không gian trạng thái bằng cách sinh ra các trạng thái kế tiếp của các trạng thái đã khám phá (đã xét).
- Còn gọi là phương pháp khai triển (phát triển) các trạng thái.

```
function GENERAL-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

- General tree – search algorithm:

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE(node)) then return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```

- Một trạng thái (state) là một biểu diễn của một hình trạng (configuration) thực tế.
- Một nút (node) là một phần cấu thành nên cấu trúc dữ liệu của một cây tìm kiếm.
 - Một nút chứa các thuộc tính: *trạng thái*, *nút cha*, *nút con*, *hành động*, *độ sâu*, *chi phí đường đi g(x)*.
- Hàm Expand tạo nên các nút mới,
 - Gán giá trị cho các thuộc tính (của nút mới).
 - Sử dụng hàm Successor – Fn để tạo nên các trạng thái tương ứng với các nút mới đó.

4. Search strategies (Chiến lược tìm kiếm).

- Một chiến lược tìm kiếm được xác định bằng việc chọn trình tự phát triển (khai triển) các nút.
- Các chiến lược tìm kiếm được đánh giá theo các tiêu chí:
 - o Tính hoàn chỉnh (completeness): Có đảm bảo tìm được lời giải? (nếu thực sự tồn tại một lời giải).
 - o Độ phức tạp về thời gian: Số lượng các nút được sinh ra.
 - o Độ phức tạp về bộ nhớ: Số lượng tối đa các nút được lưu trong bộ nhớ.
 - o Tính tối ưu: Có đảm bảo tìm được lời giải có chi phí thấp nhất?
- Độ phức tạp về thời gian và bộ nhớ được đánh giá bởi:
 - o b: hệ số phân nhánh tối đa của cây tìm kiếm.
 - o d: độ sâu của lời giải có chi phí thấp nhất.
 - o m: độ sâu tối đa của không gian trạng thái (độ sâu của cây) – có thể là ∞ .

VII. Search Algorithms:

- Bao gồm:
 1. Uninformed search algorithms.
 2. Informed search algorithms.

1. Uninformed search strategies.

- Còn được gọi là “tìm kiếm mù” (blind search).
- Không có thông tin bổ sung về các trạng thái ngoài các thông tin được đề cập tới trong vấn đề.
- Tất cả những gì chúng ta có thể làm là tạo ra các “successors” và phân biệt trạng thái đích với các trạng thái không phải là đích.
- Khác với “informed strategies” hoặc “heuristic strategies”.
- Một vài uninformed search strategies:
 - o Breadth – first search.
 - o Uniform – cost search.
 - o Depth – first search.
 - o Depth – limited search.
 - o Iterative deepening search.

a. Breadth – first search (tìm kiếm theo chiều rộng).

- Phát triển các nút chưa xét theo chiều rộng – các nút được xét theo thứ tự độ sâu tăng dần.
- Fringe là một cấu trúc kiểu hàng đợi (First in – First out queue, các nút mới được bổ sung vào cuối fringe).
- Tìm kiếm theo chiều rộng tức là xây dựng cây tìm kiếm theo chiều rộng, nút gốc được mở rộng đầu tiên, sau đó tất cả những nút con có được từ nút gốc sẽ được mở rộng, tiếp theo và sau đó là nút con của chúng và cứ thế tiếp tục.
- Tổng quát, ta có thể thực hiện như sau: tất cả các nút ở độ sâu là d trong cây tìm kiếm sẽ được mở rộng trước những nút ở độ sâu d+1.
- Các đặc điểm:
 - o Tính hoàn chỉnh? → Có, nếu b hữu hạn.
 - o Độ phức tạp về thời gian? → $O(b^{d+1})$.

- Độ phức tạp về bộ nhớ? $\rightarrow O(b^{d+1})$. – Lưu tất cả các nút trong bộ nhớ.
- Tính tối ưu? \rightarrow Có, nếu chi phí = 1 cho mỗi bước; tức là trong trường hợp tất cả các biến đổi đều có cùng chi phí.

```

Procedure Breadth_Search
Begin
  1. Khởi tạo danh sách L chứa trạng thái ban đầu;
  2. While (1)
    2.1 if L rỗng then
      {
        Thông báo tìm kiếm thất bại;
        stop;
      }
    2.2 Loại trạng thái u ở đầu danh sách L;
    2.3 if u là trạng thái kết thúc then
      {
        Thông báo tìm kiếm thành công;
        stop;
      }
    2.4 Lấy các trạng thái v kề với u và thêm vào cuối danh sách L;
      for mỗi trạng thái v kề u do
        father(v) = u;
  end
  
```

b. Uniform – cost search (tìm kiếm với chi phí đồng nhất).

- Phát triển các nút chưa xét có chi phí thấp nhất – các nút được xét theo thứ tự chi phí (từ nút gốc tới nút đang xét) tăng dần. Nếu các chi phí ở các bước bằng nhau \rightarrow BFS.
- Các đặc điểm:
 - Tính hoàn chỉnh? \rightarrow Có, nếu chi phí ở mỗi bước $\geq \epsilon$.
 - Độ phức tạp về thời gian? \rightarrow Phụ thuộc vào tổng số các nút có chi phí \leq chi phí của lời giải tối ưu: $O(b^{\lfloor C^*/\epsilon \rfloor})$, trong đó C^* là chi phí của lời giải tối ưu.
 - Độ phức tạp về bộ nhớ? \rightarrow Phụ thuộc vào tổng số các nút có chi phí \leq chi phí của lời giải tối ưu: $O(b^{\lfloor C^*/\epsilon \rfloor})$.
 - Tính tối ưu? \rightarrow Có (nếu các nút được xét theo thứ tự tăng dần về chi phí $g(n)$).

```

function Tim_kiem_UCS(bài_toán, ngăn_chứa) return lời_giải hoặc thất bại.
ngăn_chứa  $\leftarrow$  Tạo_Hàng_Đội_Rỗng()
ngăn_chứa  $\leftarrow$  Thêm(TẠO_NÚT(Trạng_Thái_Đầu[bài_toán]), ngăn_chứa)
loop do
  if Là_Rỗng(ngăn_chứa) then return thất bại.
  nút  $\leftarrow$  Lấy_Chi_phí_Nhỏ_nhất(ngăn_chứa)
  if Kiểm_tra_Câu_hỏi_đích[bài_toán] trên Trạng_thái[nút] đúng.
    then return Lời_giải(nút).
lg  $\leftarrow$  Mò(nút, bài_toán) //lg tập các nút con mới
ngăn_chứa  $\leftarrow$  Thêm_Tất_cả(lg, ngăn_chứa)
  
```

c. Depth – first search (tìm kiếm theo chiều sâu).

- Phát triển các nút chưa xét theo chiều sâu – các nút được xét theo thứ tự độ sâu giảm dần.
- Các đặc điểm:
 - Tính hoàn chỉnh? → Thất bại (không tìm được lời giải) nếu không gian trạng thái có độ sâu vô hạn, hoặc nếu không gian trạng thái chứa các vòng lặp giữa các trạng thái.
 - Đề cử: sửa đổi để tránh việc một trạng thái nào đó bị lặp lại (bị xét lại) theo một đường đi tìm kiếm. → Đạt tính hoàn chỉnh đối với không gian trạng thái hữu hạn.
 - Độ phức tạp về thời gian? → $O(b^m)$: rất lớn, nếu m lớn hơn nhiều so với d.
 - Độ phức tạp về bộ nhớ? → $O(bm)$: độ phức tạp tuyến tính.
 - Tính tối ưu? → Không.

```
Procedure Depth_Search
Begin
  1. Khởi tạo danh sách L chứa trạng thái ban đầu;
  2. While (1)
    2.1 if L rỗng then
      {
        Thông báo tìm kiếm thất bại;
        stop;
      }
    2.2 Loại trạng thái u ở đầu danh sách L;
    2.3 if u là trạng thái kết thúc then
      {
        Thông báo tìm kiếm thành công;
        stop;
      }
    2.4 Lấy các trạng thái v kề với u và thêm vào đầu danh sách L;
      for mỗi trạng thái v kề u do
        father(v) = u;
end
```

d. Depth – limited search (Tìm kiếm giới hạn độ sâu – DLS).

- Là phương pháp tìm kiếm theo chiều sâu DFS + sử dụng giới hạn về độ sâu/trong quá trình tìm kiếm. → các nút ở độ sâu / không có nút con.

```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred?  $\leftarrow$  false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
    result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
    if result = cutoff then cutoff-occurred?  $\leftarrow$  true
    else if result  $\neq$  failure then return result
  if cutoff-occurred? then return cutoff else return failure
```


e. Iterative deepening search. (Tìm kiếm sâu dần).

- Nếu tất cả các lời giải (các nút đích) nằm ở độ sâu lớn hơn giới hạn độ sâu L , thì giải thuật DLS thất bại (không tìm được lời giải).

