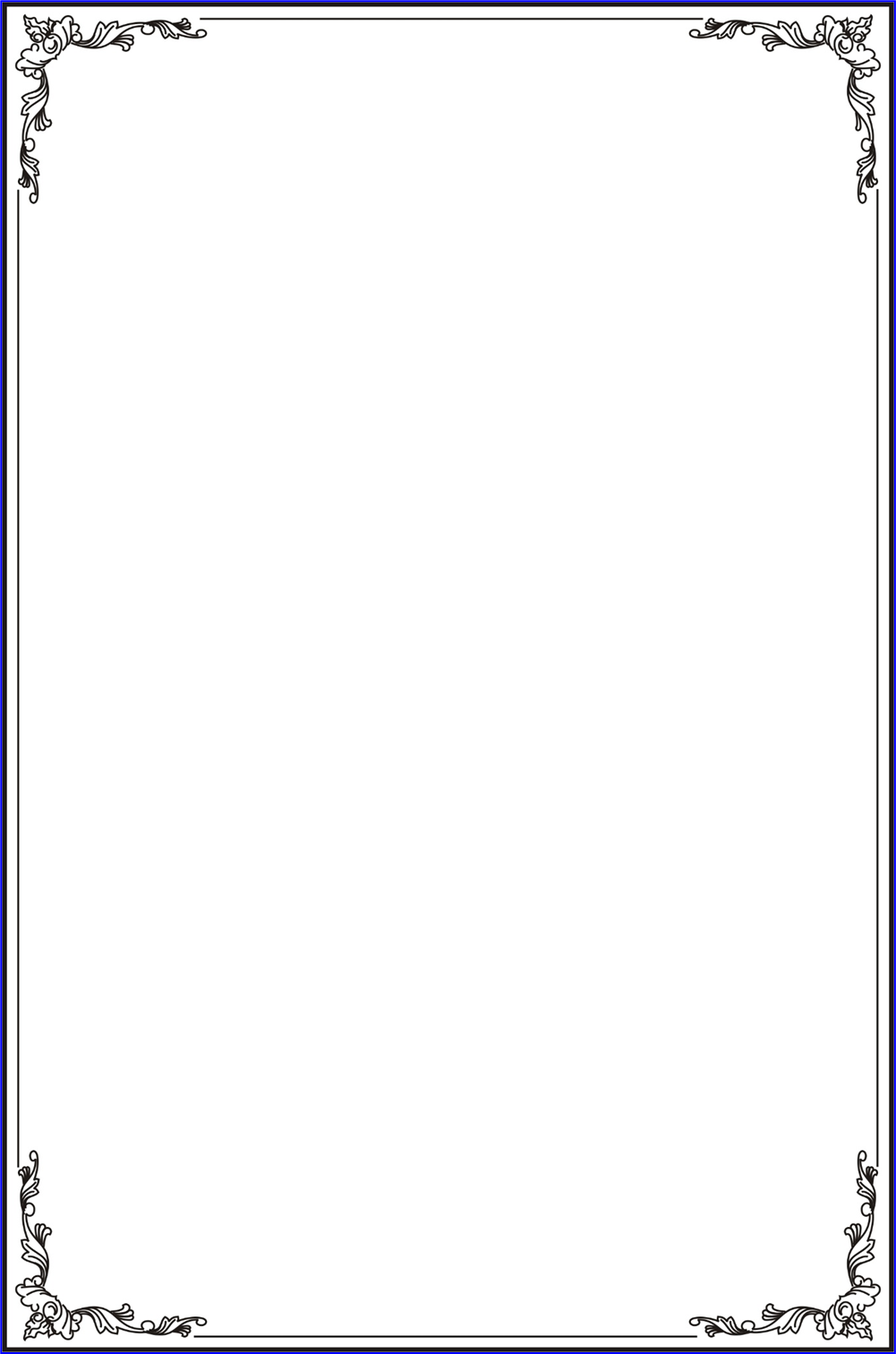
****

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**🕯✡🕮🕮✡🕯**

****

|  |  |
| --- | --- |
| ***Nhóm sinh viên thực hiện :*** | |
| Nguyễn Minh Chiến | 20110262 |
| Đỗ Minh Dũng | 20110620 |
| Lê Hải Đăng | 20110243 |
| Mai Hải Đăng | 20110244 |
| Trần Trung Hậu | 20110642 |

|  |  |
| --- | --- |
| Nguyễn Đức Khải | 20110655 |
| Lê Nguyễn Tuấn Kiệt | 20110286 |
| Trần Minh Mẫn | 20110301 |
| Nguyễn Minh Sơn | 20110713 |
| Nguyễn Đức Thành | 20110307 |

**LỜI NÓI ĐẦU**

**Giảng viên : Trần Nhật Quang**

**Báo cáo cuối kỳ:**

**TRÍ TUỆ NHÂN TẠO**

**TP. Hồ Chí Minh, tháng 05 năm 2022**

Trí tuệ nhân tạo đã xuất hiện được một thời gian dài, nhưng đến hiện tại thì vẫn chưa được phổ biến và xã hội hóa. Vậy nên dù có khá nhiều thông tin nhưng tìm hiểu về AI vẫn không phải là dễ. Qua bản báo cáo này, nếu có gì sai sót mong thầy thông cảm và góp ý để chúng em hoàn thiện hơn.

Để hoàn thành được báo cáo về môn học Trí tuệ nhân tạo, chúng em không thể quên gửi lời cảm ơn đến Giảng viên. Thầy Trần Nhật Quang đã cung cấp nguồn kiến thức và tài liệu để chúng em học tập, tìm hiểu.

Chúng em xin chân thành cảm ơn!

Tập thể nhóm

|  |  |
| --- | --- |
| **Điểm** | **Đánh giá của giảng viên** |
|  |  |

MỤC LỤC

[MỤC LỤC 4](#_Toc104270534)

[I. ĐẠO VĂN 7](#_Toc104270535)

[1. Đạo văn là gì? 7](#_Toc104270536)

[2. Những điều nên/không nên làm khi làm báo cáo: 7](#_Toc104270537)

[3. Cam kết: 7](#_Toc104270538)

[II. TÌM HIỂU VỀ TRÍ TUỆ NHÂN TẠO (ARTIFICIAL INTELLIGENCE) 9](#_Toc104270539)

[1. Trí tuệ nhân tạo là gì ? 9](#_Toc104270540)

[2. Lịch sử hình thành AI 10](#_Toc104270541)

[2.1. Alan Turing – cha đẻ của trí tuệ nhân tạo. 10](#_Toc104270542)

[2.2. Hội thảo Dartmouth – sự kiện bắt đầu cho tất cả. 10](#_Toc104270543)

[2.3. Thuật ngữ "artificial intelligence"và hội nghị Dartmouth (1955–1956) 11](#_Toc104270544)

[III. ĐẠO ĐỨC TRÍ TUỆ NHÂN TẠO 11](#_Toc104270545)

[IV. THUẬT TOÁN TÌM KIẾM CƠ BẢN 12](#_Toc104270546)

[1. Các bước cơ bản của một thuật toán tìm kiếm 12](#_Toc104270547)

[2. Tiêu chí đánh giá thuật toán 12](#_Toc104270548)

[3. Conceptual algorithm 13](#_Toc104270549)

[3.1. Conceptual algorithm #1 Tree search (cây tìm kiếm) 13](#_Toc104270550)

[3.2. Conceptual algorithm #2 Graph search (search with memory) 14](#_Toc104270551)

[4. Hai loại thuật toán tìm kiếm 15](#_Toc104270552)

[5. Các cấu trúc dữ liệu 15](#_Toc104270553)

[5.1. Thành phần của một Node 15](#_Toc104270554)

[5.2 Queue 15](#_Toc104270555)

[6. Các thuật toán Uninformed search 16](#_Toc104270556)

[6.1. Thuật toán Breadth-ﬁrst search (BFS) 16](#_Toc104270557)

[6.2. Thuật toán Uniform-cost search 17](#_Toc104270558)

[6.3. Thuật toán Depth-ﬁrst search (DFS) 18](#_Toc104270559)

[6.4. Thuật toán Depth-limited search 18](#_Toc104270560)

[6.5. Thuật toán Iterative deepening search 20](#_Toc104270561)

[7. Các thuật toán Informed search 20](#_Toc104270562)

[7.1. Các hàm ước lượng quảng đường 20](#_Toc104270563)

[7.2. Ý tưởng chung của Informed search 20](#_Toc104270564)

[7.3. Thuật toán Best-first search 21](#_Toc104270565)

[7.4. Thuật toán A\* search 21](#_Toc104270566)

[7.5. Cách tạo heuristic function 21](#_Toc104270567)

[7.6. So sanh hàm Heuristic. 22](#_Toc104270568)

[V. LOCAL SEARCH – SEARCHING FOR GOAL STATE 23](#_Toc104270569)

[1. So sánh với các thuật toán tìm kiếm trước đó: 23](#_Toc104270570)

[2. State space landscape: 23](#_Toc104270571)

[3. Local search: 24](#_Toc104270572)

[4. Hill-climbing search: 24](#_Toc104270573)

[5. Issues of Hill-climbing search: 25](#_Toc104270574)

[5.1. Quá nhiều successors: 25](#_Toc104270575)

[5.2. Local optimum: 25](#_Toc104270576)

[6. Simulated Annealing: 27](#_Toc104270577)

[6.1. Thuật toán Simulated Annealing. 27](#_Toc104270578)

[7. Searching in NONDETERMINISTIC environments 28](#_Toc104270579)

[7.1. Tìm kiếm trong môi trường không xác định 28](#_Toc104270580)

[7.2. And\_Or Graph Search 28](#_Toc104270581)

[8. Searching in Partially Observable Enviroments 30](#_Toc104270582)

[8.1. No observable 30](#_Toc104270583)

[8.2. Partially observable 31](#_Toc104270584)

[8.3. Demo 31](#_Toc104270585)

[9. Online Search 32](#_Toc104270586)

[9.1. So sánh về offline search và online search : 32](#_Toc104270587)

[9.2. Competitive ratio : 33](#_Toc104270588)

[9.3. Online Depth-first search agent : 33](#_Toc104270589)

[9.4. Online A\* search : ( tìm được solution tốt một cách nhanh chóng ) 34](#_Toc104270590)

[9.5. Learning Real – Time A\* agent : 34](#_Toc104270591)

[10. Constraint Satisfactoin Problems 36](#_Toc104270592)

[10.1. Constraint satisfaction problems (CSP) 36](#_Toc104270593)

[10.2. Constraint graph : 36](#_Toc104270594)

[10.3. Constraint propagation : 36](#_Toc104270595)

[10.4. Local consistency (node , arc , path ) 36](#_Toc104270596)

[10.5. AC – 3 algorithm (arc consistency) 36](#_Toc104270597)

[10.6. Path consistency : 38](#_Toc104270598)

[10.7. K-consistency : (dạng tổng quát của local consistency) 38](#_Toc104270599)

[10.8. Global constraints : 38](#_Toc104270600)

[10.9. Solving a Sodoku puzzle : (AC – 3, Triplets ) 38](#_Toc104270601)

[10.10. Backtracking search algorithm 38](#_Toc104270602)

[10.11. Comments on backtracking search algorithm : 40](#_Toc104270603)

[10.12. Min-conflict algorithm. 40](#_Toc104270604)

[10.13. Tree CSP solver. 41](#_Toc104270605)

[11. Reinforcement Learning (RL). 42](#_Toc104270606)

[11.1. Khái niệm. 42](#_Toc104270607)

[11.2. Các thuật ngữ sử dụng trong Reinforcement Learning. 42](#_Toc104270608)

[11.3. Ưu - nhược điểm của Reinforcement Learning. 43](#_Toc104270609)

[11.4. Các ứng dụng của Reinforcement Learning. 43](#_Toc104270610)

[11.5. Thư viên OpenAI Gym 43](#_Toc104270611)

[11.6. Cross-entropy method 43](#_Toc104270612)

[VI. GAME “MARIO SOKOBAN” 44](#_Toc104270613)

[1. Lý do chọn Mario sokoban: 44](#_Toc104270614)

[2. Giao diện, hướng dẫn sử dụng project 44](#_Toc104270615)

[3. Cấu trúc project. 48](#_Toc104270616)

[4. Các class, function: 50](#_Toc104270617)

[4.1. sokubanProblem: 50](#_Toc104270618)

[4.1.1. Hàm actions 50](#_Toc104270619)

[4.1.2. Hàm result. 51](#_Toc104270620)

[4.1.3. Hàm goal\_test 52](#_Toc104270621)

[4.2. Class BFSNode 52](#_Toc104270622)

[4.2.1. Hàm expand 53](#_Toc104270623)

[4.2.2. Hàm child\_node 53](#_Toc104270624)

[4.2.3. Hàm solution 53](#_Toc104270625)

[4.3. Class AstarNode 53](#_Toc104270626)

[4.3.2. Hàm cal\_heuristic: 54](#_Toc104270627)

[4.3.3. Hàm cal\_f 54](#_Toc104270628)

[4.4. Hàm Bfs 55](#_Toc104270629)

[4.5. Hàm astar 56](#_Toc104270630)

[TÀI LIỆU THAM KHẢO 57](#_Toc104270631)

I. ĐẠO VĂN

# 1. Đạo văn là gì?

Đạo văn (Plagiarism) là hành vi của một cá nhân, tổ chức chiếm hữu, công bố sai trái một sản phẩm nhưng có sự trùng lặp với sản phẩm của tổ chức, cá nhân khác về các yếu tố như ngôn ngữ, ý tưởng, suy nghĩ, cách diễn đạt nhưng được sự cho phép của người đó.

Đạo văn được xem là một hành vi thiếu trung thực về mặt học thuật và vi phạm đạo đức. Cá nhân, tổ chức có hành vi đạo văn có thể bị xã hội lên án và chỉ trích. Về mặt pháp luật, có thể bị cảnh cáo hoặc phạt tiền.

Một số hình thức của đạo văn phổ biến: đạo văn toàn bộ, đạo văn quên dẫn nguồn, đạo văn dịch, đạo văn quên dùng dấu ngoặc kép,...

# 2. Những điều nên/không nên làm khi làm báo cáo:

Code cũng là một sản phẩm trí tuệ của người khác. Vì vậy, khi làm báo cáo cần tránh những việc sau đây:

* Sử dụng code của người khác mà không ghi hoặc ghi sai nguồn.
* Không sao chép toàn bộ ý tưởng, code của người khác. Phải biểu diễn kết quả theo cách hiểu của bản thân.
* Tìm hiểu thật sâu và hiểu thật rõ một vấn đề nào đó nào để hạn chế dựa dẫm vào những sản phẩm sẵn có.

# 3. Cam kết:

Chúng em xin cam đoan đồ án này do các thành viên nhóm thực hiện. Nếu phạm lỗi đạo văn (như: sử dụng tài liệu, code của người khác mà không ghi rõ nguồn, sao chép trên 30% báo cáo), chúng em xin chịu hoàn toàn trách nhiệm.

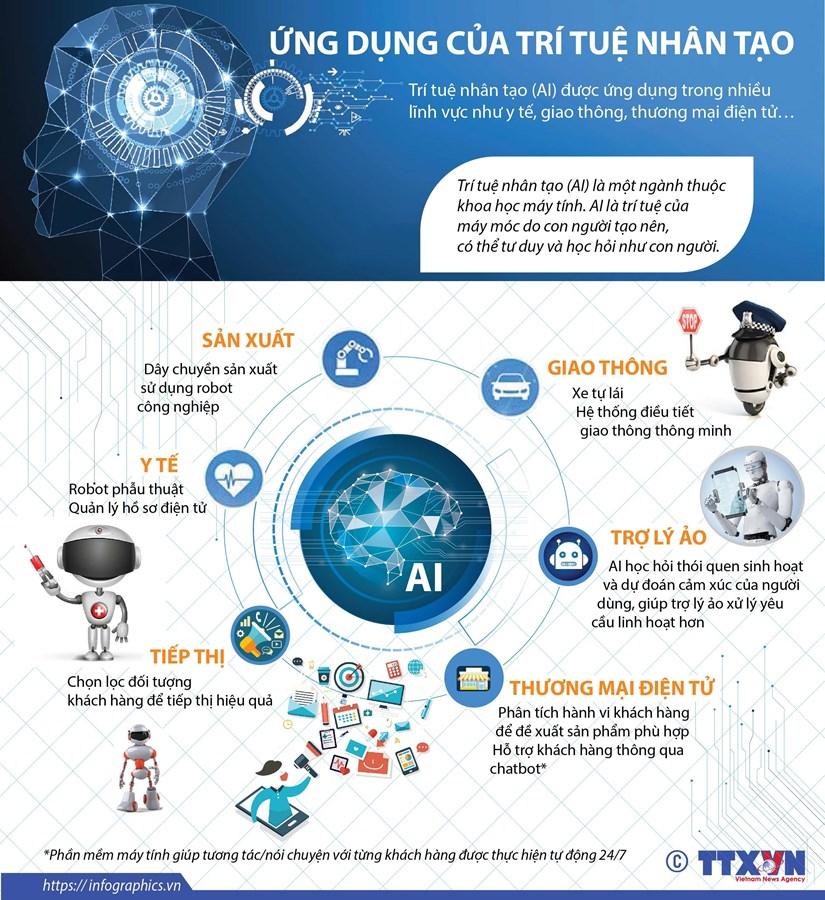
|  |  |
| --- | --- |
| **Họ tên** | **MSSV** |
| Nguyễn Minh Chiến | 20110262 |
| Đỗ Minh Dũng | 20110620 |
| Lê Hải Đăng | 20110243 |
| Mai Hải Đăng | 20110244 |
| Trần Trung Hậu | 20110642 |
| Nguyễn Đức Khải | 20110655 |
| Lê Nguyễn Tuấn Kiệt | 20110286 |
| Trần Minh Mẫn | 20110301 |
| Nguyễn Minh Sơn | 20110713 |
| Nguyễn Đức Thành | 20110307 |

II. TÌM HIỂU VỀ TRÍ TUỆ NHÂN TẠO (ARTIFICIAL INTELLIGENCE)

# 1. Trí tuệ nhân tạo là gì ?

Trí tuệ nhân tạo (AI) là một nhánh rộng lớn của khoa học máy tính liên quan đến việc xây dựng các máy móc, hệ thống có khả năng nhận biết môi trường và thực hiện các hành động nhằm tối ưu hóa cơ hội để đạt được mục đích.