```

function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth  $\leftarrow 0$  to  $\infty$  do
        result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
        if result  $\neq$  cutoff then return result

```

- So sánh các giải thuật.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

2. Informed search algorithms.

- Các chiến lược tìm kiếm cơ bản (uninformed search strategies) chỉ sử dụng các thông tin chứa trong định nghĩa của bài toán. \rightarrow Không phù hợp với nhiều bài toán thực tế (do đòi hỏi chi phí quá cao về thời gian và bộ nhớ).
- Các chiến lược tìm kiếm với tri thức bổ sung (informed search strategies) sử dụng các tri thức cụ thể của bài toán \rightarrow quá trình tìm kiếm hiệu quả hơn.
 - o Best – first search algorithms (Greedy best – first, A^*)
 - o Local search algorithms (Hill – climbing, Simulated annealing, Local beam, Genetic algorithms).

a. Best – first search.

- Ý tưởng: sử dụng một hàm đánh giá $f(n)$ cho mỗi nút của cây tìm kiếm.
 - o Để đánh giá mức độ “phù hợp” của nút đó. \rightarrow Trong quá trình tìm kiếm, ưu tiên xét các nút có mức độ phù hợp cao nhất.
- Special cases: Greedy best – first search, A^* search.

i. Greedy best – first search

- Phát triển hàm đánh giá $f(n)$ là hàm heuristic $h(n)$.
- Hàm heuristic $h(n)$ để đánh giá chi phí để đi từ nút hiện tại n tới nút đích (mục tiêu).
 - o Ví dụ: Trong bài toán tìm đường đi từ Arad đến Bucharest, sử dụng $h_{SLD}(n)$ = Ước lượng khoảng cách đường thẳng (“chim bay”) từ thành phố hiện tại n đến Bucharest.
- Phương pháp tìm kiếm Greedy best – first search sẽ xét (phát triển) nút “có vẻ” gần với nút đích (mục tiêu) nhất.

- Các đặc điểm:
 - Tính hoàn chỉnh? → Không, vì có thể vướng (chết tắc) trong các vòng lặp.
 - Độ phức tạp về thời gian? → $O(b^m)$, một hàm heuristic tốt có thể mang lại cải thiện lớn.
 - Độ phức tạp về bộ nhớ? → $O(b^m)$, lưu giữ tất cả các nút trong bộ nhớ.
 - Tính tối ưu? → Không.

ii. A^* search.

- Tránh việc xét (phát triển) các nhánh tìm kiếm đã xác định (cho đến thời điểm hiện tại) là có chi phí cao.
- Giảm thiểu tổng chi phí của lời giải.
- Sử dụng hàm đánh giá $f(n) = g(n) + h(n)$.
 - $g(n)$ = chi phí từ nút gốc cho đến nút hiện tại n .
 - $h(n)$ = chi phí ước lượng từ nút hiện tại n tới đích.
 - $f(n)$ = chi phí tổng thể ước lượng của đường đi qua nút hiện tại n đến đích.
- Các đặc điểm của A^* search:
 - Nếu không gian các trạng thái là hữu hạn và có giải pháp để tránh việc xét lặp lại các trạng thái, thì giải thuật A^* là hoàn chỉnh (tìm được lời giải) – nhưng không đảm bảo là tối ưu.
 - Nếu không gian các trạng thái là hữu hạn và không có giải pháp để tránh việc xét lặp lại các trạng thái, thì giải thuật A^* là không hoàn chỉnh (có thể không tìm được lời giải).
 - Nếu không gian các trạng thái là vô hạn, thì giải thuật A^* là không hoàn chỉnh (không đảm bảo tìm được lời giải).
 - Tính hoàn chỉnh? → Có (trừ khi có rất nhiều nút có chi phí $f \leq f(G)$)
 - Độ phức tạp về thời gian? → Bậc của hàm mũ – Số lượng các nút được xét là hàm mũ của độ dài đường đi của lời giải.
 - Độ phức tạp về bộ nhớ? → Lưu giữ tất cả các nút trong bộ nhớ.
 - Tính tối ưu? → Có.

Procedure Astar-Search

Begin

1. Đặt OPEN chỉ chứa T_0 . Đặt $g(T_0) = 0$, $h(T_0) = 0$ và $f(T_0) = 0$. Đặt CLOSE là tập rỗng.
2. Lặp lại các bước cho đến khi gặp điều kiện dừng
 - 2.a. Nếu OPEN rỗng: bài toán vô nghiệm, thoát.
 - 2.b. Ngược lại, chọn T_{\max} trong OPEN sao cho $f(T_{\max})$ là nhỏ nhất
 - 2.b.1. Lấy T_{\max} ra khỏi OPEN và đưa T_{\max} vào CLOSE.
 - 2.b.2. Nếu T_{\max} là TG (trạng thái đích) thì thoát và thông báo lời giải là T_{\max}
 - 2.b.3. Nếu T_{\max} không phải là TG. Tạo ra danh sách tất cả các trạng thái kế tiếp của T_{\max} .

Gọi một trạng thái này T_k . Với mỗi T_k , làm các bước sau:

2.b.3.1. Tính $g(T_k) = g(T_{\max}) + \text{cost}(T_{\max}, T_k)$

2.b.3.2. Nếu tồn tại $T_{k'}$ trong OPEN trùng với T_k .

Nếu $g(T_k) < g(T_{k'})$ thì

Đặt $g(T_{k'}) = g(T_k)$

Tính lại $f(T_{k'})$

Đặt $\text{Cha}(T_{k'}) = T_{\max}$

2.b.3.3. Nếu tồn tại $T_{k'}$ trong CLOSE trùng với T_k

Nếu $g(T_k) < g(T_{k'})$ thì

Đặt $g(T_{k'}) = g(T_k)$

Tính lại $f(T_{k'})$

Đặt $\text{Cha}(T_{k'}) = T_{\max}$

Lan truyền sự thay đổi giá trị g , f cho tất cả các trạng thái tiếp theo của T_i (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

2.b.3.4. Nếu T_k chưa xuất hiện trong cả OPEN lẫn CLOSE thì

Thêm T_k vào OPEN

Tính: $f(T_k) = g(T_k) + h(T_k)$

iii. Admissible heuristics.

- Một ước lượng (heuristic) $h(n)$ được xem là chấp nhận được nếu đối với mọi nút n : $0 \leq h(n) \leq h^*(n)$, trong đó $h^*(n)$ là chi phí thật (thực tế) để đi từ nút n đến đích.
- Một ước lượng (heuristic) chấp nhận được không bao giờ đánh giá quá cao (overestimate) đối với chi phí để đi tới đích.
- Ước lượng $h_{SLD}(n)$ không bao giờ đánh giá quá cao khoảng cách đường đi thực tế.
- Vì $g(n)$ là chi phí chính xác để đạt được n nên $f(n)$ không bao giờ đánh giá quá cao chi phí thực sự của một lời giải đi qua n .
- **Định lý:** Nếu $h(n)$ là đánh giá chấp nhận được, thì phương pháp tìm kiếm A^* sử dụng giải thuật **Tree – search** là tối ưu.

- **Chứng minh:** Giả sử có một đích không tối ưu (suboptimal goal) G_2 được sinh ra và lưu trong cấu trúc *fringe*. Gọi n là một nút chưa xét trong cấu trúc *fringe* sao cho n nằm trên một đường đi ngắn nhất đến một đích tối ưu (optimal goal) G .
 - Ta có: $f(G_2) = g(G_2)$ vì $h(G_2) = 0$.
 - $g(G_2) > g(G)$ vì G_2 là đích đến không tối ưu.
 - $f(G) = g(G)$ vì $h(G) = 0$.
 - Suy ra: $f(G_2) > f(G)$.
 - Ta có $h(n) \leq h^*(n)$ vì h là ước lượng chấp nhận được.
 - Suy ra: $g(n) + h(n) \leq g(n) + h^*(n)$.
 - Mà: $f(n) \leq f(G)$ vì n nằm trên đường đi tới G .
 - Vì vậy: $f(G_2) \geq f(n)$. Tức là, thủ tục A^* không bao giờ xét G_2 .

iv. **Consistent heuristics.**

- Một ước lượng (heuristic) được xem là kiên định (consistent), nếu với mọi nút n và với mọi nút tiếp theo n' của n (được sinh ra bởi hành động a):

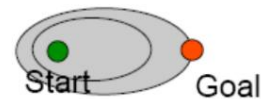
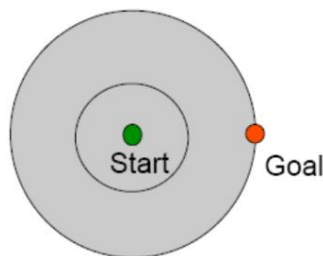
$$h(n) \leq c(n, a, n') + h(n').$$
- Nếu ước lượng h là kiên định, ta có:

$$\begin{aligned} f'(n) &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) = f(n) \end{aligned}$$

Nghĩa là: $f(n)$ không giảm trong bất kỳ đường đi (tìm kiếm) nào đi qua n .
- **Định lý:** Nếu $h(n)$ là kiên định (consistent), thì phương pháp tìm kiếm A^* sử dụng giải thuật **Graph – search** là tối ưu.

v. **So sánh BFS và UCS.**

- Tìm kiếm với chi phí cực tiểu (UCS) phát triển theo mọi hướng
- Tìm kiếm A^* phát triển chủ yếu theo hướng tới đích, nhưng đảm bảo tính tối ưu



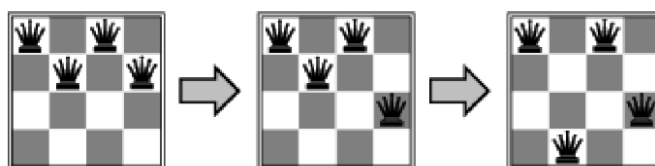
- Ta có thể thấy cả hai thuật toán UCS và BFS đều sử dụng hàng đợi ưu tiên để lưu danh sách các node chờ duyệt, và đây có lẽ là nguyên nhân mấu chốt dẫn đến việc gây nhầm lẫn giữa hai thuật toán.
- Qua mỗi bước duyệt, các node còn lại trong danh sách đều được so sánh để sắp xếp lại và chọn ra node "được cho là" có chi phí thấp nhất. Nhưng sự khác nhau ở hai thuật toán có vẻ cũng nằm ở đây. Trước khi sắp xếp lại danh sách các node, UCS vì không có hàm đánh giá nên nó cần tự thân vận động:

Tiêu chí	Chiến lược	Chi phí	Độ phức tạp theo thời gian	Độ phức tạp bộ nhớ	Danh sách các node chờ duyệt	Tính hoàn chỉnh	Tính tối ưu	Ghi chú
UCS	Tìm kiếm mù, không sử dụng hàm đánh giá	Tính từ node bắt đầu tới node hiện tại	$O(b^d)$	$O(b^d)$	Hàng đợi ưu tiên	Luôn tìm thấy	Có nếu chi phí > 0	Khi đồ thị có chi phí ở mỗi bước là như nhau thì thuật toán trở thành phương pháp tìm kiếm theo chiều rộng.
Best - First Search	Tìm kiếm kinh nghiệm, sử dụng hàm đánh giá	Tính từ node trước đến node hiện tại	$O(b^m)$	$O(b^m)$	Hàng đợi ưu tiên	Có thể đi xa khỏi lời giải, kẹt trong vòng lặp.	Không	Một hàm đánh giá tốt có thể giảm thời gian và không gian nhớ một cách đáng kể.

- Chi phí node $N = \text{Chi phí từ gốc đến node trước node } N + \text{chi phí node trước } N \text{ đến } N$.
- Trong khi đó, BFS không phải làm gì cả vì nó đã có hàm đánh giá ngay tại node rồi.

b. Các giải thuật tìm kiếm cục bộ.

- Trong nhiều bài toán tối ưu, đường đi tới đích không quan trọng – mà quan trọng là trạng thái đích.
 - Trạng thái đích = Lời giải của bài toán.
- Không gian trạng thái = Tập hợp các cấu hình “hoàn chỉnh”.
- Mục tiêu: Tìm một cấu hình thỏa mãn các ràng buộc.
 - Ví dụ: Bài toán n quân hậu (bố trí n quân hậu trên một bàn cờ kích thước $n \times n$ sao cho các quân hậu không ăn nhau).
- Trong những bài toán như thế, chúng ta có thể sử dụng các giải thuật tìm kiếm cục bộ.
- Tại mỗi thời điểm, chỉ lưu một trạng thái “hiện thời” duy nhất – Mục tiêu: cố gắng “cải thiện” trạng thái (cấu hình) hiện thời này đối với một tiêu chí nào đó (định trước).
 - Không mang tính hệ thống.
 - Sử dụng rất ít bộ nhớ - thường là một lượng không đổi.
 - Bố trí $n (=4)$ quân hậu trên một bàn cờ có kích thước $n \times n$, sao cho không có 2 quân hậu nào trên cùng hàng, hoặc trên cùng cột, hoặc trên cùng đường chéo



i. Hill – climbing algorithm.

```
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 ← MAKE-NODE(INITIAL-STATE[problem])
loop do
  neighbor ← a highest-valued successor of current
  if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
  current ← neighbor
```

- Xem các ví dụ: Bài toán 8 quân hậu, 8 – puzzle.
- Nhược điểm: Tùy vào trạng thái đầu, giải thuật tìm kiếm leo đồi có thể “tắc” ở các điểm cực đại cục bộ (local maxima).
 - o Không tìm được lời giải tối ưu toàn cục (global optimal solution).

ii. Simulated annealing search.

- Dựa trên quá trình tôi ủ (annealing process): Kim loại nguội đi và lạnh cứng lại thành cấu trúc kết tinh.
- Phương pháp tìm kiếm Simulated Annealing có thể tránh được các điểm tối ưu cục bộ (local optimal).
- Phương pháp tìm kiếm Simulated Annealing sử dụng chiến lược tìm kiếm ngẫu nhiên, trong đó chấp nhận các thay đổi làm tăng giá trị hàm mục tiêu (i.e., cần tối ưu) và cũng chấp nhận (có hạn chế) cả thay đổi làm giảm.
- Phương pháp tìm kiếm Simulated Annealing sử dụng một tham số điều khiển T (như trong các hệ thống nhiệt độ).
 - o Bắt đầu thì T nhận giá trị cao và giảm dần về 0.
 - o Ý tưởng: Thoát khỏi (vượt qua) các điểm tối ưu cục bộ bằng cách cho phép cả các dịch chuyển “tồi” từ trạng thái hiện thời, nhưng giảm dần tần suất của các di chuyển tồi này.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
       schedule, a mapping from time to “temperature”
local variables: current, a node
                 next, a node
                 T, a “temperature” controlling prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for  $t \leftarrow 1$  to  $\infty$  do
   $T \leftarrow \text{schedule}[t]$ 
  if  $T = 0$  then return current
  next ← a randomly selected successor of current
   $\Delta E \leftarrow \text{VALUE}[\text{next}] - \text{VALUE}[\text{current}]$ 
  if  $\Delta E > 0$  then current ← next
  else current ← next only with probability  $e^{\Delta E/T}$ 
```

VIII. Knowledge representation.

1. Introduction.

- Con người có thể cảm nhận thế giới quan bằng các giác quan, sử dụng các tri thức tích lũy được và bằng các lập luận, suy diễn để đưa ra những hành động thích hợp.
- Các hệ thống thông minh (intelligent agent) cần phải có tri thức về thế giới hiện thực và môi trường xung quanh để đưa ra những quyết định đúng đắn.

2. Knowledge – based agents.

- Knowledge - based agents: là hệ tri thức chứa các cơ sở tri thức (knowledge base).
- Cơ sở tri thức là tập hợp các tri thức được biểu diễn dưới dạng nào đó.
- Mỗi khi nhận được thông tin đưa vào (input data), các agent phải có khả năng suy diễn để đưa ra những phương án chính xác hợp lý.
- Hệ tri thức cần được trang bị một cơ chế suy diễn.
- Đối với hệ thống thông minh giải quyết một vấn đề nào thì cơ sở tri thức sẽ chứa các tri thức tương ứng.
- Tri thức (knowledge) là khái niệm trừu tượng.
- Theo từ điển Oxford, “knowledge is information and skills acquired through experience or education”.

3. Kinds of knowledge.

- Tri thức lý thuyết (theoretical or a priori knowledge): là tri thức đạt được mà không cần quan sát thế giới quan.
- Tri thức thực tiễn (empirical knowledge): là tri thức đạt được bằng những quan sát và tương tác với môi trường xung quanh.
- Tri thức mô tả (declarative knowledge): bao gồm những từ ngữ mô tả chính xác về một sự vật, hiện tượng.
- Tri thức quy trình (procedural knowledge): bao gồm những từ ngữ dùng để mô tả một quá trình (process) nào đó.
- Tri thức heuristic (heuristic knowledge): là tri thức nông cạn do không đảm bảo chính xác hoặc tối ưu khi giải quyết vấn đề. Thường được coi như mẹo nhằm dẫn dắt quá trình lập luận.

4. Expert system.

- Chuyên gia (expert) là những người có kiến thức sâu sắc về một vấn đề nào đó và có thể giải quyết tốt những việc mà ít ai làm được.
- Hệ chuyên gia (expert system) là chương trình máy tính có thể thực hiện những công việc trong một lĩnh vực nào đó như một chuyên gia thực thụ.
- Là các hệ tri thức dựa trên luật suy diễn phức tạp.
- Áp dụng trên rất nhiều lĩnh vực trong công nghiệp như: Marketing, nông nghiệp, y tế, điện lực, v.v...

5. Knowledge representation.

- Logic mệnh đề (propositional logic).
- Logic vị từ cấp một (first – order predicate logic).

a. **Propositional logic.**

- Logic mệnh đề là công cụ logic trong đó các mệnh đề được mã hóa cho một biến hoặc hằng, còn các biểu thức là sự liên kết có nghĩa giữa các biến và các toán tử logic nhất định.
- Ví dụ: “Nếu tôi cố gắng làm bài tập (A) thì tôi sẽ thi tốt (B)” được mô tả như sau:
 $A \rightarrow B$
- Tri thức sẽ được mô tả dưới dạng các mệnh đề trong ngôn ngữ biểu diễn tri thức.
- Gồm hai thành phần cơ bản: cú pháp và ngữ nghĩa.
 - o Cú pháp của một ngôn ngữ bao gồm các ký hiệu và quy tắc liên kết các ký hiệu (luật cú pháp) để tạo thành các câu (công thức) trong ngôn ngữ.
 - o Ngữ nghĩa của ngôn ngữ cho phép chúng ta xác định ý nghĩa của các câu trong miền nào đó của thế giới thực.
 - o Ví dụ: $1 + 2 + 3 + 4 + \dots + n$.
 - o Ngoài cú pháp và ngữ nghĩa, ngôn ngữ biểu diễn tri thức cần được cung cấp cơ chế suy diễn, giúp chúng ta tìm ra một công thức mới từ một tập nào đó các công thức.
- Ngôn ngữ biểu diễn tri thức = cú pháp + ngữ nghĩa + cơ chế suy diễn.
- Một ngôn ngữ biểu diễn tri thức tốt cần có khả năng biểu diễn rộng, tức là mô tả được hầu hết điều chúng ta muốn.
- Hiệu quả đi đến kết luận + thủ tục suy diễn đòi hỏi ít thời gian và không gian nhớ.
- Càng gần với ngôn ngữ tự nhiên càng tốt.

b. **Cú pháp.**

- Cú pháp của logic mệnh đề đơn giản và cho phép xây dựng các công thức.
- Gồm tập các ký hiệu và tập các luật xây dựng công thức.
- Các ký hiệu:
 - o Hằng logic: True và False.
 - o Các ký hiệu mệnh đề: P, Q, R, ...
 - o Các phép kết nối logic: \wedge , \vee , \rightarrow , \leftrightarrow .
 - o Các dấu mở ngoặc và đóng ngoặc.
- Các quy tắc xây dựng công thức:
 - o $A \wedge B$
 - o $A \vee B$
 - o $A \leftrightarrow B$
 - o $A \rightarrow B$

c. **Propositional logic.**

- Ngữ nghĩa của logic mệnh đề giúp ta xác định được ý nghĩa thực sự của các công thức.
- Bất kỳ một sự kết hợp các ký hiệu mệnh đề với các sự kiện trong thế giới thực được gọi là minh họa (interpretation).
Thường được gán một giá trị chân lý True hoặc False.
- Bảng chân lý giúp ta xác định ngữ nghĩa câu phức hợp.
- Một công thức được gọi là thỏa mãn (satisfiable) nếu nó đúng trong một minh họa nào đó.