Các ứng dụng AI phổ biến:

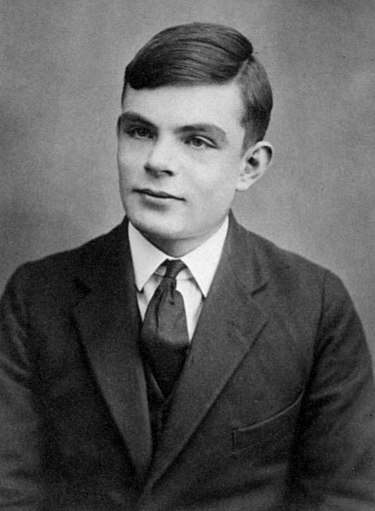


Nguồn ảnh : Tầm quan trọng của AI, http://jannguyen.com/ai---cong-nghe-chu-dao-tren-cac-thiet-bi-dien-tu-cua-tuong-lai/806/148

# 2. Lịch sử hình thành AI

## 2.1. Alan Turing – cha đẻ của trí tuệ nhân tạo.

Trong nửa đầu thế kỷ XX, cả thế giới đã quá quen thuộc với khái niệm robot khi mà các tác phẩm nghệ thuật đã cho ra mắt những cỗ máy mang hình dáng con người có thể tự động và có thể làm hành động giống con người. Sự xuất hiện của Robot được bắt đầu bởi Nick Chopper (Người rừng Thiếc) trong bộ phim Vùng đất hư cấu của xứ Oz (1900) rồi đến Maschinenmensch trong phim Metropolis (1927).



**Alan Turing (1912–1954)**

Từ đây, các tác phẩm nghệ thuật đã chi phối nhận thức của con người về robot. Có lẽ nó đã phần nào khơi gợi niềm cảm hứng cho các nhà khoa học về những cỗ máy có thể thay thế họ giải quyết những vấn đề trong cuộc sống.

Năm 1950, Alan Turing đã đưa ra một câu hỏi: “Liệu một cỗ máy có thể bắt chước trí thông minh của con người không ?”. Câu hỏi này như một ngọn lửa châm ngòi cho một sự ra mắt của một bộ môn khoa học mới mang tên Trí tuệ nhân tạo (AI).

Cùng năm đó, Turing cho ra mắt “Phép thử Turing” trong bài viết Máy tính và trí tuệ. Phép thử là một bài kiểm tra khả năng trí tuệ của máy tính. The phép thử, sẽ có một máy tính (A) và một người (B) sẽ giao tiếp với một người (C) để cố chứng minh mình là con người. Nếu C không thể nhận ra máy tính không phải là con người thì phép thử thành công.

## 2.2. Hội thảo Dartmouth – sự kiện bắt đầu cho tất cả.

The Dartmouth Summer Research Project (DSRPAI) về Trí tuệ Nhân tạo là một thảo thảo mùa hè đã diễn ra khoảng 8 tuần vào năm 1956 tại trường Cao đẳng Dartmouth, Hanover, New Hampshire. Trước đó, vào năm 1955, John McCarthy – người đã tổ chức DSRPAI đã đặt ra thuật ngữ “trí tuệ nhân tạo” và nó cũng là chủ đề của hội thảo này. Cũng chính tại hội thảo này, AI đã có được sự công nhận, có tầm nhìn và sứ mệnh riêng của nó.

Hội thảo có sự góp mặt của tiến sĩ Marvin Minsky, ông Allen Newell, giáo sự Herbert Simon – những người được xem là cha đẻ của Trí tuệ nhân tạo cùng với John McCarthy và Alan Turing.

Sau đó, hội thảo đã góp phần thuyết phục các cơ quan chính phủ như DARPA tài trợ AI tại một số tổ chức. Từ năm 1957 đến 1974, AI phát triển mạnh mẽ và đã có chỗ đứng vững chắc trong xã hội. Máy tính có thể lưu trữ nhiều thông tin hơn, các thuật toán cũng được cải thiện và được ứng dụng rộng rãi trong nhiều lĩnh vực. Tuy nhiên, vẫn còn một chặng đường dài phía trước để đạt được sự đỉnh cao và chạm đến giới hạn của bộ môn khoa học này.

## 2.3. Thuật ngữ "artificial intelligence"và hội nghị Dartmouth (1955–1956)

Cùng với Alan Turing, Marvin Minsky, Allen Newell, Herbert A. Simon, **John McCarthy** là một trong những người cha sáng lập nên Trí tuệ nhân tạo. McCarthy đặt ra thuật ngữ "artificial intelligence" vào năm 1955, đó cũng chính là chủ đề của hội nghị Dartmouth vào mùa hè năm 1956 do ông tổ chức.

Hội nghị Dartmouth có lẽ là workshop đầu tiên về Trí tuệ nhân tạo và hình thành nên khoa học nghiên cứu về Trí tuệ nhân tạo.

Các nhà nghiên cứu từ Đại học Carnegie Mellon, Học viện Công nghệ Massachusetts, các nhân viên từ công ty IBM đã gặp nhau tại hội nghị và cùng nhau nghiên cứu về thành lập các phòng nghiên cứu Trí tuệ nhân tạo.

Một vài năm sau đó, những nghiên cứu về Trí tuệ nhân tạo đạt được thành công vượt bậc. Vào thời điểm này, tất cả các chuyên gia về Trí tuệ nhân tạo đều vô cùng lạc quan về sự phát triển trong tương lai của công nghệ này.

Sau đó Trí tuệ nhân tạo cũng có những giai đoạn chữg lại như những năm 1970, 1987-1997.

Thế kỷ 21 hứa hẹn là thời gian phát triển mạnh mẽ nhất của Trí tuệ nhân tạo với những ứng dụng như Big Data, Deep Learning và ứng dụng thương mại rộng rãi …

III. ĐẠO ĐỨC TRÍ TUỆ NHÂN TẠO

Thời đại phát triển, công nghệ đang và dần thay thế sức người trong mọi việc. Không thể Trong bối cảnh ngày nay, khi mà AI đang ngày càng phát triển, nó đã mang lại nhiều lợi ích cho con người. Song, nhiều vấn đề vẫn được nhiều đặt ra về sự nguy hiểm của AI. Vì thế trong lĩnh vực AI cần trú trọng và trau dồi về nhận thức đạo đức.

Dưới đây là ba trường phái đạo đức tiêu biểu.Các trường phái đạo đức của AI:

**Virtue ethics:**

Đạo đức học đức hạnh là một trường phái đạo đức, tập trung vào tính cách vốn có của con người hơn là vào một động cụ thể. Những chuẩn mực của trường phái gồm trung thực, khoan dung, nhân ái, đức hạnh,… Đây là những tính cách khởi đầu của những tích cách khác bên trong một con người, vì vậy trường phái này chỉ đủ dùng trong những thời kì đầu của nền văn minh nhân loại.

Khi đến giai đoạn công nghiệp phát triển, tính sở hữu được đề cao và các vấn đề cũng bắt đầu được hình thành. Virtue ethics đã không đủ khả năng kiểm soát, lúc này một trường phái đạo đức mới ra đời.

**Deontology:**

Đạo đức sinh học là một trường phái đạo đức dựa trên luật lệ. Đối với trường phái này, một hành động đúng luật thì được xem có đạo đức và một hành động trái luật thì được xem là không có đạo đức.

Deotology đã phổ biến và được nhiều người sử dụng. Tuy nhiên, nó khá cứng nhắc và đã có tình huống không được xử lý công bằng. Vì thế một trường phái khác được ra đời để khắc phục những sai số đó.

**Consequentialism:**

Chủ nghĩa hậu quả cho rằng đạo đức của một hành động phụ thuộc vào kết quả của hành động đó. Những hành động tốt mang lại kết quả tốt và hành động xấu mang lại kết quả xấu.

**04 cách tiếp cận của AI:**

***Thinking humanly***: thường được áp dụng trong lĩnh vực thần kinh. Thành tựu tiêu biểu: mạng nơ-ron Neural Network nhân tạo (thành công nhất trong lĩnh vực AI)

***Thinking rationally:*** dựa theo quy tắc logic mà không nhất thiết phải làm theo hành động của con người.

***Acting humanly***: hành động giống con người mặc kệ thuật toán như thế nào. (ví dụ: Turing Test).

***Acting rationally:*** tìm ra giải pháp (solution) của một vấn đề bằng một chuỗi hành động (actions).

IV. THUẬT TOÁN TÌM KIẾM CƠ BẢN

# 1. Các bước cơ bản của một thuật toán tìm kiếm

* Bước 1 : Tùy theo search strategy(chiến lược tìm kiếm), chọn một node để expand
* Bước 2 : Expanding node vừa chọn

Lặp lại hai bước trên cho đến khi tìm được solution hoặc không còn node để expand . Tất cả các thuật toán đều bao gồm hai bước bước trên, chỉ khác nhau ở cách chọn node ở bước 1

# 2. Tiêu chí đánh giá thuật toán

Trước khi chúng ta đi vào thiết kế các thuật toán tìm kiếm cụ thể, chúng ta cần xem xét các tiêu chí có thể được sử dụng để lựa chọn một thuật toán trong số nhiều thuật toán khác nhau. Chúng ta có thể đánh giá hiệu suất của thuật toán theo bốn tiêu chí:

* Completeness: Thuật toán có đảm bảo tìm ra được solution (giải pháp) nếu có cho vấn đề được đặt ra hay không ?
* Optimality: Solution tìm được có phải là solution tối ưu nhất hay không (optimal solution)
* Time complexity: Thuật toán mất bao lâu để tìm được solution?
* Space complexity: Cần bao nhiêu bộ nhớ trong quá trình tìm kiếm?

Time complexity và space complexity luôn được xem xét đối với một số biện pháp của vấn đề khác nhau. Trong khoa học máy tính lý thuyết, thước đo điển hình là kích thước của đồ thị không gian trạng thái, | V | + | E |, trong đó V là tập hợp các đỉnh (vertices - nodes) của đồ thị và E là tập hợp các cạnh (edges, links). Điều này hoàn toàn hợp lý khi đầu vào là một đồ thị có cấu trúc dữ liệu rõ ràng (Ví dụ như : bản đồ đường đi giữa các tỉnh/thành phố)

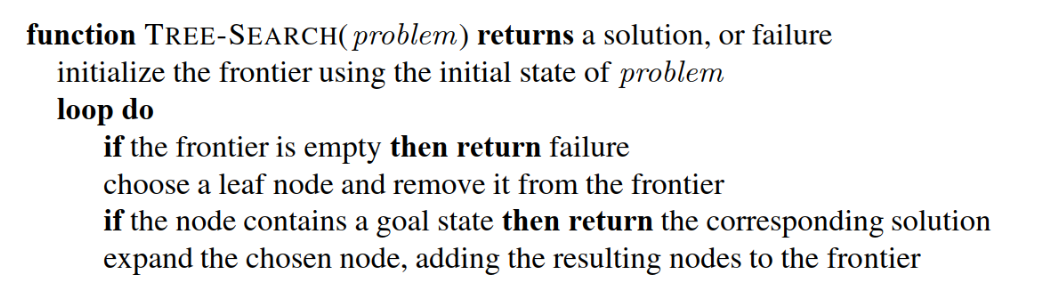
Trong Trí tuệ nhân tạo, đồ thị thường được biểu thị ngầm định bởi trạng thái ban đầu (initial state), hành động (action) và transition model. Vì thế, độ phức tạp(complexity) được biểu thị theo ba đại lượng:

* b, hệ số phân nhánh hoặc số lượng successor (nút con) tối đa của bất kỳ nút nào
* d, độ sâu (số bước dọc theo đường đi từ gốc) của goal state nông nhất
* m, độ dài tối đa của bất kỳ đường đi nào trong state space

Thời gian thường được đo bằng số lượng nút được tạo trong quá trình tìm kiếm và space tính theo số lượng nút tối đa được lưu trữ trong bộ nhớ.

# 3. Conceptual algorithm

## 3.1. Conceptual algorithm #1 Tree search (cây tìm kiếm)

* + - Mã giả thuật toán TREE SEARCH:
    - Một số thuật ngữ cần chú ý:
* Solution: đường đi từ vị trí bắt đầu đến vị trí cần tìm
  + - State: trạng thái của nút đang xét
    - Frontier: là một tập lưu trữ những nút lá
    - Goal\_state: là vị trí chúng ta cần tìm
    - Expand: có thể hiểu là từ một nút ban đầu chúng ta mở rộng nó ra thành nhiều nút khác.
    - Mô tả thuật toán:
* Input: problem (gồm 5 components)
* Output: solution hoặc failure
  + - Các bước thực thi:
* Khởi tạo tập frontier chứa trạng thái ban đầu (initial-state) của problem
  + - Thực hiện vòng lặp và kiểm tra:

+ Nếu tập frontier rỗng đồng thời không tìm được solution thì trả về failure, ngược lại thì ta chọn một node lá trong tập frontier và xóa node này ra khỏi tập frontier.

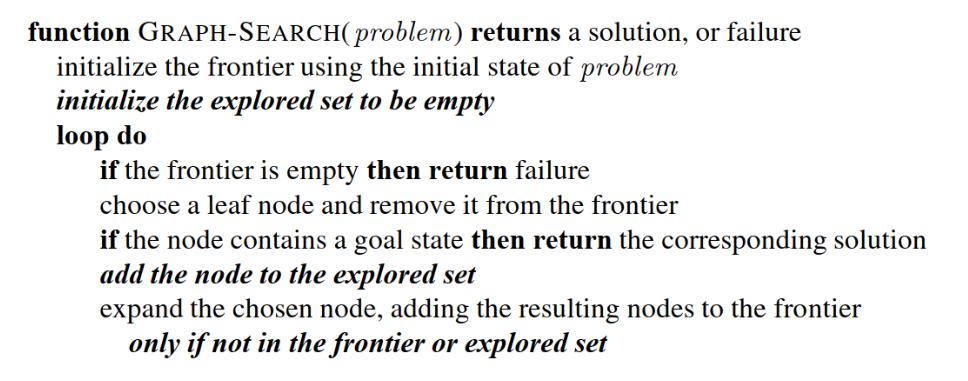
+ Kiểm tra nếu node được chọn là goal\_state thì trả về solution, ngược lại node được chọn không phải là goal\_state thì ta expand node này khi đó ta được các node mới và tiến hành đưa các node mới vào tập frontier

+ Thực hiện vòng lặp đến khi tìm được goal\_state

* + - Chú ý: trong thuật toán TREE SEARCH ở trên sẽ xảy ra một vài vấn đề sẽ có những trường hợp lặp lại những node đã đi qua và sẽ dẫn đến một vòng lặp vô hạn. Để giải quyết vấn đề này ta tiếp tục với thuật toán tiếp theo GRAPH SEARCH.

## 3.2. Conceptual algorithm #2 Graph search (search with memory)

Tương tự như tree seach nhưng thêm một tập explored set chứa các node đã được expanding, tránh được vòng lặp vô tận.

* + - Mã giả thuật toán GRAPH SEARCH:
    - Mô tả thuật toán:
    - Input: problem (gồm 5 components)
    - Output: solution hoặc failure
    - Các bước thực thi:
* Khởi tạo tập frontier chứa trạng thái ban đầu (initial-state) của problem
* Khởi tạo tập explored chứa những node đã được duyệt qua (đã expand)
* Thực hiện vòng lặp và kiểm tra:

+ Nếu tập frontier rỗng đồng thời không tìm được solution thì trả về failure, ngược lại thì ta chọn một node lá trong tập frontier và xóa node này ra khỏi tập frontier.

+ Kiểm tra nếu node được chọn là goal\_state thì trả về solution

+ Thêm node chuẩn bị được expand vào tập explored.

+ Expand node này khi đó ta được các node mới và tiến hành đưa các node mới vào tập frontier nếu như các node mới này không nằm trong tập frontier và tập explored.

+ Thực hiện vòng lặp đến khi tìm được goal\_state.