- Ví dụ: $(P \vee Q) \wedge R$ thỏa mãn vì nó có giá trị True khi $\{P: \text{False}, Q: \text{True}, R: \text{True}\}$.
- Một công thức gọi là vững chắc (valid) nếu nó đúng trong mọi minh họa.
- Một công thức gọi là không thỏa được nếu nó sai trong mọi minh họa.
- Ta gọi một mô hình (model) của một công thức là một minh họa để công thức đúng trong trường hợp nào đó.
- Cách xác định một công thức có vững chắc (thỏa mãn, không thỏa mãn): lập bảng chân trị.
 - Một tập các công thức $G = (G_1, \dots, G_n)$ là vững chắc (thỏa mãn, không thỏa mãn) nếu hội của chúng $G_1 \wedge G_2 \wedge \dots \wedge G_n$ là vững chắc (thỏa mãn, không thỏa mãn).
 - Một mô hình của G là mô hình của công thức $G_1 \wedge G_2 \wedge \dots \wedge G_n$.
- Hai công thức được gọi là tương đương nếu chúng có cùng giá trị chân lý trong mọi trường hợp. Ta chỉ hai công thức A và B tương đương, ta viết: $A \equiv B$.
 - Luật De Morgan, giao hoán, kết hợp, phân phối.
- Để viết chương trình trên máy tính thao tác các công thức, chúng ta thường chuẩn hóa chúng về dạng chuẩn tắc.
- Một công thức được gọi là chuẩn tắc nếu nó là hội của các câu phức tạp (clause). Một câu phức tạp có dạng: $A_1 \vee A_2 \vee \dots \vee A_n$, trong đó A_i là các câu đơn (literal).
 - Phương pháp chuẩn hóa:
 - Bỏ các dấu kéo theo bằng các luật.
 - Chuyển các dấu phủ định bằng luật De Morgan.
 - Áp dụng luật phân phối, thay công thức.
- Khi biểu diễn tri thức bởi các công thức trong logic mệnh đề, cơ sở tri thức là một tập nào đó các công thức. Bằng phương pháp chuẩn hoá vừa nêu, thì cơ sở tri thức là một tập hợp các câu phức hợp (clause).
- Như vậy, mọi công thức đều có thể đưa về dạng chuẩn tắc là hội của các clause. Mỗi clause có dạng:
 - $\text{Not}(A_1) \vee \dots \vee \text{Not}(A_m) \vee B_1 \vee \dots \vee B_n$ trong đó A_i, B_i là các mệnh đề.
 - Câu Kowalski có dạng:
 - $A_1 \wedge \dots \wedge A_m \rightarrow B_1 \vee \dots \vee B_n$.
 - Khi $n \leq 1$, clause chỉ chứa nhiều nhất một literal dương. Ta gọi những câu như thế là câu Horn (Alfred Horn, 1951).
 - Nếu $m > 0, n = 1$, câu Horn có dạng: $A_1 \wedge \dots \wedge A_m \rightarrow B$.
 - Trong đó các A_i được gọi là các câu điều kiện, còn B là kết luận. Các câu Horn còn gọi là các luật if – then.
 - Khi $m = 0$ và $n = 1$, câu Horn trở thành các câu đơn.
 - Không phải mọi công thức đều có thể biểu diễn dưới dạng hội của các câu Horn.
 - Trong các ứng dụng, cơ sở tri thức thường là tập hợp các câu Horn (tập hợp các luận if – then).

d. **Luật suy diễn.**

- Một công thức H được xem là hệ quả logic (logical consequence) của một tập các công thức $G = \{G_1, \dots, G_n\}$ nếu trong bất kỳ mô hình nào mà $\{G_1, \dots, G_n\}$ đúng thì H cũng đúng.
- Luật suy diễn là phương pháp sử dụng những tri thức có sẵn trong cơ sở tri thức để suy ra tri thức mới là hệ quả logic của các công thức đó.
 - o Luật Modus Ponens: $[(A \rightarrow B) \wedge A] \rightarrow B$
 - o Luật Modus Tollens: $[(A \rightarrow B) \wedge \neg B] \rightarrow \neg A$
 - o Luật bắc cầu.
 - o Luật loại bỏ hội.
 - o Luật đưa vào hội.
 - o Luật đưa vào clause.
 - o Luật phân giải: $[(A \vee B) \wedge (\neg B \vee C)] \rightarrow (A \vee C)$
- Một luật suy diễn là tin cậy nếu bất kỳ mô hình nào của giả thiết của luật cũng là mô hình kết luận.
- Luật phân giải là luật suy diễn tổng quát, bao gồm luật Modus Ponens, Modus Tollens, luật bắc cầu. (CM).
- Giả sử chúng ta có một tập các công thức. Bằng các luật suy diễn, ta có thể suy ra những công thức mới.
 - o Các công thức đã được cho được gọi là các tiên đề.
 - o Các công thức được suy ra được gọi là các định lý.
 - o Dãy các luật suy diễn được áp dụng để dẫn đến các định lý được gọi là một chứng minh của định lý.
- Nếu các luật suy diễn là tin cậy, thì các định lý là hệ quả logic của các tiên đề.
- Trong các hệ tri thức, bằng cách sử dụng các luật suy diễn, người ta thiết kế lên các thủ tục suy diễn để từ các tri thức trong cơ sở tri thức, ta suy ra các tri thức mới đáp ứng nhu cầu người sử dụng.
- Hệ hình thức (formal system) bao gồm một tập các tiên đề và một tập các luật suy diễn nào đó trong một ngôn ngữ biểu diễn tri thức nào đó.
- Một tập suy diễn được gọi là đầy đủ nếu mọi hệ quả logic của một tập các tiên đề đều chứng minh được bằng cách chỉ sử dụng các luật trong tập đó.
- Phương pháp chứng minh bác bỏ là phương pháp được sử dụng phổ biến trong toán học.
- Chứng minh bác bỏ bằng luật phân giải :
 - o Luật phân giải trên các clause.
 - o Luật phân giải trên câu Horn.
 - o Thuật toán Havard (1970).
 - o Thuật toán Robinson (1971).

e. **Proof by contradiction.**

- Để chứng minh P đúng, ta giả sử P sai và dẫn đến một mâu thuẫn.
- Để thuận tiện hơn cho việc sử dụng luật phân giải, ta sẽ cụ thể hóa luật phân giải trên các câu quan trọng.

❑ Luật phân giải trên các câu tuyển (clause)

$$\frac{A_1 \vee \dots \vee A_m \vee C \quad \neg C \vee B_1 \vee \dots \vee B_n}{A_1 \vee \dots \vee A_m \vee B_1 \vee \dots \vee B_n}$$

trong đó $A_1, A_2, \dots, A_m, C, B_1, \dots, B_n$ là literals.

❑ Luật phân giải trên các câu Horn:

$$\frac{P_1 \wedge \dots \wedge P_m \wedge S \Rightarrow Q, \quad R_1 \wedge \dots \wedge R_n \Rightarrow S}{P_1 \wedge \dots \wedge P_m \wedge R_1 \wedge \dots \wedge R_n \Rightarrow Q}$$

❑ Hai câu có thể áp dụng được luật phân giải được gọi là hai câu phân giải được và kết quả nhận được gọi là phân giải thức của chúng

- Hai câu phân giải được nếu một câu chứa một literal đối lập với một literal trong câu kia.
- Giả sử G là một tập các câu tuyển (clause). Ta ký hiệu $R(G)$ là tập bao gồm các câu thuộc G và tất cả các câu nhận được từ G thông qua dãy áp dụng luật phân giải.
- Luật phân giải là luật đầy đủ để chứng minh một tập câu là không thỏa được.
- Định lý phân giải: Một tập câu tuyển là không thỏa được khi và chỉ khi $R(G)$ chứa câu rỗng.
- **Thuật toán Harvard:**
 - Step 1: phát biểu lại giả thiết (GT) và kết luận (KL) của bài toán dưới dạng chuẩn sau: $GT_1, \dots, GT_n \rightarrow KL_1, \dots, KL_m$
 - Trong đó, GT_i, KL_j được xây dựng từ các biến mệnh đề và các phép nối: AND, OR, NOT.
 - Step 2: bước bỏ phủ định. Khi cần bỏ các phủ định, chuyển về.
 - GT_i sang về kết luận KL_j và ngược lại.
 - Step 3: Thay dấu AND ở GT_i và OR ở KL_j bằng “ , ”.
 - Step 4: Nếu GT_i còn dấu OR và KL_j còn dấu AND, tách chúng thành 2 dòng con.
 - Step 5: Một dòng được chứng minh nếu tồn tại chung một mệnh đề ở cả hai vế.
 - Step 6: Bài toán được chứng minh khi và chỉ khi các dòng được chứng minh. Ngược lại, bài toán không được chứng minh.

- **Thuật toán Robinson:**

- Step 1: phát biểu lại giả thiết (GT) và kết luận (KL) của bài toán dưới dạng chuẩn sau: $GT_1, \dots, GT_n \rightarrow KL_1, \dots, KL_m$
 - Trong đó, GT_i, KL_j được xây dựng từ các biến mệnh đề và các phép nối: AND, OR, NOT.
- Step 2: Thay dấu AND ở GT_i và OR ở KL_j bằng “, ”.
- Step 3: Chuyển về KL_j sang về GT_i với dấu phủ định để còn một vế
- Step 4: Xây dựng một mệnh đề mới bằng cách tuyển một cặp mệnh đề từ danh sách các mệnh đề. Nếu mệnh đề mới có các biến mệnh đề đối ngẫu thì mệnh đề đó được loại bỏ.
- Step 5: Bổ sung mệnh đề mới này vào danh sách và lặp lại bước 4
- Step 6: Bài toán được chứng minh khi và chỉ khi còn hai mệnh đề đối ngẫu. Ngược lại bài toán không được chứng minh.

f. **Predicate logic.**

- Logic mệnh đề cho phép chúng ta biểu diễn các sự kiện.
- Một ký hiệu trong logic mệnh đề được minh họa như những sự kiện trong thế giới thực, sử dụng các kết nối logic để tạo ra những câu phức hợp biểu diễn các sự kiện có ý nghĩa phức tạp hơn.
- Khả năng biểu diễn của logic mệnh đề chỉ giới hạn trong phạm vi các sự kiện.
- Để mô tả các thuộc tính của đối tượng, trong logic vị từ, người ta đưa vào các vị từ (predicate).
- Ngoài các kết nối logic như trong logic mệnh đề, logic vị từ cấp một còn sử dụng các lượng từ. Chẳng hạn, lượng từ \forall cho phép ta tạo ra các câu nói tới mọi đối tượng trong một miền đối tượng nào đó.
- Các ký hiệu:
 - Các ký hiệu hằng: a, b, c, John, Jerry, ...
 - Các ký hiệu biến: x, y, z, u, v, w, ...
 - Các ký hiệu vị từ: P, Q, R, S, Prime, Odd, Love, ...
- Mỗi vị từ là vị từ của n biến. Ví dụ: Lova là vị từ của hai biến, Prime là vị từ của một biến.
- Các ký hiệu vị từ không biến là các ký hiệu mệnh đề.
- Các ký hiệu hàm: f, g, cos, sin, v.v...
- Mỗi hàm là hàm của n biến.
- Các ký hiệu kết nối logic giống như trong logic mệnh đề.
- Các ký hiệu lượng từ: \forall, \exists .
- Các ký hiệu ngăn cách: dấu phẩy, dấu mở ngoặc, đóng ngoặc.
- Các hạng thức (term) là các biểu thức mô tả đối tượng.
- Các hạng thức được xác định đệ quy như sau:
 - Các ký hiệu hằng hay biến là hạng thức.
 - Nếu a, b, c, ..., z là n hạng thức và h là hàm n biến thì $h(a, b, c, \dots, z)$ cũng là hạng thức.

- Một hạng thức không chứa biến được gọi là hạng thức cụ thể.
- Chúng ta sẽ biểu diễn các tính chất của đối tượng và các quan hệ giữa các đối tượng bằng công thức phân tử (câu đơn).
- Các câu đơn được xác định đệ quy như sau:
 - Các ký hiệu vị từ không biến (các ký hiệu mệnh đề) là câu đơn.
 - Nếu a, b, c, \dots, z là n hạng thức và P là vị từ của n biến thì $P(a, b, \dots, z)$ là công thức phân tử (câu đơn).
 - Ví dụ: Mary là một ký hiệu hằng, Love là một vị từ hai biến, husband là hàm 1 biến thì $\text{Love}(\text{Mary}, \text{husband}(\text{Mary}))$ là một công thức phân tử.
- Từ các công thức phân tử, ta sử dụng các kết nối logic và các lượng từ để xây dựng các công thức (các câu) bằng đệ quy như sau:
 - Các công thức phân tử là các công thức.
 - Nếu G, H là các công thức thì các biểu thức logic của G và H là công thức.
 - Nếu G là một công thức và x là biến thì các biểu thức $(\exists x G), (\forall x G)$ là công thức.
 - Các công thức không phải là công thức phân tử thì được gọi là các công thức phức hợp.
- Các công thức không chứa biết thì được gọi là các công thức cụ thể.
- Lượng từ phổ dụng \forall cho phép ta mô tả một lớp các đối tượng.
- Lượng từ tồn tại \exists cho phép ta nói đến một đối tượng nào đó trong một lớp đối tượng.
- Một công thức phân tử hoặc phủ định công thức phân tử được gọi là literal.
- Một công thức là tuyển của các literal được gọi là câu tuyển.
- Một công thức mà các biến bắt buộc xuất hiện thì gọi là công thức đóng.
- Ý nghĩa của các công thức trong một thế giới hiện thực nào đó thì được gọi là minh họa.
- Trong một minh họa, các ký hiệu vị từ sẽ được gán với một thuộc tính hoặc một quan hệ cụ thể nào đó.
- Khi đã xác định được ngữ nghĩa một câu đơn, ta có thể xác định được ngữ nghĩa của các câu phức hợp.
- Hai công thức tương đương nếu như nó cùng sai hoặc cùng đúng trong mọi minh họa.
- Từ các câu phân tử, bằng cách sử dụng các kết nối logic và các lượng từ, ta có thể tạo ra các câu phức hợp có cấu trúc phức tạp. Để dễ dàng cho việc lưu trữ các câu trong bộ nhớ và thuận lợi cho việc xây dựng các thủ tục suy diễn, ta cần chuẩn hóa các câu bằng cách đưa chúng về dạng chuẩn tắc hội (hội của các câu tuyển).
- *Thủ tục chuẩn hóa các công thức:* (xem slide)
 - Loại bỏ các kéo theo.
 - Chuyển các phủ định tới các phân tử.
 - Loại bỏ các lượng từ tồn tại.
 - Loại bỏ các lượng từ phổ dụng.
 - Chuyển các tuyển tới literals

- Loại bỏ các hội.
- Đặt tên lại các biến.
- *Các luật suy diễn:* (xem slide)
 - Luật thay thế phổ dụng.
 - Hợp nhất.
 - Luật Modus Ponens tổng quát.
 - Luật phân giải tổng quát.
 - Luật phân giải trên các câu Horn.

g. ***Predicate logic – first order.***

- Logic vị từ cấp 1 cho phép chúng ta biểu diễn các đối tượng trong thế giới thực với các tính chất của chúng và các quan hệ của chúng.
- Để biểu diễn tri thức của chúng ta về một miền các đối tượng nào đó trong logic vị từ cấp một, chúng ta cần đưa ra các ký hiệu
 - Các ký hiệu hằng để chỉ ra các đối tượng cụ thể.
 - Các ký hiệu biến để chỉ ra các đối tượng bất kỳ trên miền đối tượng,
 - Các ký hiệu hàm để biểu diễn quan hệ hàm.
 - Các ký hiệu vị từ để biểu diễn mối quan hệ khác nhau của các đối tượng
- Các ký hiệu đã nêu tạo thành hệ thống từ vựng về miền đối tượng mà chúng ta quan tâm.
- Sử dụng các từ vựng đã đưa ra, chúng ta sẽ tạo ra các câu trong logic vị từ cấp một để biểu diễn tri thức của chúng ta về miền đối tượng đó.
- Tập hợp tất cả các câu được tạo thành sẽ lập nên cơ sở tri thức trong hệ tri thức mong muốn.
- Ngoài ra, có thể sử dụng vị từ bằng, danh sách và các phép toán trên danh sách và tập hợp để biểu diễn tri thức mong muốn.

IX. ***Neural Network.***

1. ***Tổng quan về neural network.***

- Artificial Neural Network là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thần kinh sinh vật.
- Bao gồm số lượng lớn các Neural được gắn kết để xử lý thông tin.
- Artificial Neural Network giống như não người, được học bởi kinh nghiệm thông qua huấn luyện, có khả năng lưu trữ tri thức và sử dụng chúng trong việc dự đoán những dữ liệu chưa biết.

2. ***Natural neurons.***

- Natural neurons nhận tín hiệu thông qua các khớp thần kinh (synapses) trên các cấu trúc hình cây (dendrites) hoặc các màng (membrane) của tế bào thần kinh (neuron).
- Khi các tín hiệu nhận được đủ mạnh (vượt qua một ngưỡng nào đó), các neuron sẽ được kích hoạt và truyền một tín hiệu thông qua axon (nerve fibre).
- Tín hiệu này có thể được truyền đến synapse khác và có kích hoạt tiếp những neuron khác.

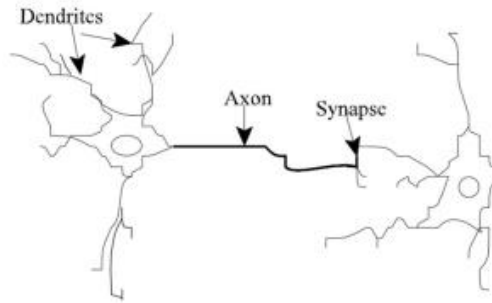


Figure 1. Natural neurons (artist's conception).