=> Từ những thuật toán **Conceptual** ta phát triển lên những thuật toán áp dụng vào thực tế: có hai loại **Uninformed search, Informed search** trong nội dung sắp tới chúng ta sẽ tìm hiểu về loại Uninformed search

# 4. Hai loại thuật toán tìm kiếm

* Uninformed search (tìm kiếm mù)
  + Chỉ biết thông tin về vấn đề cần giải quyết thôi.
  + Chọn node một cách ngẫu nhiên để expand.
  + VD: Trong ván cờ, chỉ biết vị trí bắt đầu, không biết đối thủ là ai, điểm mạnh, điểm yếu của đối thủ.
* Informed search (tìm kiếm có thông tin) dựa vào thông tin có được, phỏng đoán f(n) để chọn node có giá trị f(n) nhỏ nhất để expand.
  + Ngoài thông tin về vấn đề cần giải quyết, còn có một số thông tin khác hỗ trợ giải quyết vấn đề.
  + Dựa vào thông tin có được, phỏng đoán f(n) và chọn node có giá trị f(n) nhỏ nhất để expand.
  + VD: Ngoài các vấn đề trên còn có thông tin về tìm kiếm biết con xe đi thế nào, con hậu đi thế nào

# 5. Các cấu trúc dữ liệu

## 5.1. Thành phần của một Node

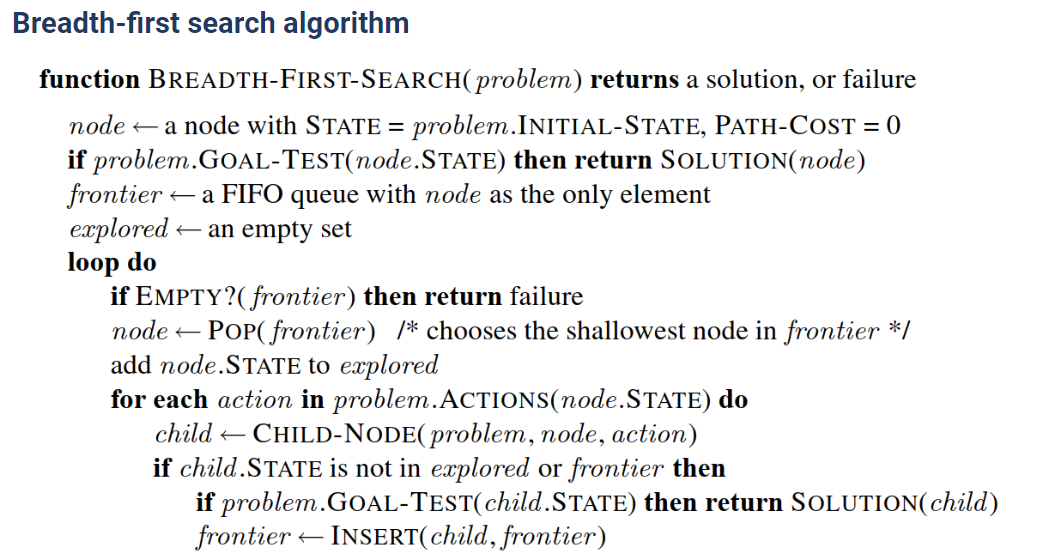
* State: trạng thái tại node đang xét.
* Parent: chứa node cha của node đang xét.
* Action: là action mà node cha thực thi để đến được node con đang xét.
* Path cost: chi phí đi từ vị trí ban đầu đến node đang xét.
* CHILD-NODE: một hàm sinh ra node con từ node cha và action tương ứng.

## 5.2 Queue

* Toán tử của queue
  + Empty ? (queue)
  + Pop (queue)
  + Insert (element, queue)
* Các loại queue
  + FIFO Queue: first in - first out (vào trước ra trước).
  + LIFO Queue (stack): last in - first out (vào sau ra trước).
  + Priority queue: phần tử trong hàng được được lấy ra dựa vào độ ưu tiên tức là độ ưu tiên nào cao hơn sẽ được lấy ra trước.

# 6. Các thuật toán Uninformed search

## 6.1. Thuật toán Breadth-ﬁrst search (BFS)

* Mã giả thuật toán Breadth-first searh (BFS):
* Các bước thực thi:
* Kiểm tra xem State của node đang xét có phải là Goal\_state hay không, nếu là Goal\_state thì trả về solution
* Tạo tập frontier sử dụng FIFO chứa node ban đầu
* Tạo tập explored
* Thực hiện vòng lặp:

+ Nếu tập frontier rỗng mà chưa tìm thấy solution nó sẽ trả về failure, ngược lại lấy 01 node từ tập frontier và thêm node này vào tập explored

+ Thực hiện từng action để expand node đang xét (vòng lặp)

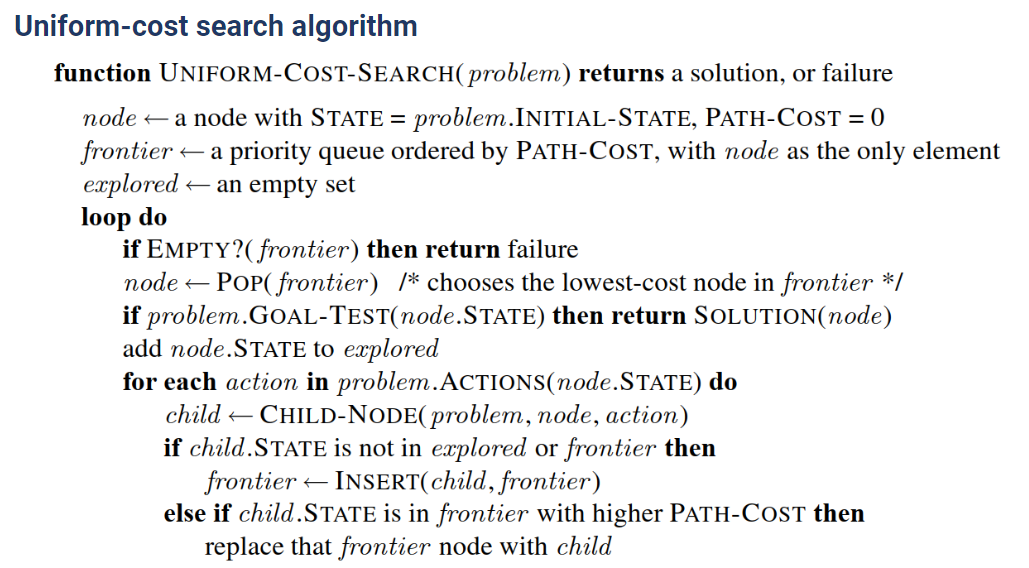
+ Nếu state của node mới được expand mà thuộc tập frontier hay thuộc tập explored thì loại bỏ node này, ngược lại thì kiểm tra các state của các node con nếu là goal\_state thì trả về là solution còn không thì đưa node con này vào tập frontier

+ Tiếp tục thực hiện vòng lặp

* Đánh giá về thuật toán BFS:
* Completeness: thuật toán BFS luôn cho ra được solution
* Optimality: thuật toán BFS optimality khi có ràng buộc điều kiện PAHT-COST phải là hàm tăng dần theo chiều sâu
* Complexity: thuật toán BFS rất tốn kém về thời gian và bộ nhớ

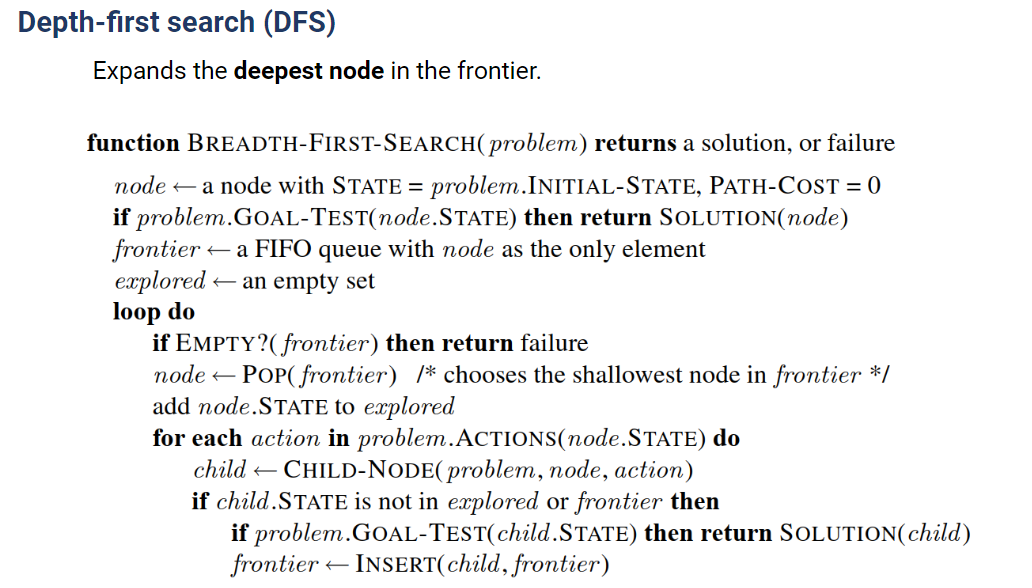
=> Để cải thiện Optimality cho thuật toán BFS thì ta có thuật toán Uniform-cost search

## 6.2. Thuật toán Uniform-cost search

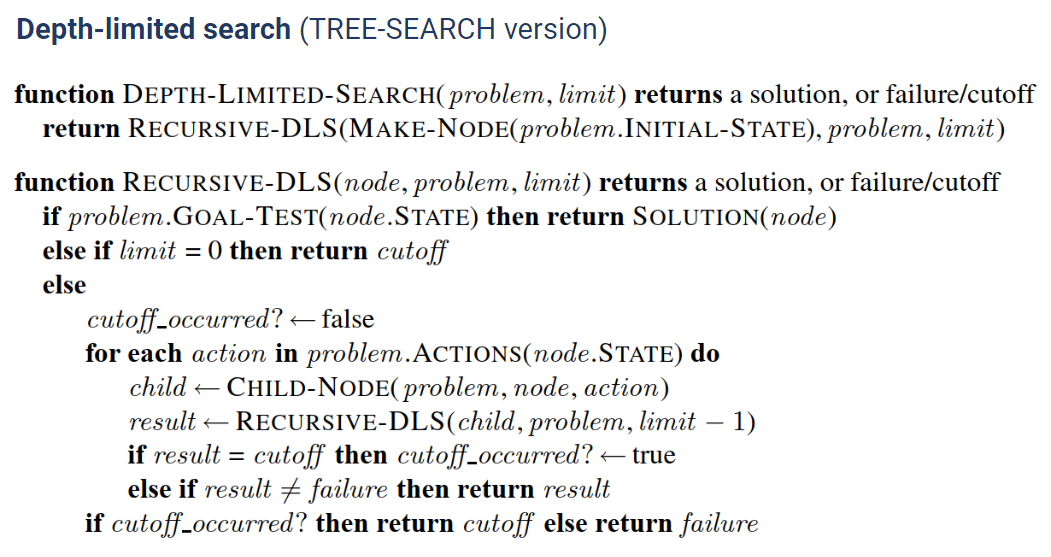
* Mã giả thuật toán Uniform-cost search:
* Về mô tả thuật toán thì tương tự như BFS nhưng thay vì tập frontier sử dụng loại hàng được FIFO thì ta sử dụng Priority queue ta xét theo PATH-COST.
* Trong khi thực hiện vòng lặp ta lấy node từ tập frontier thay vì lấy như BFS thì ta xét xem node nào có PATH-COST ít nhất thì được ưu tiên lấy ra trước
* Sau khi expand các node con từ node đang xét tương ứng với một action ngoài điều kiện là node con này không thuộc tập frontier và tập explored mới được đưa vào tập frontier thì trường hợp node con này đã có trong tập frontier ta sẽ so sánh nếu node con có PAHT-COST ít hơn node con đã tồn tại trong tập frontier thì ta thay thế node con cho node đã tồn tại trong tập frontier

=> Tuy thuật toán Uniform - cost search giải quyết được vấn đề Optimality của BFS nhưng nó vẫn chưa giải quyết được vấn đề về Complexity.

## 6.3. Thuật toán Depth-ﬁrst search (DFS)

* Mã giả thuật toán Depth-first search (DFS):
* DFS giải quyết được vấn đề tồn động lại của BFS và Uniform cost search là Complexity bằng cách đổi loại hàng đợi của tập frontier thay vì dùng FIFO và Priority queue ta sử dụng LIFO.
* Về code và mô tả thuật toán thì DFS gần như tương tự BFS chỉ thay đổi loại hàng đợi của tập frontier. Chính vì sử dụng LIFO nên thuật toán sẽ duyệt theo chiều sâu nó sẽ bắt lấy một nhánh và đào sâu vào đến khi đến đường cùng thì nó loại bỏ nhánh đó hoàn toàn khỏi bộ nhớ và sang nhánh khác đây là mấu chốt giúp cho DFS sử dụng ít bộ nhớ và thời gian hơn BFS.
* Tuy nhiên nó sẽ phát sinh ra vấn đề DFS sẽ cho ra solution không Optimality. Vì phải duyệt theo chiều sâu nên DFS có thể không Completeness khi nó chọn vào nhánh có chiều sâu vô cùng và không tìm thấy goal\_state.

## 6.4. Thuật toán Depth-limited search

* Để giải quyết vấn đề Completeness của DFS ta sử dụng Depth-limited search, đơn giản DLS sẽ ước lượng và giới hạn lại chiều sâu có thể duyệt.
* Mã giả của thuật toán Depth-limited search:
* Mô tả thuật toán:
* Input: problem, limit (chiều sâu giới hạn có thể tìm kiếm)
* Output: solution, failure, cutoff (khi đến chiều sâu quy định nhưng chưa tìm được goal\_state)
* Sử dụng kỹ thuật đệ quy
* Mô tả hàm RECURSIVE\_DLF :

+ Hàm RECURSIVE\_DLF hàm này nhận vào node, problem, limit và trả về solution hoặc failure/cutoff

+ Kiểm tra vị trí đầu tiên có phải goal\_state hay không, nếu là goal\_state trả về solution, ngược lại kiểm tra limit nếu bằng 0 thì trả về cutoff (lúc này đã duyệt đến chiều sâu quy định nhưng chưa tìm được goal\_state)

+ Nếu không vào hai trường hợp trên thì khởi tạo cutoff\_occurred = false

+ Thực hiện vòng lặp lặp qua từng action đế expand ra các node con. Gọi đệ quy hàm RECURSIVE\_DLF truyền vào node con vừa expand, problem và limit - 1

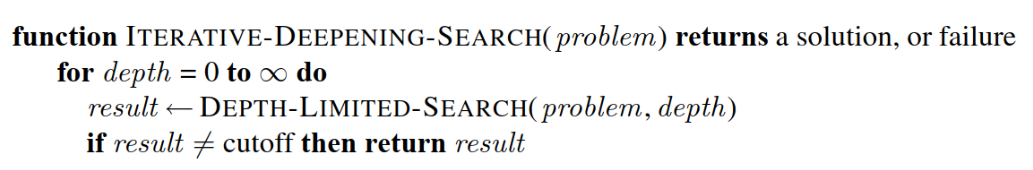
+ Thực hiện đệ quy đến điều kiện dừng

+ Kiểm tra kết quả trả về nếu là cutoff thì gán biến cutoff\_occurred = true, ngược lại nếu là solution thì trả về solution và kết thúc

+ Nếu không là solution thì kiểm tra biến cutoff\_occurred nếu là false thì trả về failure ngược lại trả về cutoff

=> DLS tuy giải quyết được vấn đề khi chọn nhánh vô tận nhưng làm sao để ước lượng được chiều sâu hợp lý ?

## 6.5. Thuật toán Iterative deepening search

* Đây là thuật toán tối ưu nhất trong các thuật toán tìm kiếm mù. Nó giải quyết được vấn đề Complexity của BFS, vấn đề Completeness của DFS và vấn đề ước lượng chiều sâu của DLS.
* IDS nó sẽ gọi DFS với độ sâu tăng dần
* Mã giả của thuật toán Iterative deepening search:

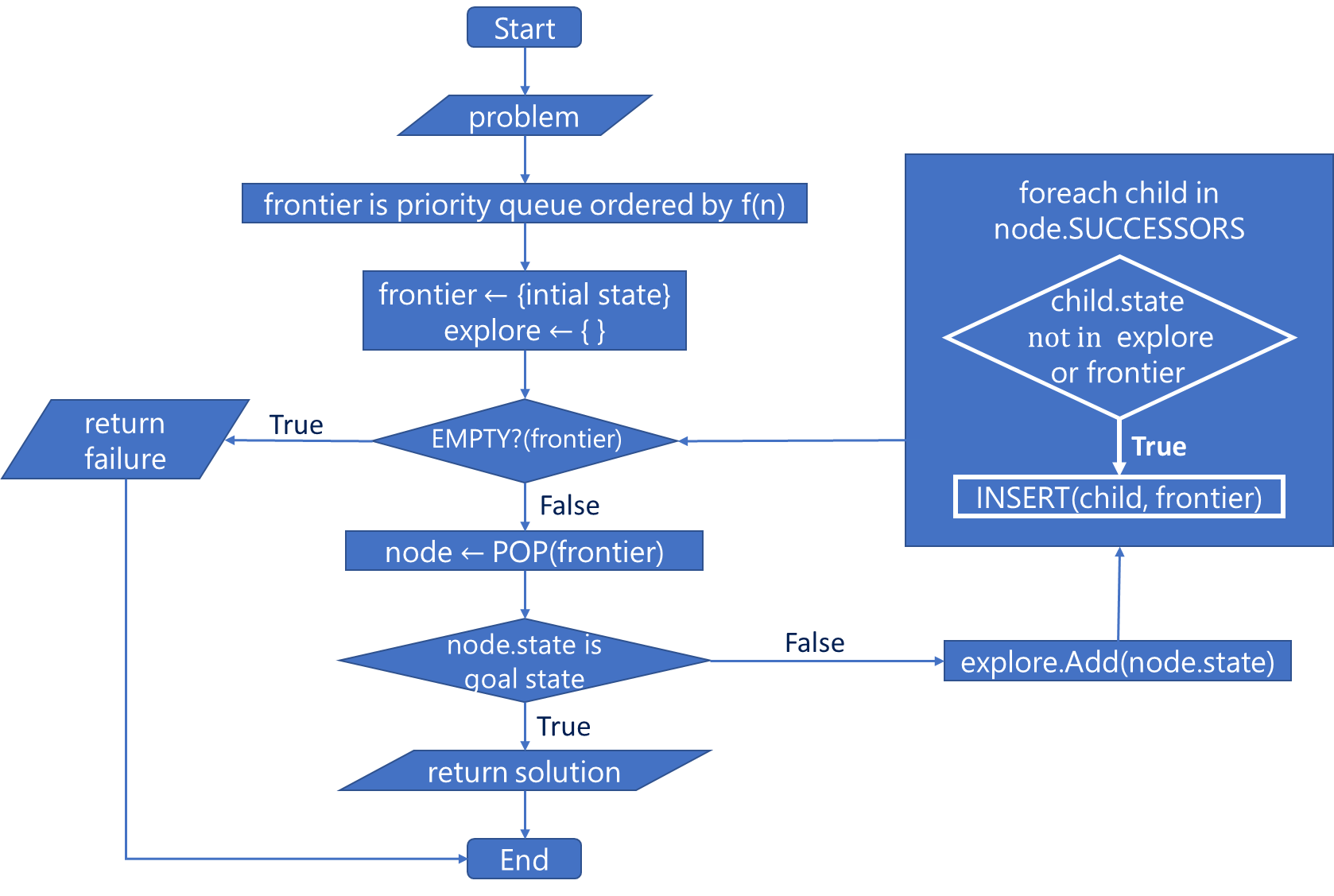
# 7. Các thuật toán Informed search

## 7.1. Các hàm ước lượng quảng đường

***Evaluation function f(n)*** là hàm sử dụng thông tin có được về môi trường để ước lượng chi phí đường đi từ initial state đến goal state. Giá trị này được dùng để thay đổi chiến lược tìm kiếm hiệu quả hơn.