3. Cấu trúc của một neuron nhân tạo.

- Một neuron nhân tạo (artificial neuron) bao gồm ba thành phần chính:
 - Input (giống như synapses): được nhân bởi các trọng số (weights), mô tả độ lớn của tín hiệu.
 - Các inputs sau đó được tính toán thông qua một hàm toán học để xác định sự kích hoạt của neuron.
 - Mạng neuron nhân tạo sẽ kết hợp nhiều artificial neuron như thế để tiến hành xử lý thông tin. Kết quả xử lý của một neuron có thể làm input cho một neuron khác.

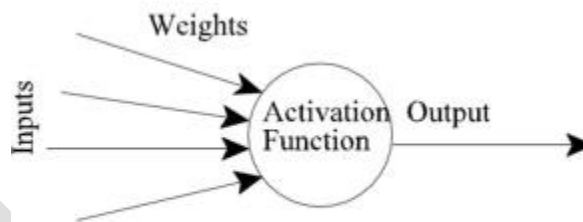
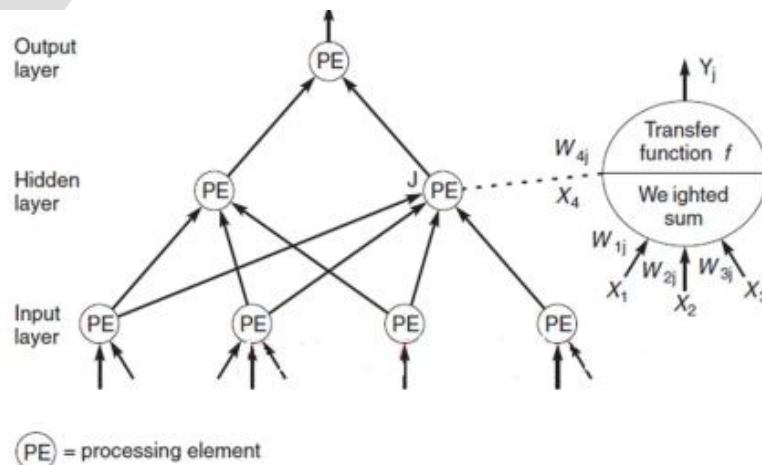
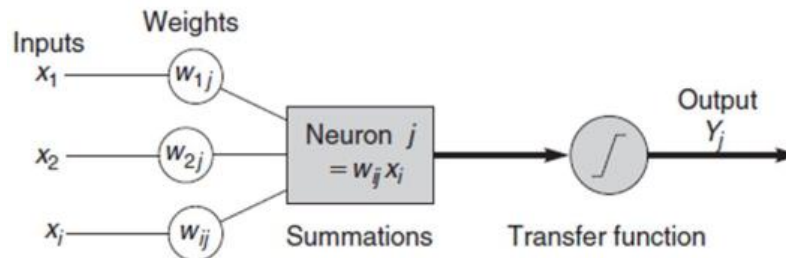


Figure 2. An artificial neuron

- Các đơn vị xử lý (processing elements) của một artificial neuron network là những neuron.
- Một artificial neuron network gồm 3 thành phần chính: input layer, hidden layer và output layer.



- Sau đây là mô hình chi tiết cho quá trình xử lý thông tin trên ANN:



4. Quá trình xử lý thông tin của ANN.

- Inputs: mỗi input tương ứng với một thuộc tính (attribute) của dữ liệu (patterns). Ví dụ: xét hệ thống đánh giá mức độ rủi ro cho vay trong ngân hàng, mỗi input là một thuộc tính của khách hàng như thu nhập, nghề nghiệp, giới tính, ...
- Outputs: là kết quả của Artificial Neural Networks hay một giải pháp cho một vấn đề. Ví dụ: trong bài toán xem xét chấp nhận cho khách hàng vay tiền trong ngân hàng sẽ là cho vay hay không cho vay...
- Trọng số liên kết (connection weights): là thành phần vô cùng quan trọng trong một hệ thống mạng neuron nhân tạo. Nó thể hiện mức độ quan trọng của dữ liệu đầu vào đối với quá trình xử lý thông tin (quá trình chuyển đổi dữ liệu giữa các layer).
- Hàm tổng (summation function): tính tổng trọng số của tất cả các input được đưa vào mỗi neuron. Một hàm tổng của một neuron đối với n input sẽ được tính theo công thức sau đây:

$$Y = \sum_{i=1}^n X_i W_i$$

- Hàm tổng với nhiều neurons trong cùng một layer:

$$Y_j = \sum_{i=1}^n X_i W_{ij}$$

- Hàm chuyển đổi (transformation function):
 - o Hàm tổng của một neuron cho chúng ta biết khả năng kích hoạt của neuron đó và còn gọi là kích hoạt bên trong (internal activation).
 - o Các neuron này có thể sinh ra một output hoặc không trong hệ thống mạng neuron nhân tạo hay nói cách khác output của một neuron có thể được chuyển đến layer tiếp theo trong mạng neuron hay không.
 - o Mỗi quan hệ giữa Internal Activation và kết quả (Output) được thể hiện bằng hàm chuyển đổi (transfer function).
 - o Việc lựa chọn hàm chuyển đổi có tác động lớn đến kết quả ANN. Hàm chuyển đổi phi tuyến hay sử dụng mạng neuron nhân tạo là sigmoid (logical activation) function.
 - o Kết quả của sigmoid function thuộc khoảng [0, 1] nên còn gọi là hàm chuẩn hóa (normalized function). Kết quả xử lý tại các neuron đôi khi rất lớn. Chính vì thế, hàm chuyển đổi được sử dụng để xử lý những kết quả này trước khi chuyển đến layer tiếp theo.

- Trong thực tế, thay vì sử dụng các hàm chuyển đổi đã nêu trên, người ta có thể sử dụng giá trị ngưỡng (threshold value) để kiểm soát các output của các neuron tại một layer nào đó trước khi chuyển các output này đến các layer tiếp theo. Nếu output của một neuron nào đó nhỏ hơn threshold thì nó sẽ không được chuyển đến layer tiếp theo.

X. *Fuzzy logic.*

- Logic mờ dựa trên ý tưởng rằng nhiều thông tin có thể được đánh giá, nhưng ở mức độ không rõ ràng.
 - o Nhiệt độ trong phòng hơi nóng.
 - o Cậu bé khá cao so với tuổi.
 - o Tốc độ của xe máy rất nhanh.
 - o Khoảng cách từ đây đến đây là xa.
 - o Cô gái kia trông đẹp
- Logic mờ (fuzzy logic) cho phép biểu diễn (diễn đạt) các thông tin không rõ ràng.

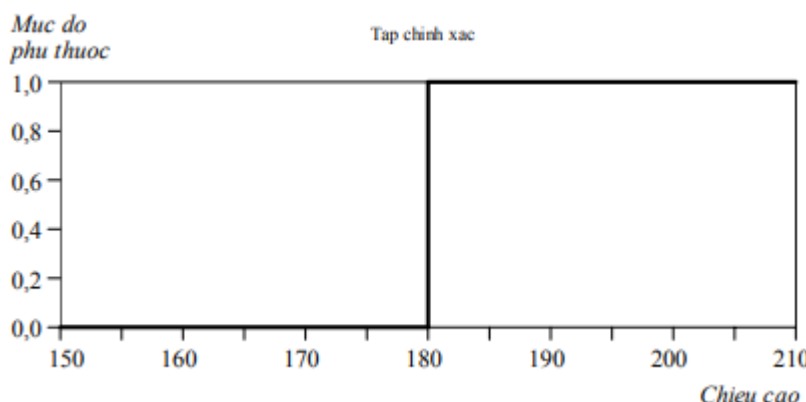
1. *Tập mờ (fuzzy set).*

- Khái niệm tập hợp là một khái niệm cơ bản của toán học.
 - Mỗi phần tử chỉ có thể thuộc hoặc không thuộc vào tập hợp.
- Logic mờ (fuzzy logic) dựa trên ý tưởng mỗi phần tử thuộc vào một tập hợp ở một mức độ nào đó (degree) nào đó.
 - o Ví dụ: Tập “*Những người đàn ông cao*”. Các thành phần của tập mờ “*Những người đàn ông cao*” là tất cả đàn ông, nhưng **mức độ phụ thuộc** (degree of membership) của các thành phần vào tập hợp thì tùy thuộc vào chiều cao của họ
- Logic mờ sử dụng các quy tắc (công thức) toán học cho phép biểu diễn tri thức dựa trên mức độ phụ thuộc.
 - o Hoàn toàn thuộc vào (hoàn toàn đúng) – 1 (True)
 - o Hoàn toàn không thuộc vào (hoàn toàn sai) – 0 (False)
 - o Thuộc vào ở một mức độ (đúng ở một mức độ) – $x \in (0, 1)$.

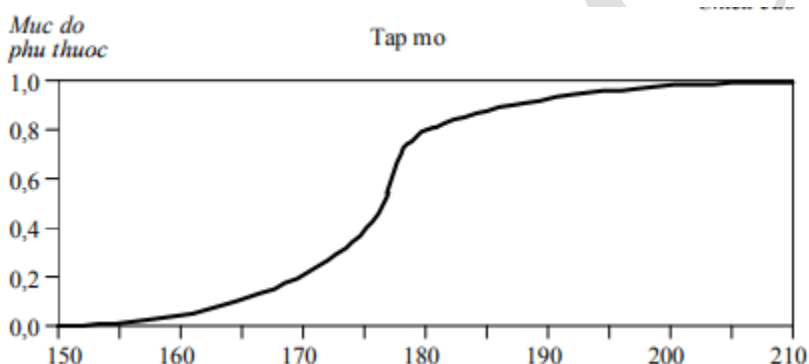
Name	Height, cm	Degree of Membership	
		<i>Crisp</i>	<i>Fuzzy</i>
Chris	208	1	1.00
Mark	205	1	1.00
John	198	1	0.98
Tom	181	1	0.82
David	179	0	0.78
Mike	172	0	0.24
Bob	167	0	0.15
Steven	158	0	0.06
Bill	155	0	0.01
Peter	152	0	0.00

2. *Tập chính xác và tập mờ.*

- Chiều tọa độ ngang (X) biểu diễn các giá trị (có thể) chiều cao của một người đàn ông.