***Heuristic functions h(n)*** là hàm ước lượng quảng đường đi từ node n đến goal state. Ví dụ : straight line distance heuristic

## 7.2. Ý tưởng chung của Informed search



Hầu hết các thuật toán Informed search đề dựa trên ý tưởng này, chỉ khác nhau ở cách tính f(n).

## 7.3. Thuật toán Best-first search

* Heuristic function h(n) là ước lượng từ node n đến goal\_state
* Đặc điểm của Best-first search là **f(n) = h(n)**
* Best-first search còn tồn động vấn đề là không tối ưu (not optimal) vì hai lý do:
* Ước lượng chi phí của nó không đầy đủ thiếu chính xác **f(n) = h(n)** là ước lượng chi phí của Best-first search tuy nhiên để đầy đủ hơn ***f(n) = chi phí từ vị trí ban đầu đến node hiện tại*** (PATH-COST) + h(n).
* Ước lượng không chính xác h(n) khác biệt rất lớn so với thực tế dẫn đến việc không tối ưu.

## 7.4. Thuật toán A\* search

* Để giải quyết các vấn đề của Best-first search A\* search có một vài cải tiến như sau:
* f(n) = PATH-COST + h(n) việc bổ sung thêm PATH-COST giúp cho ước lượng của A\* search trở nên đáng tin cậy và đầy đủ.
* A\* search ràng buộc thêm điều kiện cho h(n) (consistent heuristic satisfies) đá sad h(n) <= cost(n, n’) + h(n’) bất đẳng thức này đơn giản chỉ là bất đẳng thức tam giác.

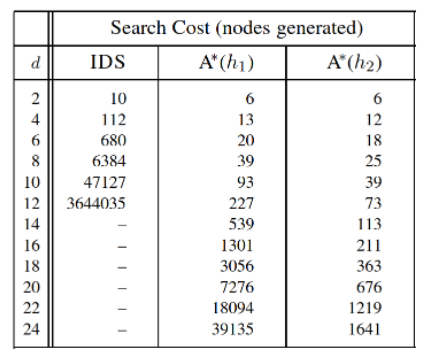
=> Những cải tiến giúp cho A\* search complete và optimal\* (h(n) consistent)

* Chú ý:
* Nếu như h(n) <= cost(n, n’) + h(n’) thì h(n) <= actual cost điều này có nghĩa là nếu như h(n) là consistent thì ước lượng h(n) sẽ luôn nhỏ hơn hoặc bằng đường đi thực tế.
* Straight-line heuristic là consistent
* Do các thuật toán informed search được cải tiến từ Uniform cost search mà Uniform cost search là biến thể của BFS nên A\* search vẫn còn vấn đề về bộ nhớ vẫn chưa được tối ưu.

## 7.5. Cách tạo heuristic function

* Cách 1 : Giảm nhẹ vấn đề bằng cách bỏ bớt các hạn chế (restriction) cho mỗi action.
* Được định nghĩa là problem có ít hạn chế ít quy định hơn các problem gốc
* Cost của relaxed problem luôn luôn nhỏ hơn hoặc bằng cost của problem gốc
* Khi chúng ta có nhiều heuristic tại một thời điểm để tìm heuristic tốt nhất thì ta đưa tất cả vào một tập hợp và tìm max của tập heuristic khi đó ta sẽ tìm được heuristic lớn nhất dẫn đến kết quả sẽ tối ưu nhất.
* Cách 2 : Dùng cơ sở dữ liệu mẫu (pattern databases).
* Chọn một subproblem (subproblem không phải là relaxed problem vì subproblem chỉ thu hẹp lại không gian tìm kiếm chứ không làm ít đi những quy định hạn chế của problem gốc)
* Xây dựng pattern databases và sử dụng nó để tính heuristic cho full problem
* Để xây dựng pattern databases ta sẽ tìm kiếm ngược từ goal\_state của subproblem tại đây thực 1 action thì ta được 1 state mới và cost để biến đổi state này về goal\_state tiếp tục thực hiện action thì ta lại có thêm state mới và từ state mới này ta cũng có được cost để về goal\_state và cứ tiếp tục như vậy đến khi nào không có state mới được tạo ra hay hết action thì ngưng => pattern databases
* Chúng ta có thể xây dựng nhiều pattern databases từ những subproblem và tìm max của những heuristic này để tối ưu được tìm kiếm.

## 7.6. So sanh hàm Heuristic.

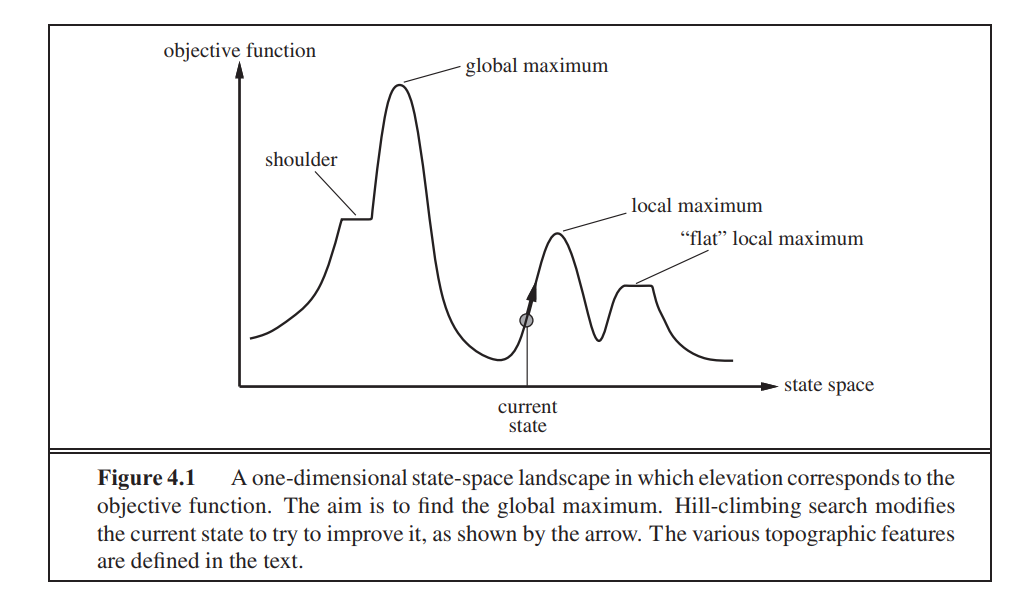
* Với những h(n) consistent thì h(n) nào càng lớn thì càng tốt
* Ví dụ: ta dùng bài toán 8-Puzzle
* Khi sử dụng A\* search ta xét hai h(n) khác nhau:
* h1(n) = tổng số những ô đặt sai vị trí = 8 => h1(n) consistent
* h2(n) = tổng khoảng cách từ các ô sai vị trí đến vị trí đúng = 18 => h2(n) consistent do đây là tính khoảng cách
* Khi đó ta được kết quả như sau:
* Bảng kết quả cho thấy được thời gian chạy của 3 thuật toán dựa vào số node mà từng thuật toán expand qua mỗi độ sâu.
* Từ bảng kết quả rõ ràng ta thấy A\* search vượt trội hơn hẳn IDS (thuật toán đại diện tối ưu nhất của nhóm thuật toán tìm kiếm mù). Ngoài ra ta còn thấy được A\*(h2) tối ưu hơn A\*(h1) đều này chứng minh cho ta thấy được với các h(n) consistent thì h(n) nào lớn hơn thì tối ưu hơn.

V. LOCAL SEARCH – SEARCHING FOR GOAL STATE

# 1. So sánh với các thuật toán tìm kiếm trước đó:

* Các thuật toán trước đó cần phải biết các thông tin:
  + Obversable: biết rõ được đang ở state nào của môi trường.
  + Discrete: những action ứng với mỗi state của môi trường, là rời rạc
  + Deterministic: mỗi action có 1 hành động duy nhất, nghĩa là mình có thể xác định được khi mình thực hiện action thì mình thu được kết quả gì.
* Các thuật toán phía sau sẽ loại bỏ những ràng buộc trên. Khi đó đòi hỏi phải thay đổi mục tiêu tìm kiếm, cách lưu

# 2. State space landscape:

Local search sẽ tập trung vào 2 thứ: current state và mục đích nó hướng đến (chi phí thấp nhất/phần thưởng nhiều nhất/…). Biểu diễn trực quan dưới dạng:

≫ Objective function = reward function = – cost function

≫ Current state: từ trạng thái hiện tại, ta tìm kiếm các trạng thái kế cận nó.

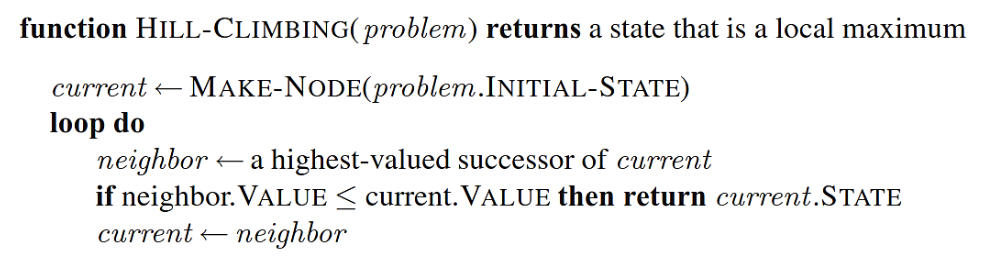
≫ Khi đạt global optimum, ta nói thuật toán đã optimal.

≫ Optimum = mininum hoặc maximum.

# 3. Local search:

* Tập trung tìm kiếm các state ở trạng thái hiện tại hoặc gần đó
* Khác với các thuật toán đã học trước đó thay vì tìm ra các action để đi đến goal\_state thì thuật toán này sẽ tìm ra goal\_state.
* Ý tưởng:
* Khởi tạo một state -> tính ra giá trị objective function (hàm mục tiêu dựa vào giá trị của hàm mục tiêu để chọn ra các state phù hợp)
* Sao đó thực hiện action tại state này -> tạo ra các state mới và tính objective function của các state mới và so sánh state nào tạo ra được giá trị objective function lớn hơn
* Cập nhật state mới có objective function lớn hơn, tiếp tục thực hiện các action để tìm kiếm các state mới tính giá trị objective function tại state đó và tiếp tục cập nhật state có objective function lớn hơn.
* Ưu điểm: tại một thời điểm chỉ cần lưu lại current node và các neighbors (có thể không lưu lại ) vì vậy nó tiết kiệm không gian bộ nhớ rất nhiều và có thể tìm được trong các không gian infinite mà các thuật toán trước không làm được.
* Nhược điểm: không optimal vì có thể rơi vào điểm local maximum / local minimum

# 4. Hill-climbing search:

* Ý tưởng: tại current state thực hiện một action tạo ra các state mới (neighbor) từ các state mới tính objective function chọn ra state có objective function lớn nhất. Thực hiện lại quy trình tương tự với state vừa cập nhật đến khi không tìm được objective function lớn hơn thì dừng lại.
* Mô tả thuật toán:
* Input: một problem và initial\_state (có thể tự khởi tạo)
* Output: trả về state giúp cho objective function đạt được local maximum
* Các bước thực thi:

+ Khởi tạo node current dựa vào initial\_state

+ Thực hiện vòng lặp

+ Gán vào neighbor những successor có value lớn nhất (objective function lớn nhất) trong tất cả các successor của current state

+ Kiểm tra nếu value của neighbor nhỏ hơn bằng value của current thì trả về current.state (lúc này đã đạt được value lớn nhất).

+ Ngược lại gán neighbor cho current (chưa đạt được value lớn nhất), thực hiện lại vòng lặp tiếp tục tìm các neighbor tốt nhất và so sánh các value cứ thực hiện như vậy đến khi tìm được local maximum thì kết thúc vòng lặp.

* Đánh giá:
* Completeness: vì nó có thể mắc kẹt lại những state local maximum
* Optimal: không, vì chưa đảm bảo trả về những state global maximum
* Complex: tốn ít bộ nhớ vì chỉ lưu những giá trị của trạng thái hiện tại

# 5. Issues of Hill-climbing search:

## 5.1. Quá nhiều successors:

* Sẽ dẫn đến việc tốn rất nhiều thời gian để tìm max value của những successor để xác định neighbor có giá trị VALUE cao nhất.
* Giải quyết: dùng Stochastic hill climbing để sinh ngẫu nhiên các successor cho đến khi tìm được successor có value lớn nhất
* Điều kiện dừng:
* Điều kiện dừng có thể dẫn đến thuật toán dừng tại vị trí Local maxima và Plateaux mà không phải là Global maxima dẫn đến không optimal.
* Giải quyết:

+ Plateaux: cho phép nó đi ngang bằng cách bỏ điều kiện bằng tại câu lệnh điều kiện if. Tuy nhiên nên giới hạn số bước đi ngang để tránh trường hợp đi ngang đi lại vô tận.

+ Local maxima:

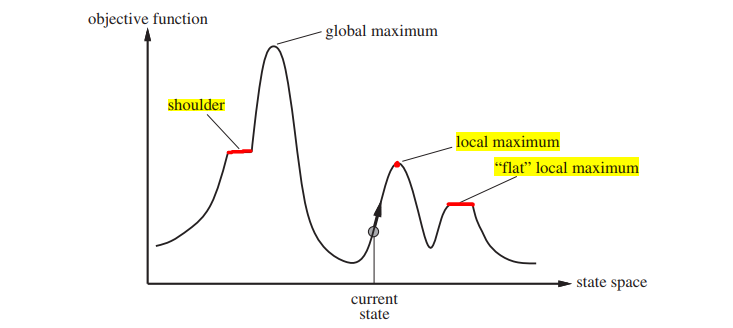
+ Random - restart hill climbing: chạy lại thuật toán nhiều lần với giá trị inital\_state được chọn random.

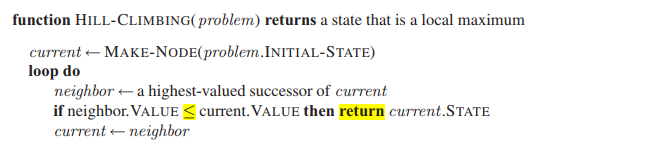
+ Local beam search: cải tiến của Random - restart hill climbing (thực hiện lại nhiều lần) đồng thời thực tìm kiếm theo k hướng khác nhau. Ngoài ra còn liên kết các luồng tìm kiếm lại với nhau giúp cho việc tìm kiếm nhanh hơn.

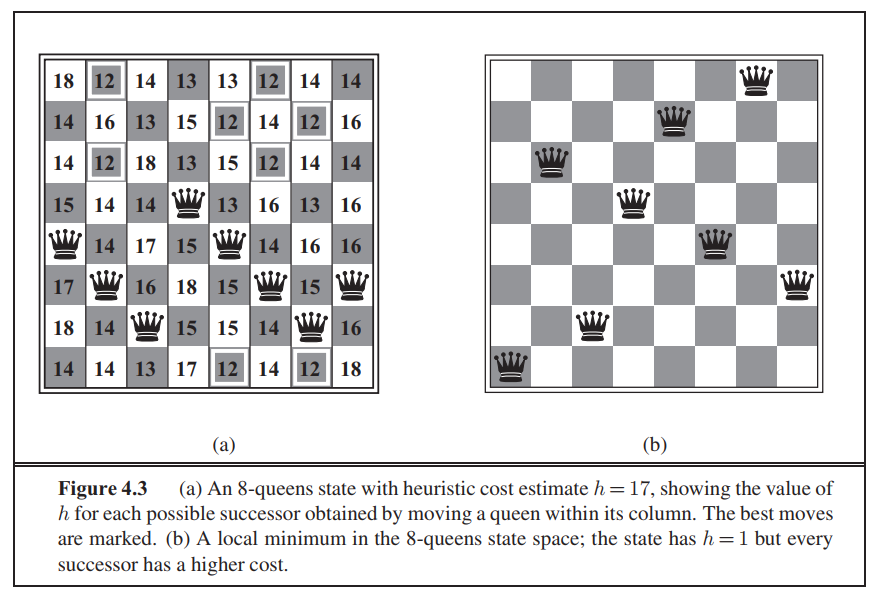
+ Stochastic beam search: Việc sử dụng phương pháp Local beam search sẽ dẫn đến việc có quá nhiều successor để giải quyết vấn đề này ta sử dụng lại phương pháp Stochastic hill climbing. Việc chọn ngẫu nhiên các successor dựa vào xác suất, xác suất này tỷ lệ thuận với value để đảm bảo tính đa dạng của việc tìm kiếm.

## 5.2. Local optimum:

Khi current state đạt được local optimum hoặc flateaux (“flat” local optimum hoặc shoulder) (*như hình*). Khi đó, do thuật toán Hill-climbing chỉ lấy điều kiện so sánh *neighbor.VALUE ≤ current.VALUE*, cho nên khi điều kiện trên thỏa mãn thì return và kết thúc quá trình tìm kiếm. Do đó, thuật toán này không optimal.

****

****

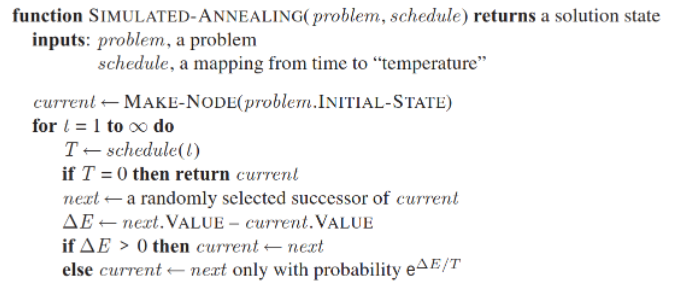
Ví dụ: (Hình b) Không thể di chuyển bất kỳ quân hậu nào, để giảm số lượng quân hậu có thể tấn công nhau.

**🡪** Giải quyết:

* Để vượt qua các flateaux: cho phép di chuyển ngang **với số bước giới hạn** (*để tránh bị lặp vô tận*), khi gặp neighbor có giá trị VALUE không tăng so với giá trị của current. => Tăng tỉ lệ tìm được lời giải cho bài toán 8-queens từ 14% lên 94%.
* Để giải quyết local optimum:
  + Dùng Random-restart hill-climbing. Nghĩa là trong trường hợp bị rơi vào local optimum, ta sẽ chạy lại thuật toán hill-climbing với giá trị INITIAL\_STATE là một state được chọn random (khác với Initial State ở trường hợp lúc đầu).
  + Dùng Local beam search (*beam: chùm ánh sáng*): (cải tiến của random restart hill-climbing).
  + ***Instead of trying k random searches sequentially, do them*** ***simultaneously.*** Nghĩa là đồng thời thực hiện việc tìm kiếm theo k hướng khác nhau (đa nhiệm), thay vì lượt lượt thử từng cách một, nếu chưa thấy thì restart.
  + ***Local beam search: not just simultaneously, choose best of all searches.*** Không chỉ chọn best neighbor cho mỗi search, mà nó chọn best of all searches (*lấy cái tốt nhất của các best successors thành phần* 🡪 tốn thời gian để tìm max).
  + Stochastic beam search: combines Local beam search and Stochastic hill-climbing. Giải quyết vấn đề này sinh của Local beam search là phải tìm best successor. Khi có quá nhiều successor sẽ dần đến tốn nhiều chi phí cho việc tính Value cho mỗi successor và tìm max.
    - Không tìm max
    - Chọn successor ngẫu nhiên với xác suất dựa theo VALUE

# 6. Simulated Annealing:

## 6.1. Thuật toán Simulated Annealing.

* Mô tả thuật toán
* Input: problem, shedule
* Output: solution state
* Các bước thực hiện:

+ Khởi tạo node current dựa vào initial\_state

+ Thực hiện vòng lặp:

+ Tính giá trị nhiệt độ T

+ Kiểm tra nếu T = 0 trả về trạng thái hiện tại của current

+ Ngược lại, chọn giá trị next một cách ngẫu nhiên, next là một successor của current

+ Tính độ thay đổi delta E

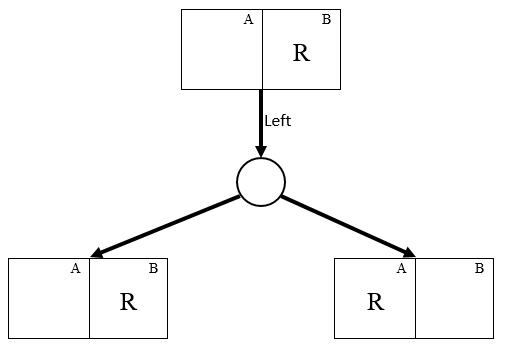
+ Kiểm tra nếu delta E > 0 (next có value lớn hơn current) nên gán next cho current

+ Ngược lại, chỉ chọn next với xác suất e^(delta E / T) gán cho current

# 7. Searching in NONDETERMINISTIC environments

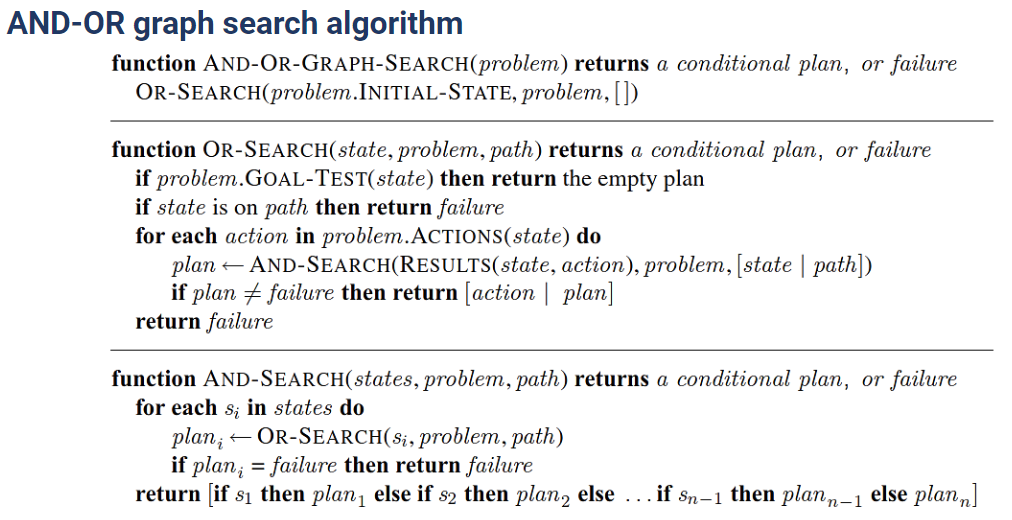
## 7.1. Tìm kiếm trong môi trường không xác định

* Một action có thể trả về nhiều trạng thái
  + Results (State, action) 🡪 States{s1, s2, s3,…}
  + Ví dụ: Robot thực hiện di chuyển sang trái (Vị trí hiện tại là B di chuyển sang A)
    - Results (B, Left) 🡪 States {s1 = B, s2 = A}



## 7.2. And\_Or Graph Search

***7.2.1. Thuật toán***

******

* + Một số thuật ngữ cần chú ý :
    - state: trạng thái hiện tại (thực hiện or search).
    - problem: vấn đề cần giải quyết.
    - path: mảng chứa các state đã xuất hiện.
    - plan: chứa kết quả trả về (bao gôm action và trạng thái đạt được khi thực hiện action đó).
    - states: chứa nhiều state (thực hiện and search).
  + Mô tả thuật toán :
    - Bước 1 : Điều kiện thoát đệ quy (thoát nếu là goal state)
    - Bước 2 : Điều kiện thoát đệ quy (thoát nếu đã có trong path)
    - Bước 3 : Chạy toàn bộ các action có thể thực hiện ở state hiện tại
    - Bước 4 : Lấy kết quả khi thực hiện action ⇒ Đưa state hiện tại vào path ⇒ Gọi And search với các states được tạo ra.
    - Bước 5 : Kiểm tra kết quả trả về của hàm And search, nếu kết quả trả về khác failure thì thêm action vào plan còn không thì return failure.
    - Bước 6 : Lấy ra từng state trong states để thực hiện :
    - Bước 7 : Gọi hàm Or search với state được lấy ra và lưu kết quả trả về vào plani
    - Bước 8 : Kiểm tra plani = failure thì return failure (nhánh hiện tại không đi đến kết quả mong muốn)
    - Bước 9 : Trả về tất cả kết quả của từng state

***7.2.2. Đánh giá thuật toán***

|  |
| --- |
| **And-Or Search** |
| Đầu vào: Problem ( vấn đề cần giải quyết ) |
| Đầu ra: Conditional plan hoặc có thể là Failure nếu không tìm được giải pháp. |
| Thời gian chạy: tốn ít thơi gian, vì tận dụng được ưu điểm của thuật toán DFS |
| Chi phí bộ nhớ: Nhiều (do phải lưu trữ các trường hợp có thể của AND-OR SEARCH TREE để kiểm tra) |
| Completeness: Có |
| Optimality: Thuật toán không optimals vì trong thuật toán sử dụng phương pháp tìm kiếm làm DFS (tìm kiếm theo chiều sâu) |

***7.2.3. Ứng dụng thực tế:*** Giải các bài toán thực tế mà không biết rõ tất cả môi trường xung quanh hay không biết kết quả thực sự trả về khi thực hiện hạnh động.

# 8. Searching in Partially Observable Enviroments

* Không biết hoặc quan sát đươc một phần của môi trường
* Ưu điểm:
  + Tiết kiệm được chi phí mua và lắp đặt sensor
  + Tiết kiệm thời gian để sensor lấy thông tin
* Nhược điểm
  + Xử lý không được nhiều bài toán
  + Xử lý vấn đề không hiệu quả
* Belief state: tập hợp các state mà agent có thể đang ở trong thực tế

## 8.1. No observable

***Ý tưởng giải:***

* + Xây dựng belief state
    - initial state: tất cả trạng thái trong thực tế có thể có
    - possible actions: ACTIONS(belief state b) 🡪 b = {s1, s2}

actionsp(s1) = {a, b, c}

actionsp(s2) = {a, c, d}

có thể chọn actions(b) = {a, b, c, d}

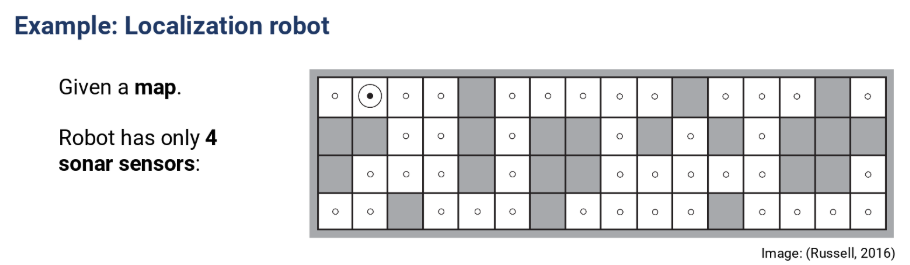
hoặc = {a, b}

* + - Transition model: RESULT (belief state b, action a) = new belief state b’
      * b = {s1, s2}
      * b’ = Resultp (s1, a) ∪ Resultp (s2, a)
    - Goal test: toàn bộ trạng thái trong belief state để là goal state
      * b = {s1, s2}
      * nếu goal-testp(s1) và goal-testp(s2) đều trả về là đúng thì goal-test(b) là đúng
    - cost: step-cost(b) and path-cost(b)
  + Chuyển bài toán thành fully observable
  + Xử dụng các thuật toán BFS, DFS, A\*, AND-OR, … để giải

## 8.2. Partially observable

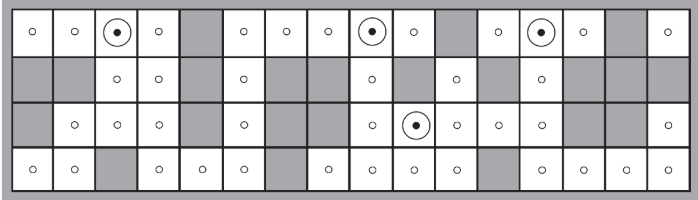
Tương tự như no observable nhưng transition model sẽ được thu hẹp dần nhờ đưa trạng thái belief vào percept kiểm tra nếu thõa điều kiện thì stateb đó được gửi lại trong beilef nếu không thõa thì loại bỏ nó khỏi bilief state.

## 8.3. Demo



Belief state ban đầu là tất cả các vị trí

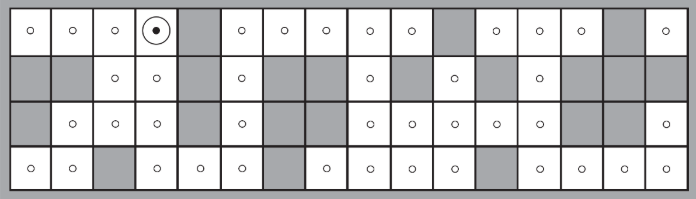
* Step 1: move right



Percept có 1 tường chặn ở trên

🡪Belief state còn lại 4 state

* Step 2: move right



Percept có tường chặn ở trên và bên phải

🡪Belief state còn lại một state

# 9. Online Search

## 9.1. So sánh về offline search và online search :

* Khái niệm offline search và online search
* Offline search
  + Offline search là các thuật toán chạy trên máy tính trước tiên, để tìm ra solution và đưa cho agent chạy trong thực tế để tìm ra giải pháp. Mỗi action của agent đều có thể biết được kết quả dựa trên transition model.
  + Ví dụ : DFS, BFS, A\*,…….
* Online search
* Là các thuật toán vừa chạy trong môi trường thực tế vừa học để tìm ra solution, với các bài toán không có transition model. Nếu càng thực hiện thì online search càng thông minh.
* Ví dụ : Dự báo thời tiết ,…
* Bảng so sánh : (online search sẽ useful trong trường hợp : there is a penalty for, nondeterministic enviroments, no enviroment model)

|  |  |  |
| --- | --- | --- |
|  | Offline Search | Online Search |
| Đầu vào | -Node gồm 5 thành phần:   * Initial state * Possible actions: ACTIONS * Transition model: RESULT * Goal test: GOAL-TEST * Cost: STEP-COST, PATH-COST   - Đưa vào 1 lần ngay khi chương trình bắt đầu | -Node gồm 4 thành phần:   * Initial state * Possible actions: ACTIONS * Goal test: GOAL-TEST * Cost: STEP-COST, PATH-COST   -Không cần transaction model  - Đưa vào liên tục, sau các hành động thì lại có các input tiếp theo. |
| Đầu ra | * Solution | * Một action |
| Cách thức hoạt động | Giải quyết vấn đề trong môi trường ảo dựa vào Transition model để tìm ra solution trên máy tính, đưa ra solution và thực hiện. | Đặt agent vào môi trường để agent exploring và tìm solution. |
| Thuật toán ứng dụng | A\*, DFS, Simulated Annealing, BFS, … | Online depth-first search, Learning Real-Time A\* |
| Ứng dụng | Các bài toán như tìm đường đi trong mê cung, n-queens problems | Ứng dụng trong các bài toán tìm đường đi cần phải tìm action tiếp theo, ví dụ như giải mê cung,.. |

## 9.2. Competitive ratio :

Trong online search có 2 loại của path costs đó là :

* + Actual path cost : cost thực tế mà agent thực hiện, bao gồm các bước explore môi trường và tìm kiếm solution
  + Shortest path cost : cost ngắn nhất để tìm được solution

Competitive ratio = actual path cost / shortest path cost (độ hiệu quả của thuật toán)