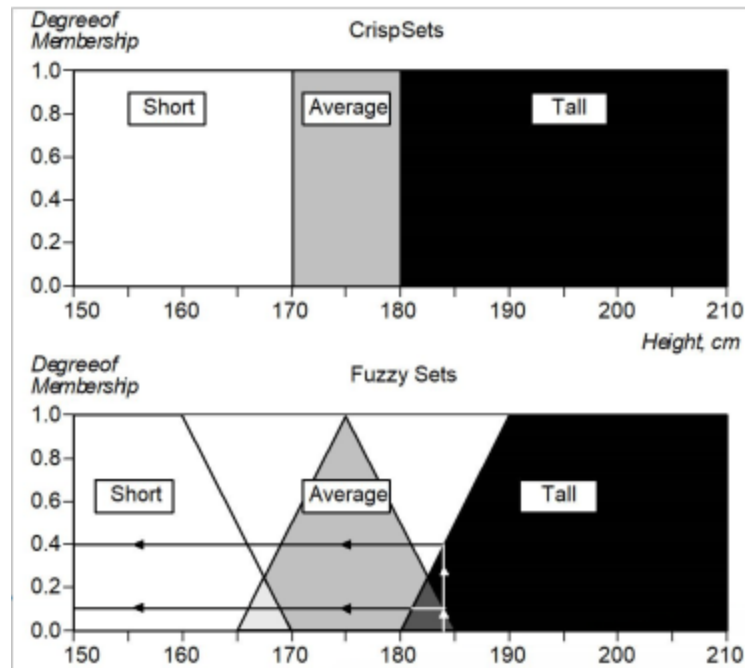


- Chiều tọa độ dọc (Y) biểu diễn mức độ phụ thuộc của tập mờ.



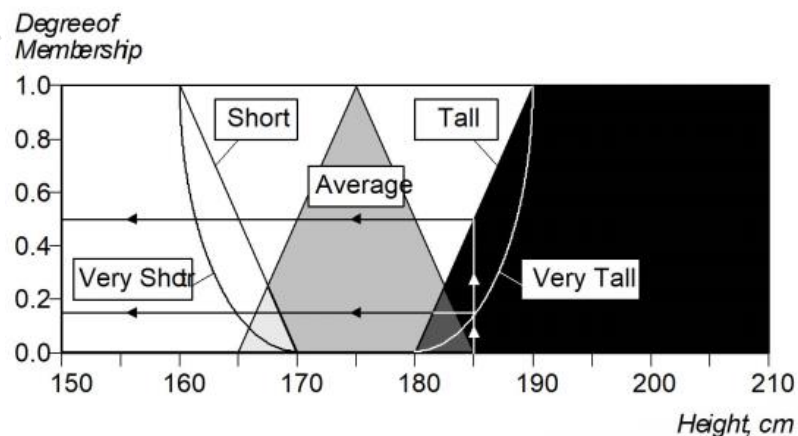
3. Các giới hạn mờ.

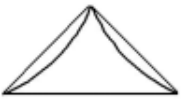



- Trong lý thuyết mờ, một tập mờ A của miền giá trị X được định nghĩa (được xác định) bởi hàm $\mu_A(x)$.
- $\mu_A(x)$ được gọi là hàm phụ thuộc (**membership function**) của tập mờ A.
 - $A = \{ \mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \dots, \mu_A(x_n)/x_n \}$
 - $\mu_A(x): X \rightarrow \{0, 1\}, \begin{cases} \mu_A(x) = 1, & \text{nếu } x \text{ hoàn toàn phụ thuộc trong } A \\ \mu_A(x) = 0, & \text{nếu } x \text{ không thuộc trong } A \\ 0 < \mu_A(x) < 1, & \text{nếu } x \text{ thuộc một phần trong } A \end{cases}$
- Đối với mỗi phần tử (giá trị) x của miền giá trị X, hàm phụ thuộc $\mu_A(x)$ chỉ ra mức độ tương ứng mà x là một thành phần của A.
- Mức độ này (là một giá trị trong khoảng từ 0 đến 1) biểu diễn mức độ phụ thuộc của phần tử x trong A.







4. Linguistic variables and hedges.

- Trong hệ thống chuyên gia mờ, các biến ngôn ngữ (linguistic) được sử dụng trong các quy tắc mờ, ví dụ:
 - IF wind is strong THEN sailing is good.
 - IF project_duration is long THEN completion_risk is high.
 - IF speed is slow THEN stopping_distance is short.
- Phạm vi giá trị có thể của một biến ngôn ngữ biểu thị vũ trụ diễn ngôn của biến đó. Ví dụ:
 - Vũ trụ diễn ngôn về tốc độ biến ngôn ngữ có thể phạm vi trong khoảng từ 0 đến 220 km/h và có thể bao gồm các tập hợp con mờ như rất chậm, chậm, trung bình, nhanh và rất nhanh.
- Biến ngôn ngữ mang theo khái niệm hạn định về tập mờ, gọi là hàng rào (hedges).
- Hàng rào (hedges) là thuật ngữ dùng để sửa đổi hình dạng của tập mờ. Chúng bao gồm các trạng từ như rất, hơi, khá, nhiều hoặc ít, hơi hơi, v.v.. (very, somewhat, quite, more or less and slightly).



<i>Hedge</i>	<i>Mathematical Expression</i>	<i>Graphical Representation</i>
A little	$[\mu_A(x)]^{1.3}$	
Slightly	$[\mu_A(x)]^{1.7}$	
Very	$[\mu_A(x)]^2$	
Extremely	$[\mu_A(x)]^3$	

<i>Hedge</i>	<i>Mathematical Expression</i>	<i>Graphical Representation</i>
Very very	$[\mu_A(x)]^4$	
More or less	$\sqrt{\mu_A(x)}$	
Somewhat	$\sqrt{\mu_A(x)}$	
Indeed	$\begin{aligned} &2 [\mu_A(x)]^2 \\ &\text{if } 0 \leq \mu_A \leq 0.5 \\ &1 - 2 [1 - \mu_A(x)]^2 \\ &\text{if } 0.5 < \mu_A \leq 1 \end{aligned}$	

5. *Phần bù (Complement).*

- Tập chính xác (crisp set): Phần tử không phụ thuộc vào tập hợp .
- Tập mờ (fuzzy set): Mức độ một phần tử không thuộc vào tập hợp.
- Nếu A là một tập mờ, thì phần bù của A (ký hiệu là $\neg A$) được định nghĩa như sau:

$$\mu_{\neg A}(x) = 1 - \mu_A(x) ; \text{ với mọi phần tử } x.$$

6. *Tập bao hàm (Container).*

- Tập chính xác: Những tập là tập con (subset) của tập khác.
- Trong lý thuyết mờ, nếu tập A là một tập con của B , thì:

- $\mu_A(x) \leq \mu_B(x) \forall x$
- Mỗi thành phần sẽ có mức độ phụ thuộc (membership value) vào tập A nhỏ hơn hoặc bằng mức độ phụ thuộc vào tập B.
- Ví dụ: A là tập “Những người đàn ông rất cao”, B là tập “Những người đàn ông cao”.

7. Giao (Intersection).

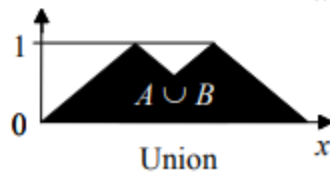
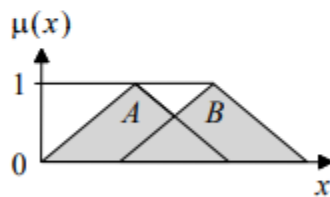
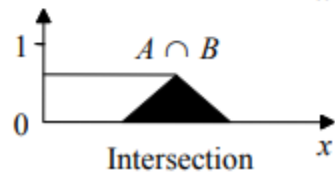
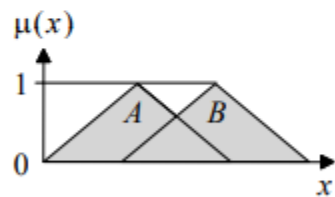
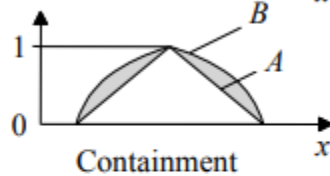
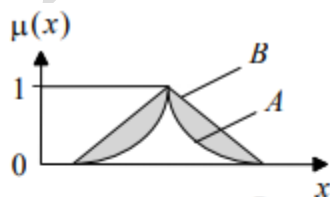
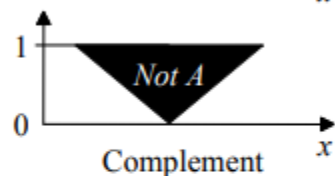
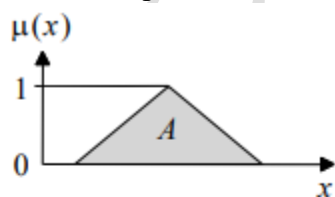
- Tập chính xác: Những phần tử thuộc vào cả 2 tập.
- Tập mờ: Mức độ mỗi phần tử thuộc vào cả 2 tập.
- Phần giao mờ (fuzzy intersection) được xác định bởi *giá trị phụ thuộc thấp nhất* đối với 2 tập mờ.
- Giao cả 2 tập mờ cũng là 1 tập mờ, được định nghĩa như sau:

$$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \cap \mu_B(x), \forall x$$