## 9.3. Online Depth-first search agent :

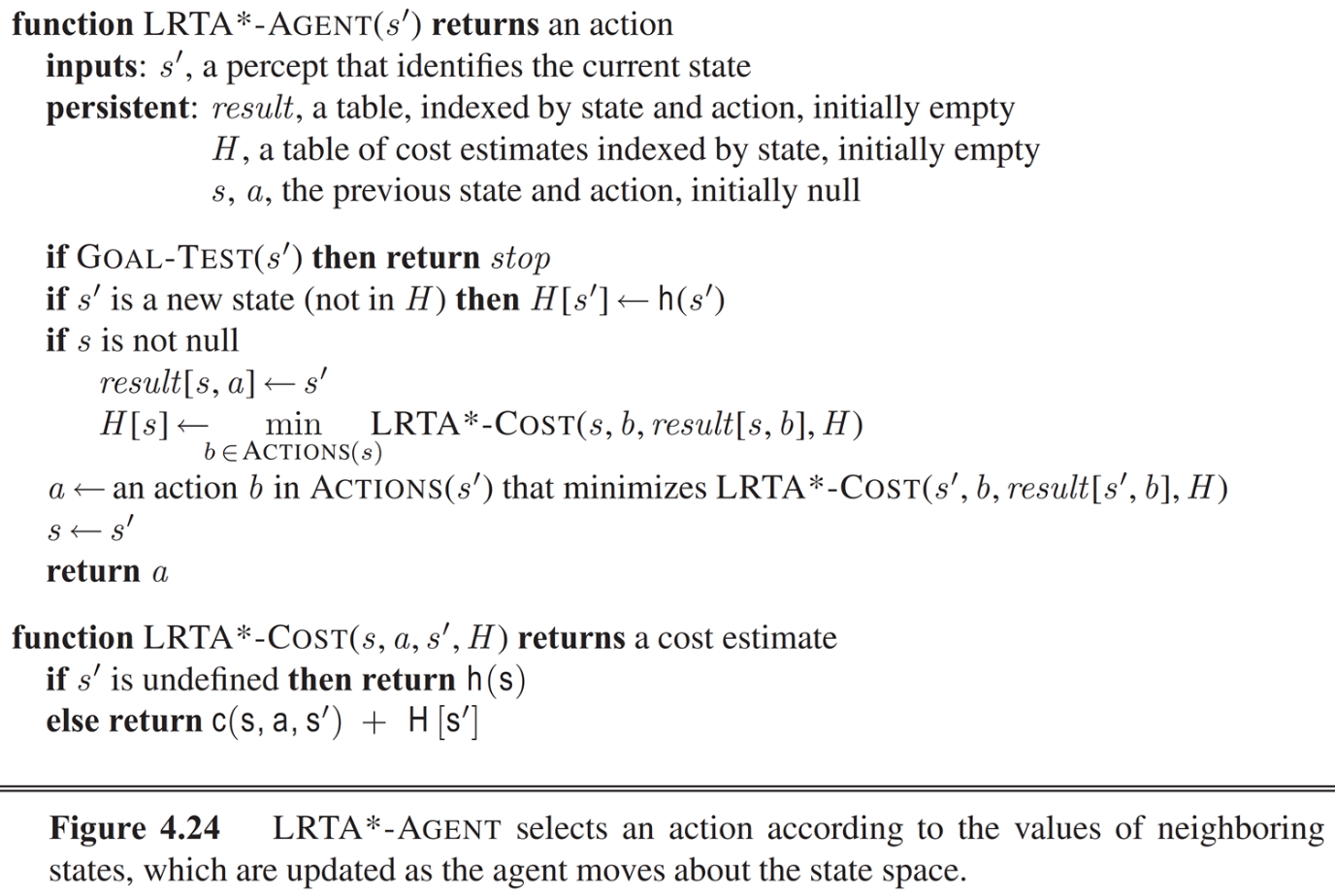
* Mô tả thuật toán:
* **Bước 1** : Đầu vào là (s’) state môi trường hiện tại ⇒ với mỗi state thì trả ra một acction tương ứng
* **Bước 2 :**  inputs (s’ : state hiện tại mà agent nghĩ vị trí của state đang ở đó)
* **Bước 3 :**  persistent
* Result : trả về một bảng chứa (s, a , s’) , có nghĩa là trả về vị trí agent đang đứng ở state (s) và thực hiện action a thì được state (s’)
* Untried : một mảng chứa các state và action mà chưa chạy thử qua.
* Unbacktracked : một mảng chứa các state của các trạng thái trước đó mà nó chưa xét tới.
* **Bước 4 :** Kiểm tra xem (s’) có phải là trạng thái đích chưa, nếu đúng thì dừng lại.
* **Bước 5 :** Kiểm tra xem (s’) có phải là một sate mới không (không nằm trong untried) thì ta đưa các Action(s’) vào untried[s’].
* **Bước 6 : kiểm tra xem** s có ở trạng thái null không, nếu null thì nhảy qua Bước 7 còn không thì thực hiện :
* Đưa  s’ vào bảng result [s, a]
* Thêm (s) vào unbacktracked[s’]
* **Bước 7 :** Kiểm tra xem nếu untried[s’] là rỗng thì ta qua Bước 8 còn không thì tiếp tục thực hiện :
* Nếu unbacktracked[s’] là rỗng thì dừng lại.
* Ngược lại, unbacktracked[s’] không là tập rỗng, thì ta thực hiện action b để quay lại trạng thái trước đó để thực hiện (backtrack)
* **Bước 8 :** Nếu ở Bước 7, action của (s’) không là tập rỗng thì ta lấy ra 1 action từ untried[s’]
* **Bước 9 :** s’ lúc này sẽ là s.
* **Bước 10 :** return action a.
* Completeness : thuật toán sẽ tìm được solution nếu bài toán đó có solution
* Thời gian thực hiện của online search sẽ lâu hơn offline search.

## 9.4. Online A\* search : ( tìm được solution tốt một cách nhanh chóng )

* Ý tưởng : ở từng bước, chúng ta chọn một successor’s theo estimated
* ( ban đầu khỏi tại sẽ là khoảng cách chim hueristic, sau đó agent sẽ vừa đi trong thực tế và vừa cập nhật lại H )

## 9.5. Learning Real – Time A\* agent :

* Đầu vào : (s’) percept cho biết cái state hiện tại của enviroment.
* Đầu ra : với mỗi state mà agent nghĩ mình đang ở đó, thì nó nên có action gì, hay nói cách khác là trả về một action đối với 1 state.
* Mô tả thuật toán:



* Bước 1 : đưa đầu vào là (s’) state hiện tại và trả về một action.
* Bước 2 : inputs ( s’, a)
* Bước 3 : persistent
  + Result : trả về một bảng chứa (s, a , s’) , có nghĩa là trả về vị trí agent đang đứng ở state (s) và thực hiện action a thì được state (s’)
  + H : được khởi tạo ban đầu là rỗng, là một bảng ước lượng quảng đường đi từ trạng thái s tới đích.
* Bước 4 : Nếu (s’) là trạng thái đích thì dừng chương trình lại.
* Bước 5 : Nếu (s’) là một state mới, không có trong H, thì ta khởi tạo H(s’) ← h(s’)
* Bước 6 : Kiểm tra s nếu không null thì thực hiện :
  + Đưa [s, a] s’ vào bảng result
  + Cập nhật H[s] bằng giá trị min ở hàm LRTA\*-COST, trạng thái s có nhiều action, mỗi action có 1 cost khác nhau ⇒ chọn ra action có cost thấp nhất.
* Bước 7 : Đầu vào của LRTA\*-COST là, một state s nào đó, thực hiện action a, thu được trạng thái s’ chưa biết.
* Bước 8 : Kiểm tra xem s’ có trong H chưa :
  + Nếu s’ chưa có trong H thì khởi tạo h(s)
  + Ngược lại nếu nó nằm trong H rồi thì ta cập nhật c(s, a, s’) + H[s’]
* Bước 9 : Chọn action nào đó giúp cho bước đi tiếp theo có cost thấp nhất
* Bước 10 : s’ trở thành s
* Bước 11 : Trả về action a
* Completeness : thuật toán sẽ tìm được solution nếu bài toán đó có solution.
* Thời gian thực hiện được cải thiện vì nó sử dụng estimated cost.

# 10. Constraint Satisfactoin Problems

## 10.1. Constraint satisfaction problems (CSP)

* Trong CSP có 3 components :
  + X ( tập hợp tất cả các biến )
  + D ( tập hợp các giá trị mà các biến được phép nhận )
  + C ( tập hợp các ràng buộc mà các biến phải thỏa mãn )
* Solution của CSP : gán giá trị cho tất cả các biến với giá trị thuộc domains của nó sao cho thỏa mãn tất cả các ràng buộc.

## 10.2. Constraint graph :

* Nodes : là tất cả các biến.
* Links : là 2 biến trong 1 ràng buộc bất kỳ

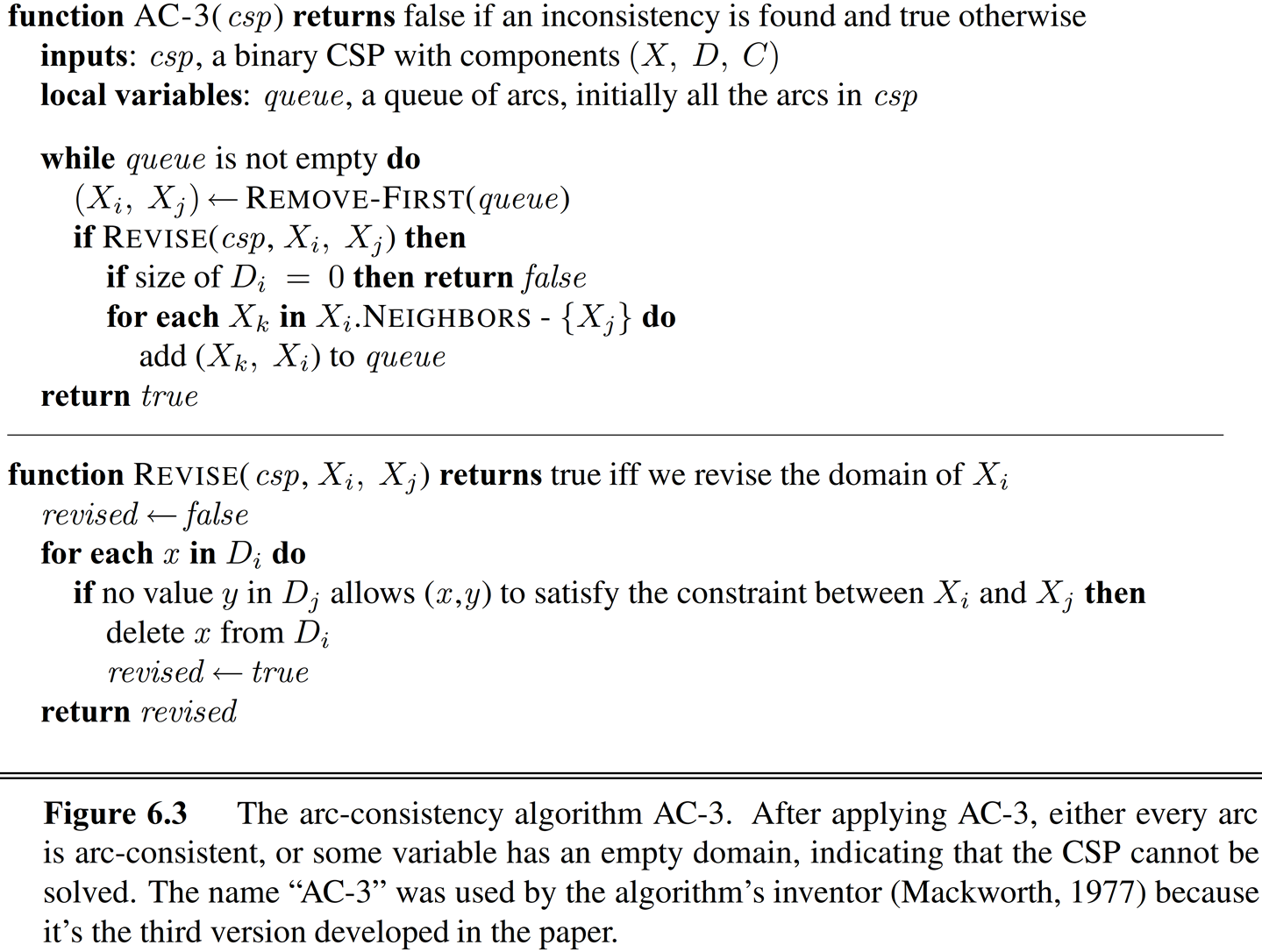
## 10.3. Constraint propagation :

* Ý tưởng : thu hẹp domains của các biến xung quanh nhờ sử dụng các biến có domains là 1 giá trị. (csp giải quyết các bài toán nhanh hơn các thuật toán search thông thường như (BFS, DFS, A\*,…..), trong nhiều trường hợp không gian tìm kiếm quá rộng thì các thuật toán search không tìm ra được nhưng CSP vẫn tìm ra được. )
* Cụ thể : sử dụng những ràng buộc để thu hẹp domain cho các biến, khi một biến nào đó đã được thu hẹp domain thì có thể nó sẽ ảnh hưởng đến domain của biến thứ 2, rồi thứ 3 (sự lan truyền). Đến khi thu hẹp domain cho mỗi biến chỉ còn 1 giá trị thì chúng ta đã tìm được solution.

## 10.4. Local consistency (node , arc , path )

* Consistent : các biến thỏa mãn các ràng buộc thì ta gọi là consisten
* Local consistency : 1 nhóm các biến nằm gần nhau thỏa mãn ràng buộc thì ta gọi là local consistency
* Cách hoạt động của local consistency : làm cho các biến nằm gần nhau thỏa mãn hết các ràng buộc, sau đó lan truyền sự thỏa mãn điều kiện đó sang các biến xung quanh, để đảm bảo các biến xung quanh thỏa mãn được các ràng buộc. (loại bỏ những giá trị nằm trong domain làm cho biến đó không consisten)

## 10.5. AC – 3 algorithm (arc consistency)

* Đầu vào : một binary csp của một problem với các components (X, D, C)
* Đầu ra : False nếu không thỏa mãn ràng buộc, True nếu thỏa mãn ràng buộc
* Mô tả thuật toán:
* Bước 1 : đầu vào là csp gồm có 3 thành phần (X, D, C)
* Bước 2 : inputs (csp, một binary csp với các components (X, D, C))
* Bước 3 : queue (tập hợp của những constraint có 2 biến)
* Bước 4 : Chạy lần lượt qua tất cả các constraint đó
  + Đưa 1 constraint 2 biến (Xi, Xj) vào hàm Revise
  + Nếu hàm Revise trả về true thì thực hiện :
  + Kiểm tra xem nếu tập domain của Di = 0 thì ta return false ( không có solution)
  + Nếu tập domain của Di # 0 ⇒ Chọn ra biến Xk lân cận với biến Xi. Thêm (Xk , Xi) vào queue
* Bước 5 : Hàm Revise nhận đầu vào là (csp, Xi, Xj) và kiểm tra nếu có thu hẹp domain của Xi thì trả về true, false nếu không thu hẹp được domain của Xi
* Bước 6 : Gán giá trị cho biến revised là false
* Bước 7 : Lấy ra từng giá trị x trong tập domain Di của Xi .Xét nếu không tồn tại giá trị y trong tập domain Dj của Xj , giúp cho (x, y) thỏa mãn ràng buộc thì ta tiếp tục thực hiện :
  + Loại bỏ x khỏi tập domain Di
  + Gán cho revised là true
* Bước 8 : Trả về revised
* Bước 9 : Nếu đã thực hiện xét xong các constraint trong queue thì trả về true.

## 10.6. Path consistency :

* Sử dụng cho các constraint 3 biến.
* Chúng ta có một bộ 2 biến { (Xi, Xj) } được xem là path – consistency , khi ta có { Xi = a, Xj = b} sau đó ta tìm được giá trị cho Xm để thoản mãn constraints với {Xi, Xm} và {Xm, Xj}. Còn nếu không tìm được thì tồn tại cặp giá trị (a, b) làm cho nó inconsistent.

## 10.7. K-consistency : (dạng tổng quát của local consistency)

Ý tưởng : lấy ra một tập có K -1 biến , sau đó tìm giá trị cho biến thứ K, nếu tìm được thì nó đạt K-consistency, nếu tìm không được thì nó không đạt K-consistency.

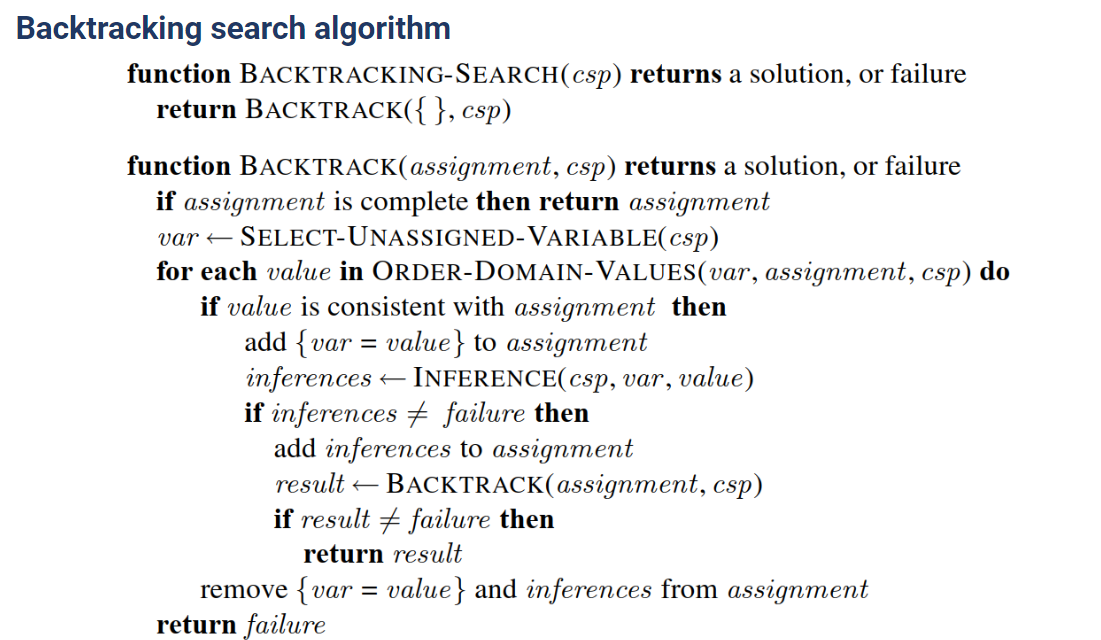
## 10.8. Global constraints :

* Định nghĩa : là một constraint có số lượng biến tùy ý
* Ví dụ : Alldiff constraint, Atmost constraint
  + Alldiff( x1, x2, x3, x4)
  + Với bài toán sodoku 9x9 thì :
  + Alldiff(0,1, 2, 3, 4, 5, 6, 7, 8)

## 10.9. Solving a Sodoku puzzle : (AC – 3, Triplets )

* Triplets :
  + B1 : trong 1 unit, chúng ta tìm ra 3 ô mà domain của 3 ô này là tập con của 1 tập X có 3 giá trị.
  + B2 : loại bỏ 3 số trong tập X ra khỏi domains của các ô còn lại trong unit đó

## 10.10. Backtracking search algorithm

* Ý tưởng : nếu có những biến có domain lớn hơn 1 giá trị, thì ta chọn một giá trị để thử, đưa vào sau đó gán giá trị cho các ô khác, nếu chúng ta gán được cho tất cả các ô thì đã tìm được solution , nhưng nếu chúng ta gán sau 1 số bước thì gặp một ô có domain là rỗng thì không tìm được solution, chứng tỏ giá trị chúng ta chọn ban đầu đã sai, chúng ta loại bỏ giá trị đó khỏi domain của ô đó.
* Các bước chính:
  + B1 : chọn các biến chưa được gán giá trị, thử gán một giá trị cho nó
  + B2 : kiểm tra giá trị đó nếu nó inconsistent với assignment hiện tại thì ta thử một giá trị khác, nếu nó consistent với assignment hiện tại thì ta tiếp tục quay lại B1.
* Bản chất : DFS. Cải tiến hơn DFS , do backtracking sử dụng thuộc tính giao hoán
* Đầu vào : csp
* Đầu ra : solution hoặc failure
* Mô tả thuật toán
* Đầu vào là một csp gồm có các components (X, D, C) và trả về solution (nếu tất cả các biến đã được gán giá trị) hoặc trả về failure
* return BackTrack ({}, csp)
* Hàm BackTrack nhận vào assignment, csp ( đối với bài toán sodoku thì assignment ban đầu là các ô đã có giá trị), (nếu chưa có gì thì đưa tập rỗng vào assignment)
* Nếu assignment đã hoàn thành, có nghĩa là tất cả các biến đều được gán bằng một giá trị duy nhất, thì assignment đó cũng chính là solution, ta return assignment.
* Nếu assignment chưa hoàn thành thì ta chọn ra một biến chưa được gán giá trị.
* Gán cho biến vừa chọn ra một giá trị từ tập domain của nó. Thực hiện vòng lặp:

**+** Nếu giá trị vừa gán cho biến đó consistent với assignment hiện tại thì ta qua

**+** Đưa giá trị đó cho biến và thêm nó vào assignment.

**+** Kiểm tra xem tập domain của các biến xung quanh có giảm hay không.

**+** Nếu inferences không gặp biến nào có tập domain là rỗng thì qua

**+** Thêm inferences vào assignment nếu nó có tập domain chỉ có 1 giá trị

**+** Gọi đệ quy, lấy biến khác và tiếp tục gán giá trị cho nó.

**+** Nếu result khác failure thì return result

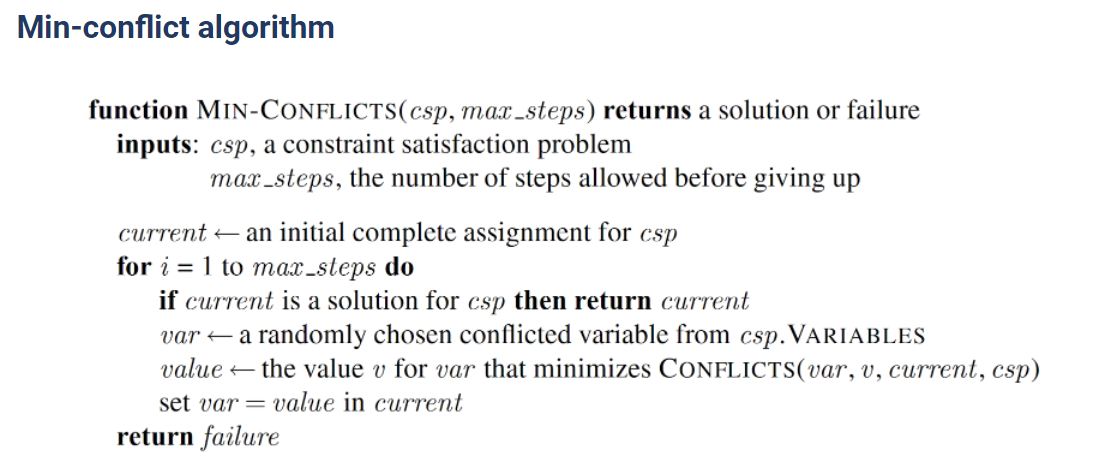
**+** Nếu inferences gặp trạng thái failure thì ta remove {var = value} và inferences khỏi assignment.

* Return false và qua nhánh khác thực hiện.

## 10.11. Comments on backtracking search algorithm :

* Select next variable and value
  + Chọn biến theo minimum remaining-values heuristic : chọn ra biến có domain nhỏ nhất.
  + Chọn biến theo Degree heuristic : chọn ra biến có số lượng degree lớn nhất ( degree : số lượng constraint mà các biến còn liên quan tới ).
  + Chọn value theo Least-constraning-value heuristic : chọn ra biến để lại cho chúng ta nhiều lựa chọn nhất cho các biến khác.
* Inference
  + Forward checking : kiểm tra domain của các biến lân cận nó
  + Maintaining arc consistency : kiểm tra domain của tất cả các biến
* Backtracking strategies
  + Backtrack theo Backjumping : tạo ta tập conflict-set , chứa các assignments có khả năng xung đột với biến hiện tại mình đang xét.
  + Khi gặp failuer thì ngay lập tức nhảy về tập conflict-set.

## 10.12. Min-conflict algorithm.

* Ý tưởng : ban đầu sẽ khởi tạo giá trị cho tất cả các biến, sau đó chọn ra một biến và thay đổi giá trị biến này sao cho số lượng contrainst bị vi phạm của biến nhỏ nhất có thể.
* Đầu vào : csp, max\_step (có chức năng giới hạn số lần duyệt)
* Đầu ra : solution hoặc failure
* Mô tả thuật toán:
* Gán một giá trị nào đó cho biến hiện tại (current)
* Lặp lại max\_step lần:

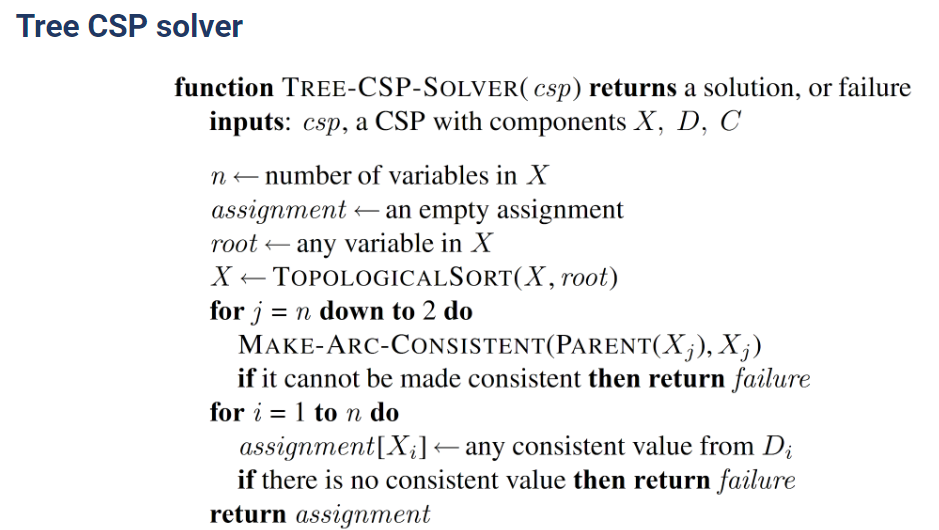
+ Kiểm tra current có phải là giải pháp mình cần tìm thì mình sẽ dừng thuật toán và trả về solution.

+ Ngược lại, chúng ta chọn một biến nào đó giá trị còn conflict (vi phạm constraint).

+ Lấy một giá trị nhỏ nhất lấy từ hàm CONFLICTS gán cho biến vừa chọn.

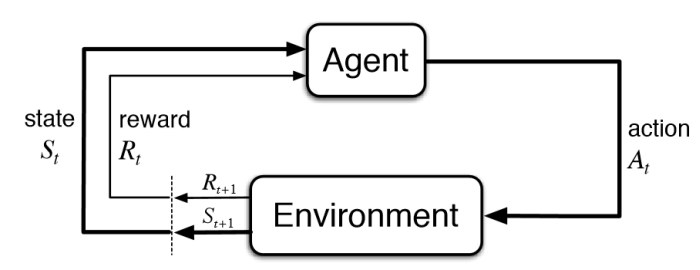
* Nếu như đi hết số lần lặp tối đa (max\_step) mà vẫn chưa tìm ra solution thì trả về failure.

## 10.13. Tree CSP solver.

* Đặc điểm: thuật toán này chỉ chạy được trên tree, nếu không phải là tree thì loại bỏ các node tạo thành vòng trong graph. Thời gian giải bài toán chỉ tính trên thời gian tuyến tính và không cần dùng backtrack.
* Các bước chính:
  + Bước 01: chọn một variable (node) làm root. Sau đó sắp xếp các biến trên tree thành có thứ tự (biến đổi graph thì mảng một chiều).
  + Chạy thuật toán nào đó trên constraint hai biến (có thể là AC3) duyệt qua tất cả các biến để thu hẹp domain
  + Chọn các giá trị trong domain gán cho biến.
* Đầu vào : csp
* Đầu ra : solution hoặc failure (bài toán không có lời giải)
* Mô tả thuật toán:
  + Gọi biến n là số lượng variables X.
  + Khởi tạo biến assignment (dùng để lưu những biến đã được gán giá trị, ban đầu là rỗng).
  + Chọn một biến bất kì là root.
  + Sau khi có root thì thực hiện TopologicalSort, sau bước này thì tập X sẽ là tập có thứ tự và được xem như mảng một chiều.
  + Thực hiện vòng lặp chạy ngược (từ cuối mảng) để lấy ra cặp biến (nút đang xét và nút cha của nó).
    - Chạy thuật toán bất kỳ (AC3) để thu hẹp domain.
    - Nếu như có một domain nào đó bằng rỗng thì dừng thuật toán và trả về failure.
  + Nếu vòng lặp trên được duyệt qua, tất cả domain được thu hẹp nhưng không có có domain nào bị rỗng thì sẽ thực hiện vòng lặp tiếp theo để gán giá trị. Đi theo chiều từ đầu mảng.
    - Lấy lần lượt từng biến ra để gán giá trị consistent
    - Nếu không có giá trị consistent thì dừng thuật toán và trả về failure.
  + Sau khi vòng lặp trên kết thúc thì sẽ tìm được solution.

# 11. Reinforcement Learning (RL).

## 11.1. Khái niệm.

Reinforcement Learning là một phần của Machine Learning. Ở đây, các agents được tự học về cơ chế khen thưởng và trừ điểm. Các agent sẽ thực hiện các hành động hoặc đi con đường tốt nhất có thể để đạt được số điểm cộng tối đa và điểm trừ tối thiểu thông qua các quan sát trong một tình huống cụ thể.

Nguồn ảnh : Introduction to Reinforcement Learning for Beginners, https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners

## 11.2. Các thuật ngữ sử dụng trong Reinforcement Learning.

Agent – Là người ra quyết định và người học duy nhất

Environment – một thế giới vật lý nơi agent học hỏi và quyết định các hành động được thực hiện.

Action – Danh sách các hành động mà agent có thể thực hiện

State – trạng thái hiện tại của agent trong môi trường.

Reward – phần thưởng cho agent

Policy – agent chuẩn bị các chiên lược (quyết định) để lập ra bản đồ các tình huống cho actions.

Value Function – Hàm giá trị cho thấy phần thưởng đạt được tính đến thời điểm các policy được thực hiện.

## 11.3. Ưu - nhược điểm của Reinforcement Learning.

* Ưu điểm:
  + Có thể giải quyết các vấn đề phức tạp và bậc cao.
  + Là mô hình học hỏi liên tục nên một sai lầm được thực hiện trước đó sẽ khó có thể xảy ra trong tương lai.
  + Phần tốt nhất là ngay cả khi không có dữ liệu đào tạo, nó sẽ học được thông qua kinh nghiệm mà nó có được từ việc xử lý dữ liệu đào tạo.
* Nhược điểm:
  + Chúng ta sẽ lãng phí sức mạnh xử lý và không gian không cần thiết bằng cách sử dụng nó cho các vấn đề đơn giản hơn.
  + Trong thực tế, chi phí bảo trì rất cao.

## 11.4. Các ứng dụng của Reinforcement Learning.

* ***Trong kinh doanh, tiếp thị và quảng cáo***: phân tích sở thích của khách hàng và giúp quảng cáo sản phẩm tốt hơn.
* ***Trong trò chơi***: ứng dụng các thuật toán như AlphaGo, Alpha Zero để tính toán chiến thuật chơi các trò chơi như cờ vua, shogi và cờ vây. Ngoài ra RL còn cung cấp môi trường chơi game tốt hơn bằng cách sửa đổi trình giả lập.
* ***Trong hệ thống gợi ý, đề xuất***: chúng ta dễ thấy ứng dụng này trong các nền tảng như Youtube, Netflix. Thuật toán này dựa vào sở thích của khách hàng để hiển thị các chương trình xu hướng mới nhất.

## 11.5. Thư viên OpenAI Gym

OpenAI Gym là một bộ công cụ cung cấp nhiều loại môi trường mô phỏng (trò chơi Atari, trò chơi trên bàn, mô phỏng vật lý 2D và 3D, v.v.), có thể đào tạo các agents, so sánh chúng hoặc phát triển các thuật toán Máy học mới (Reinforcement Learning).

## 11.6. Cross-entropy method

* Ý tưởng:

+ Bước 1: cho chạy N episodes (được hiểu như là các lần done)

+ Bước 2: tính toán total reward cho từng episode và quyết định ngưỡng reward. Thường thì chúng ta sẽ dùng percentile cho các rewards (50th và 70th)

+ Bước 3: bỏ các episode có total reward nhỏ hơn ngưỡng.

+ Bước 4: Huấn luyện các “elite” episodes bằng cách sử dụng các quan sát làm input các actions làm đầu ra mong muốn.

+ Bước 5: lặp lại bước 1 cho đến khi có được kết quả như mong đợi.

VI. GAME “MARIO SOKOBAN”

# 1. Lý do chọn Mario sokoban:

Mario Sokoban lấy ý tưởng từ tựa game kinh điển có tên “sokoban”. Nhiệm vụ của sokoban là đưa những chiếc thùng gỗ vào đúng vị trí của nó. Người chơi sẽ chiến thắng cho chơi nếu tất cả chiếc thùng gỗ được đưa vào đúng vị trí và ngược lại, sẽ thua trò chơi nếu không còn khả năng di chuyển nữa.

Nguồn ảnh : Game Sokoban, https://www.numuki.com/game/sokoban/

Cái tên “Mario” dựa trên một nhân vật điện tử nổi tiếng. Từ đó nhóm chúng em xin mượn những hình ảnh quen thuộc, đầy tuổi thơ từ tựa game này để tạo nên một project vừa mang tính học thuật, vừa gợi lại những ký ức không thể nào quên của một thời thơ ấu vô tư, hồn nhiên.

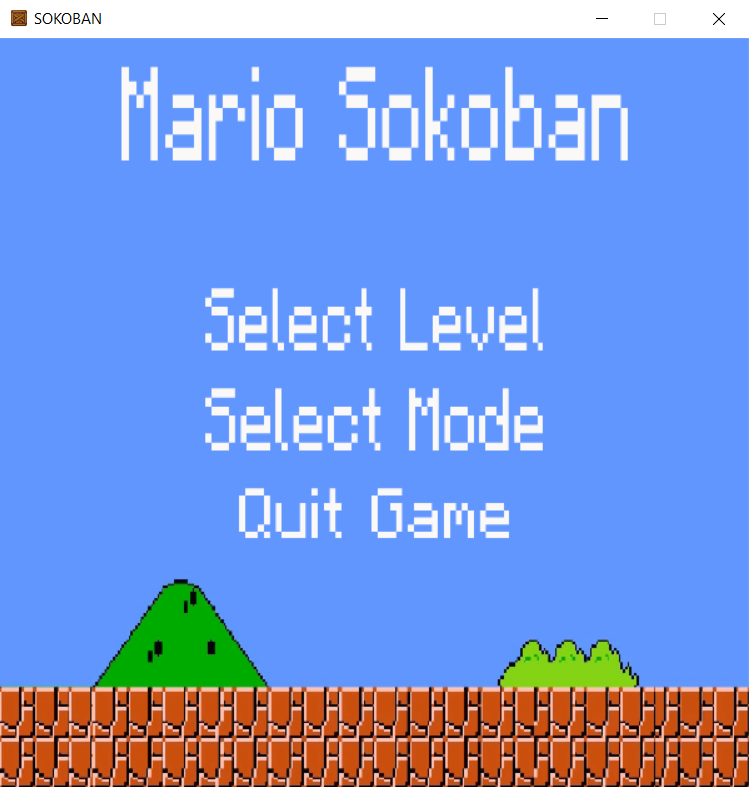
Xuyên suốt mario sokoban. Nhân vật sẽ sử dụng những thuật toán tìm kiếm như Astar và BFS để tìm được kết quả cuối cùng. Từ đó có thể so sánh tốc độ cũng như độ tối ưu của mỗi thuật toán đem lại.

Project sử dụng ngôn ngữ python và thư viện pygame để cài đặt phần giao diện cũng như thuật toán để chạy ra kết quả cuối cùng.

# 2. Giao diện, hướng dẫn sử dụng project

Đầu tiên, chúng em xin giới thiệu những hình ảnh đến từ project mà nhóm em đã thiết kế. Qua đó chúng em sẽ nói về các chức năng chính của project nhóm mình.