8. Hợp (Union).

- Tập chính xác: Những phần tử thuộc vào 1 trong 2 tập.
- Tập mờ: Mức độ mỗi phần tử thuộc vào 1 trong 2 tập.
- Phần hợp mờ (fuzzy union) được xác định bởi *giá trị phụ thuộc cao nhất* đối với 2 tập mờ.
- Hợp của 2 tập mờ cũng là 1 tập mờ, được định nghĩa như sau:

$$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \cup \mu_B(x), \forall x$$



XI. *Fuzzy rules.*

- Năm 1973, Lotfi Zadeh xuất bản tác phẩm có ảnh hưởng thứ hai của ông giấy. Bài viết này đưa ra một cách tiếp cận mới để phân tích các hệ thống phức tạp, trong đó Zadeh đề xuất nắm bắt kiến thức của con người về các luật mờ.
- Một luật mờ có thể được định nghĩa là một luật có điều kiện được phát biểu dưới dạng:
IF x is A – THEN y is B.
 - o Trong đó, x và y là các biến ngôn ngữ; A và B là các giá trị ngôn ngữ được xác định bởi các tập mờ trên vũ trụ diễn ngôn X và Y tương ứng.
- Ví dụ, chúng ta có thể biểu diễn quy tắc khoảng cách dừng ở dạng mờ:

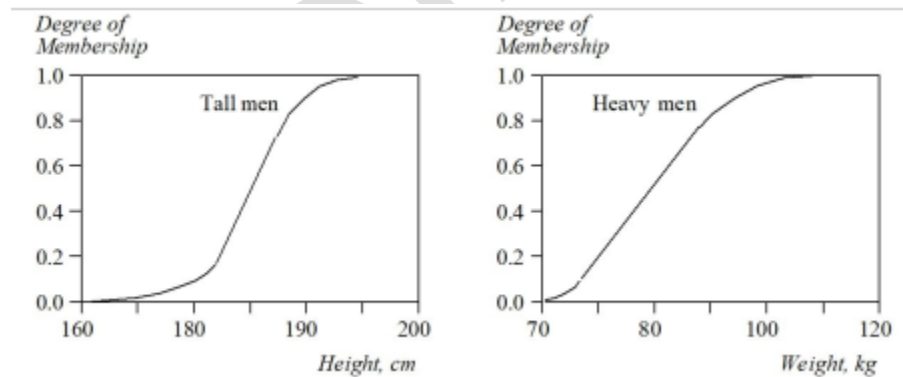
Rule: 1

**IF speed is fast
THEN stopping_distance is long**

Rule: 2

**IF speed is slow
THEN stopping_distance is short**

- Trong fuzzy rules, biến ngôn ngữ speed cũng có phạm vi (speed của tập vũ trụ diễn ngôn) trong khoảng từ 0 tới 220 km/h, nhưng phạm vi này bao gồm các tập mờ, chẳng hạn như chậm, trung bình và nhanh. . Biến ngôn ngữ stopping_distance có thể là trong khoảng 0 đến 300m và có thể bao gồm các tập mờ như ngắn, trung và dài.
- Luật mờ liên quan đến tập mờ.
 - Trong hệ thống mờ, tất cả các quy tắc đều có hiệu lực ở một mức độ nào đó hoặc ở mức độ khác. Nếu tiền đề đúng với một mức độ phụ thuộc thì hệ quả cũng đúng đến một mức độ tương tự.
 - Ví dụ:



→ Những tập mờ này cung cấp cơ sở cho việc ước lượng trọng số mô hình. Mô hình này dựa trên mối quan hệ giữa chiều cao và cân nặng của người đàn ông.

→ **IF height is tall THEN weight is heavy**

→ Monotonic selection

A fuzzy rule can have multiple antecedents, for example:

IF **project_duration** is long
AND **project_staffing** is large
AND **project_funding** is inadequate
THEN risk is high

IF **service** is excellent
OR **food** is delicious
THEN tip is generous

The consequent of a fuzzy rule can also include multiple parts, for instance:

IF **temperature** is hot
THEN **hot_water** is reduced;
 cold_water is increased

XII. Data – mining process

- Trích xuất thông tin từ tập dữ liệu và chuyển đổi nó thành một cấu trúc dễ hiểu để sử dụng tiếp.
- Đề khám phá các mẫu và mối quan hệ trong dữ liệu để giúp đưa ra các quyết định kinh doanh tốt hơn.
- Databases ngày nay có thể có kích thước lên tới hàng terabyte.
- Trong khối dữ liệu này có rất nhiều thông tin quan trọng và ẩn giấu.

The screenshot shows a presentation slide titled 'Hadoop in the Real World' with a large 'Hadoop' watermark in the background. It lists several companies and their Hadoop implementations:

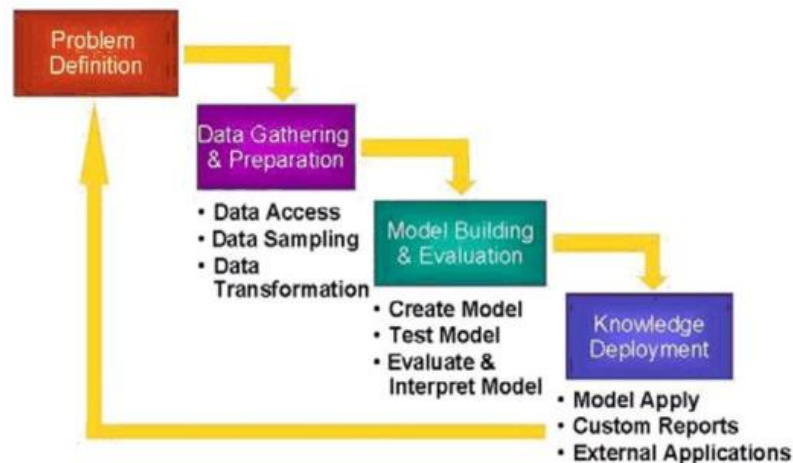
- EBay**
 - 532 nodes cluster (8 * 532 cores, 5.3PB).
 - Heavy usage of Java MapReduce, Pig, Hive, HBase
 - Using it for Search optimization and Research.
- Facebook**
 - We use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and more.
 - Currently we have 2 major clusters:
 - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
 - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
 - Each (commodity) node has 8 cores and 12 TB of storage.
 - We are heavy users of both streaming as well as the Java APIs. We have built a higher level data warehousing framework (<http://hadoop.apache.org/hive/>). We have also developed a FUSE implementation over HDFS.
- Spotify**
 - We use Hadoop for content generation, data aggregation, reporting and analysis (see more: [Hadoop at Spotify](#))
 - 690 node cluster = 8280 physical cores, 38TB RAM, 28 PB storage
 - +7,500 daily Hadoop jobs (scheduled by Luigi, our home-grown and recently open-sourced job scheduler - [code](#) and [video](#))

Handwritten notes on the slide include:

- shine_production 16.1806640625 GB
- 5.5 PB = 5500 TB = 5500.000 GB

- Market segmentation (phân khúc thị trường): xác định những đặc điểm chung của khách hàng mua sản phẩm tương tự từ công ty.
- Customer churn (khách hàng rời bỏ): dự đoán khách hàng nào có khả năng rời khỏi việc mua hàng từ công ty chúng ta và đến công ty đối thủ.
- Interactive marketing: dự đoán những gì mỗi cá nhân truy cập trang Web công ty quan tâm, xem sản phẩm nào, v.v..

- Market basket analysis (phân tích thị trường): hiểu sản phẩm hoặc dịch vụ thường được mua cùng nhau.
- Automated prediction of trends and behaviors (dự đoán tự động các xu hướng và hành vi): khai thác dữ liệu tự động hóa quá trình tìm kiếm thông tin dự đoán trong một database lớn.



- Automated discovery of previously unknown patterns (tự động phát hiện các mẫu chưa biết trước đó): công cụ khai thác dữ liệu quét qua database và xác định các mẫu ẩn trước đó:
 - o Ví dụ: phân tích dữ liệu bán hàng của một công ty để xác định những sản phẩm dường như không liên quan nhưng thường được mua cùng nhau.
- Thường mất từ 50% đến 90% thời gian và công sức của toàn bộ quá trình khám phá tri thức.
- Xác định các lĩnh vực quan trọng nhất trong việc dự đoán kết quả và xác định giá trị dẫn xuất nào đó có thể hữu ích.
- Đây là bước chuẩn bị dữ liệu cuối cùng trước khi xây dựng mô hình.
- Có bốn phần chính:
 - o Chọn các biến.
 - o Chọn hàng.
 - o Xây dựng các biến mới.
 - o Chuyển đổi các biến.
- Khám phá các mô hình thay thế để tìm ra mô hình tốt nhất trong việc giải quyết vấn đề kinh doanh.
- Chọn loại mô hình để đưa ra dự đoán.
- Yêu cầu quy trình đào tạo và có giao thức xác nhận (học có giám sát) để có kết quả chính xác và tốt nhất.
- Đánh giá mô hình và giải thích ý nghĩa kết quả của nó.
- Tỷ lệ chính xác chỉ áp dụng cho mô hình được xây dựng.
- Khi mô hình khai thác dữ liệu được xây dựng và xác nhận, nó có thể được sử dụng để đề xuất các hành động hoặc áp dụng mô hình cho các hoạt động khác nhau cho bộ dữ liệu (dataset).