* Màn hình menu:



Đây là màn hình khi bắt đầu game, bao gồm 3 mục chính đó là:

+ Select level: Chọn các màn để bắt đầu chơi

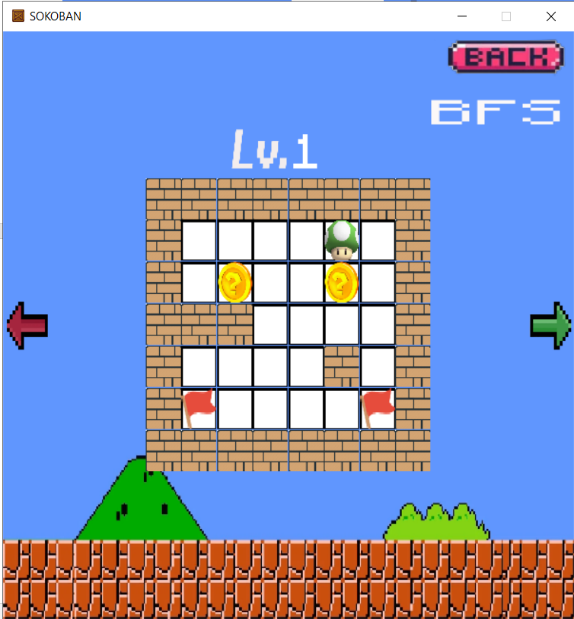
+ Select Mode: Có 2 mode chính là Astar và bfs

+ Quit Game: Thoát trò chơi

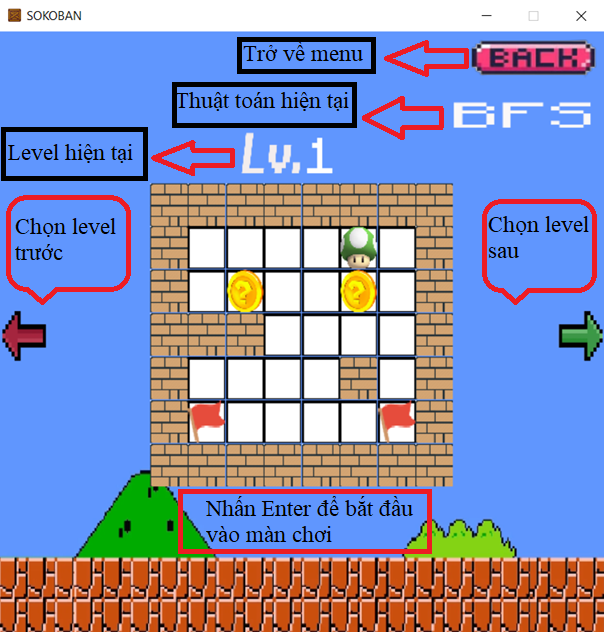
****

**Select Mode**

Click chuột trái vào select mode để mở mode menu

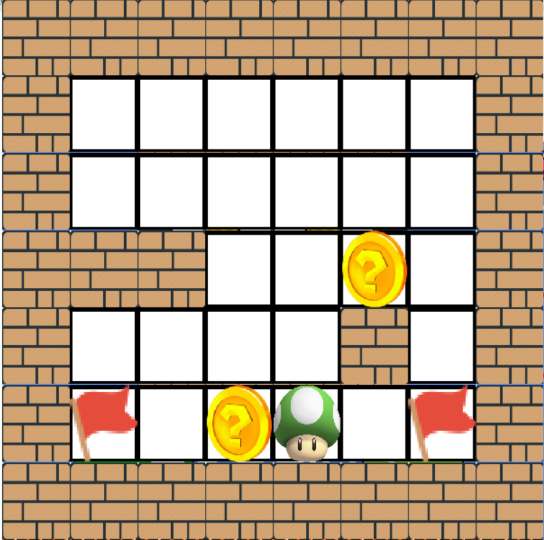
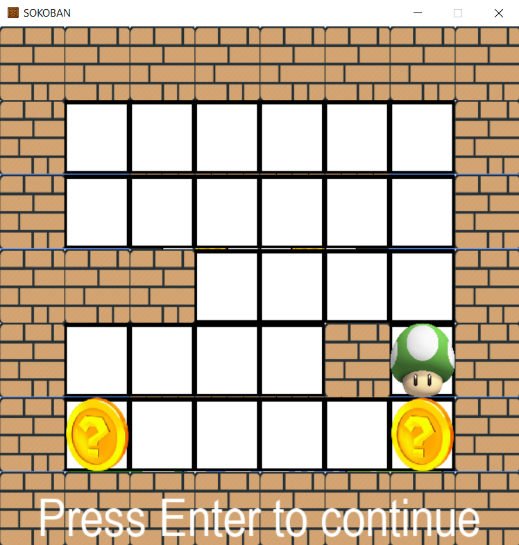


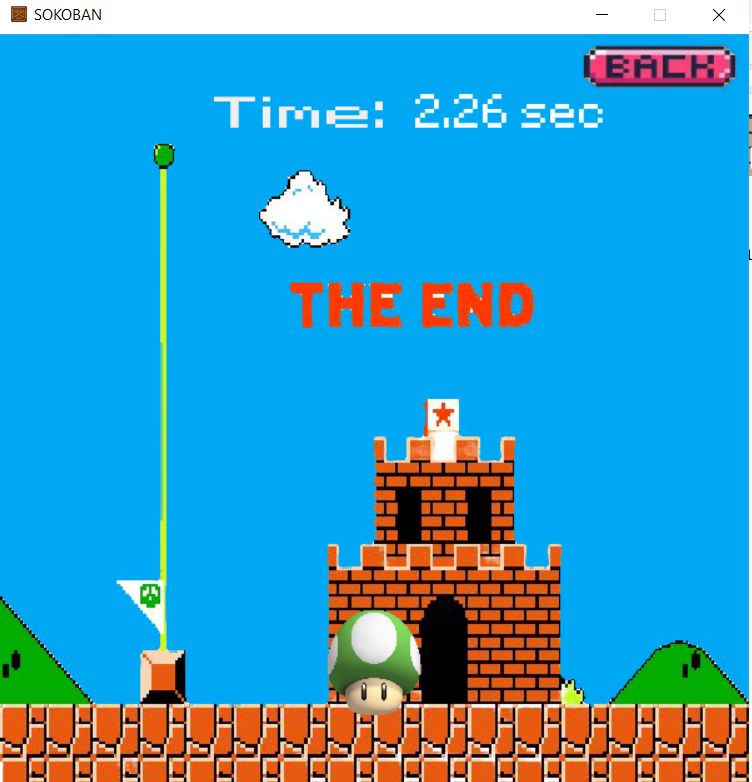
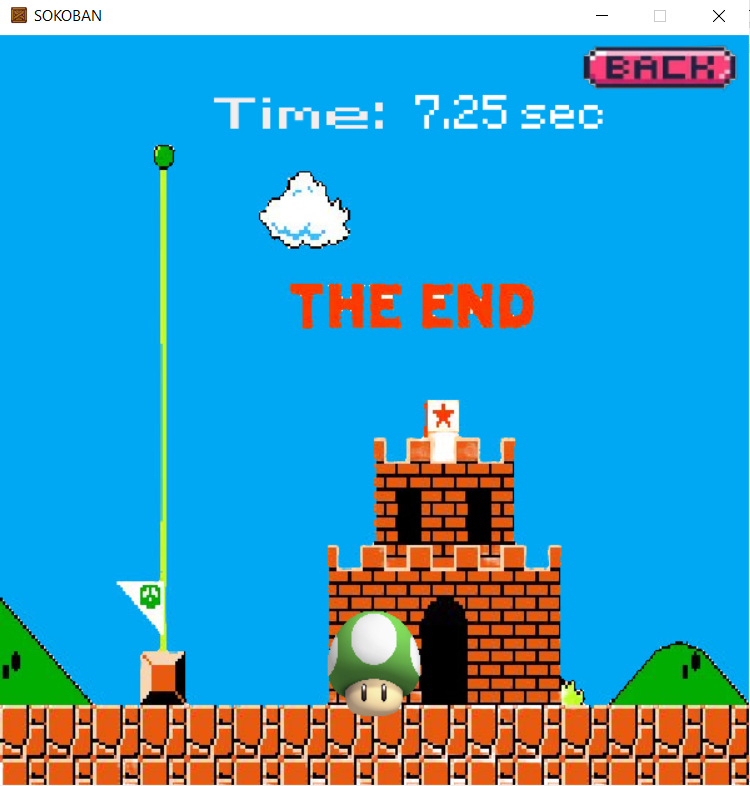
**Select level**

Click chuột phải vào select level để mở lên giao diện chính.Tại giao diện chính, có những thao tác cơ bản như sau:

Khi ấn phím Enter, căn cứ vào mode được chọn hiện tại (Astar hay BFS) mà sẽ tiến hành chạy thuật toán để tìm đường đi.

Sau khi tìm thấy đường đi, phần giao diện sẽ tiến hành thể hiện các bước đi ứng với kết quả và tự đó nhân vật sẽ tìm được đường để đẩy các “đồng tiền” vào những “lá cờ”.



Sau khi chạy xong, đồng tiền đã vào đúng vị trí, ấn phím enter để “về đích”

BFS ( if mode == “BFS) Astar ( if mode == “Astar”)

Time: Thời gian để thuật toán giải ra kết quả.

Ta có thể chạy 02 thuật toán trên các map khác nhau để thu về được thời gian giải câu đó ở mỗi thuật toán. Từ đó có thể thấy thời gian giải bài toán của Astar là nhanh hơn so với BFS.

# 3. Cấu trúc project.

Với từng level, ta có các problem tương ứng. Quy định như sau:

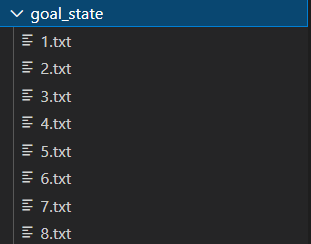
1: Chướng ngại vật

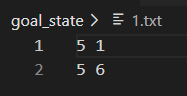
0: Đường đi được

g: Goal state ( đích )

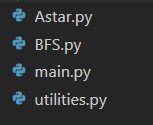
\*: Đồng tiền ( vật )

x: Cây nấm ( nhân vật )



****

Mỗi problem đều có 01 goal state tương ứng

****

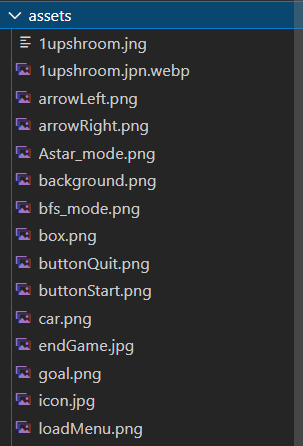
Có 4 file chính trong đó:

+ Astar: chứa giải thuật Astar

+ BFS.py: chứa giải thuật BFS

+ Utilities.py: chứa các support function

+ Main.py: nơi chạy chương trình, chứa code cài đặt đồ họa bằng pygame

Ngoài ra còn có 1 folder assets chứa hình ảnh, phục vụ cho giao diện:

# 4. Các class, function:

## 4.1. sokubanProblem:

Text

Description automatically generatedProblem chứa thông tin về trạng thái khởi đầu, các điểm goal của trò chơi (vị trí đẩy thùng tới).

Chứa các hàm liên quan đến giải quyết các vấn đề:

## 4.1.1. Hàm actions

Text

Description automatically generated

***Input:*** state bất kỳ.

***Output:*** Các actions mà nhân vật có thể thực hiện được trong input state.

***Chức năng:*** Tìm các actions mà nhân vật có thể thực hiện được trong input state (nhân vật có thể thực hiện các actions: UP, DOWN, LEFT, RIGHT. ).

* Text

  Description automatically generatedHàm **can\_move**

***Input:*** maze (state hiện tại), pos (vị trí tiếp theo của nhân vật khi thực hiện action), action (action tương ứng cần xem xét).

***Output:*** True nếu nhân vật có thể di chuyển với action tương ứng, ngược lại là False.

***Chức năng:*** giúp xác định xem nhân vật có thể di chuyển bằng cách thực hiện hành động tiếp theo hay không. Nhân vật không thể di chuyển khi vượt quá biên, khi đẩy trúng nhiều hơn 2 thùng liên tiếp, khi đụng vào tường hoặc khi đẩy thùng đụng vào tường.

## 4.1.2. Hàm result.

Text

Description automatically generated

***Input***: State bất kỳ, action.

***Output:*** State mới.

***Chức năng:*** tìm state tiếp theo sau khi nhân vật thực hiện action tương ứng trong input state.

* Text

  Description automatically generatedHàm **get\_player\_position**

***Input:*** state bất kỳ.

***Output:*** Vị trí nhân vật trong state đó.

***Chức năng:*** Tìm vị trí nhân vật trong input state.

## Text Description automatically generated4.1.3. Hàm goal\_test

***Input:*** State bất kỳ.

***Output:*** True nếu tất cả các thùng đều đúng vị trí, ngược lại là False.

***Chức năng:*** Kiểm tra xem các thùng trong input state có được đặt đúng ở các vị trí goal tương ứng hay chưa.

* Text

  Description automatically generatedHàm **get\_box\_position**

***Input:*** state bất kỳ.

***Output:*** danh sách vị trí các thùng trong state đó.

***Chức năng:*** Tìm vị trí các thùng trong input state.

## 4.2. Class BFSNode

Mỗi Node chứa thông tin về state hiện tại, state trước đó, action dẫn đến state hiện tại, path\_cost, độ sâu của node hiện tại ( = độ sâu của node trước đó + 1)

Text

Description automatically generated

Chứa các hàm liên quan đến từng Node

## Text Description automatically generated4.2.1. Hàm expand

***Input:*** Problem cần giải quyết.

***Output:*** Các child node của node hiện tại.

***Chức năng:*** Tìm các child node của node hiện tại khi thực hiện các action tương ứng với state của Node hiện tại.

## 4.2.2. Hàm child\_node

***Input:*** Problem cần giải quyết, action cần thực hiện.

***Output:*** Node sau khi thực hiện action.

***Chức năng:*** Tìm Node con của node hiện tại sau khi thực hiện action tương ứng.

## 4.2.3. Hàm solution

Text

Description automatically generated

***Input:*** không.

***Output:*** các state cần thực hiện để đạt được goal.

***Chức năng:*** tìm các state cần thực hiện để đẩy tất cả thùng vào các vị trí goal tương ứng.

## 4.3. Class AstarNode

Mỗi Node đều chứa thông tin về state hiện tại, state trước đó dẫn tới state hiện tại, vị trí các điểm đích, giá trị f, g, n (khởi tạo bằng 0).

Text

Description automatically generated

Các hàm tương ứng.

## A screenshot of a computer Description automatically generated with medium confidence4.3.2. Hàm cal\_heuristic:

***Input:*** không có.

***Output:*** Không có.

***Chức năng:*** thực hiện tính toán hệ số heuristic cho từng Node tương ứng. Hệ số heuristic tại mỗi state bằng tổng khoảng cách giữa vị trí từng thùng đến vị trí đích gần nhất với thùng đó (khoảng cách đường chim bay).

* Text

  Description automatically generatedHàm **get\_goal\_nearest\_box\_dis**

***Input:*** box\_pos(Vị trí một thùng bất kỳ), goal\_poses (Tất cả vị trí goal trong state tương ứng).

***Output:*** Khoảng cách nhỏ nhất giữa box\_pos với goal\_pos nào đó.

***Chức năng:*** Tìm điểm goal sao cho khoảng cách từ box đến goal là nhỏ nhất theo đường chim bay.

## Text Description automatically generated4.3.3. Hàm cal\_f

***Input:*** không.

***Output:*** không.

***Chức năng:*** tính toán hệ số g, h và f = g + h và lưu lại cho từng node.

## Text Description automatically generated4.4. Hàm Bfs

***Input:*** Problem cần giải quyết.

***Output:*** Node chứa state kết quả, thời gian thực thi thuật toán.

***Chức năng:*** Thực hiện giải thuật BFS tìm kiếm bằng cách duyệt các Node theo chiều rộng để tìm ra Node chứa state kết quả.

## Text Description automatically generated4.5. Hàm astar

***Input:*** Problem cần giải quyết.

***Output:*** Node chứa state kết quả, thời gian thực thi thuật toán.

***Chức năng:*** Thực hiện thuật toán tìm kiếm Astar dựa trên hệ số f của mỗi Node (ưu tiên f nhỏ nhất) để tìm ra Node chứa state kết quả.

TÀI LIỆU THAM KHẢO

* Rockwell Anyoha, (28/8/2017). The History of Artificial Intelligence. Retrieved from [https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/](https://aitopics.org/misc/brief-history)
* Wikipedia, *Artificial intelligence*. Truy cập ngày 5/5/2022 từ <https://wikipedia.org/wiki/Artificial_intelligence>
* (Ngày 3/7/2019). *Time complexity of Uniform-cost search*. Truy cập ngày 5/5/2022, từ <https://intellipaat.com/community/3601/time-complexity-of-uniform-cost-search>
* *Time/Space Complexity of Depth First Search*. Truy cập ngày 7/5/2022, từ <https://stackoverflow.com/questions/36479640/time-space-complexity-of-depth-first-search>
* Analyticsvidhya, *Introduction to Reinforcement Learning for Beginners*. Truy cập từ [https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners](https://www.analyticsvidhya.com/blog/2021/02/introduction-to-reinforcement-learning-for-beginners%20)
* Geeksforgeeks, A\* Search Algorithm. Truy cập ngày 9/5/2022, từ [https://www.geeksforgeeks.org/a-search-algorithm/](https://www.geeksforgeeks.org/a-search-algorithm/%20)
* Trần Nhật Quang (2022). *Bài giảng môn Trí tuệ nhân tạo.*