# CHƯƠNG 1: TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

* 1. **Khái niệm về trí tuệ nhân tạo**

Trí tuệ nhân tạo (tiếng anh là Artificial Intelligence (AI)) còn được gọi là trí thông minh nhân tạo, là một lĩnh vực của khoa học máy tính, tập trung vào việc tạo ra những cỗ máy thông minh, có thể hoạt động và phản ứng như con người. Thông thường, thuật ngữ trí tuệ nhân tạo được sử dụng để mô tả những cỗ máy có thể bắt chước chức năng "nhận thức" mà con người phải sử dụng trí não để thực hiện, như học tập, giải quyết vấn đề.

* 1. **Vai trò về trí tuệ nhân tạo**

Trí tuệ nhân tạo bao quát rất nhiều lĩnh vực nghiên cứu. Nó nghiên cứu từ các lĩnh vực tổng quát như máy nhận biết, suy luận logic, đến các bài toán như chơi cờ, chứng minh định lý.

Trong các lĩnh vực khác trí tuệ nhân tạo được dùng kỹ thuật hệ thống hoá và tự động hoá các xử lý tri thức cũng như các phương pháp thuộc lĩnh vực mang tính con người.

Trí tuệ nhân tạo nghiên cứu kỹ thuật làm cho máy tính có thể “suy nghĩ một cách thông minh” và mô phỏng quá trình suy nghĩ của con người khi đưa ra những quyết định, lời giải. Trên cơ sở đó, thiết kế các chương trình cho máy tính để giải quyết bài toán.

* 1. **Phân loại về trí tuệ nhân tạo**

Trí tuệ nhân tạo được phân loại theo nhiều cách khác nhau, dưới đây là hai ví dụ điển hình về phân loại trí tuệ nhân tạo.

Đầu tiên là phân loại trí tuệ nhân tạo AI theo hệ thống bao gồm trí tuệ nhân tạo mạnh hoặc yếu. Trí tuệ nhân tạo yếu hay còn được gọi là Narrow AI, là hệ thống trí tuệ nhân tạo được thiết kế và đào tạo cho các tác vụ cụ thể. Các trợ lý ảo chẳng hạn như Siri của Apple là một dạng trí tuệ nhân tạo yếu.

Trí tuệ nhân tạo mạnh còn được gọi là Artificial General Intelligence hay trí tuệ nhân tạo tổng hợp, là hệ thống AI được trang bị khả năng nhận thức tổng quát của con người để khi thực hiện các tác vụ không quen thuộc, nó đủ thông minh để tìm ra các giải pháp. Phép thử Turing được phát triển bởi nhà toán học Alan Turing vào năm 1950 là phương pháp được sử dụng để xác thực xem một máy tính có thể có những suy nghĩ giống con người hay không, mặc dù phương pháp này gây nhiều tranh cãi.

Ví dụ thứ hai là từ Arend Hintze, một trợ lý giáo sư sinh học hợp nhất và khoa học máy tính và kỹ thuật tại Đại học bang Michigan. Ông phân loại trí tuệ nhân tạo AI thành 4 loại, từ loại hệ thống AI hiện nay đến các hệ thống cảm giác, mà chưa tồn tại.

* **Loại 1: Máy phản ứng.**
* **Loại 2: Bộ nhớ hạn chế**
* **Loại 3: Lý thuyết.**
* **Loại 4: Tự nhận thức.**
  1. **Ứng dụng về trí tuệ nhân tạo**

Một số ứng dụng sử dụng trí tuệ nhân tạo AI bao gồm hệ chuyên gia (expert system), các ứng dụng nhận diện giọng nói và các ứng dụng machine vision.

Trí tuệ nhân tạo, AI được đặt tên bởi nhà khoa học máy tính người Mỹ, John McCarthy vào năm 1956 tại Hội nghị Dartmouth. Ngày nay thuật ngữ AI được bổ sung thêm cả tự động hóa quy trình bằng Robot và Robot thực tế.

Thời gian gần đây nổi lên thuật ngữ mới có tên Big Data hay gia tăng tốc độ, kích thước và thu thập các dữ liệu doanh nghiệp đa dạng. AI có thể thực hiện các tác vụ như xác định các mẫu trong dữ liệu hiệu quả hơn con người, cho phép các doanh nghiệp hiểu rõ hơn về dữ liệu của mình.

* 1. **Các thuật toán nổi bật**
* Tìm kiếm theo chi phí cực tiểu (Uniform – cost search)
* Tìm kiếm giới hạn độ sâu (Depth – Limited – Search)
* Tìm kiếm sâu dần (Interative – Depth – Search)
* Giải thuật Best – first – search
* Thuật toán A Star
* Giải thuật tìm kiếm cục bộ
* Giải thuật leo đồi đơn giản
* Thuật toán leo đồi dốc đứng
* Thuật toán nhánh và cận
* Tìm kiếm có đối thủ
* Giải thuật MINIMAX
  1. **Ưu – nhược điểm**
* Ưu điểm:

+ Nhờ có AI mà nó đã thay thế được sức người nhờ có các máy móc hiện đại.

+ Công nghệ AI hiện tại có thể hỗ trợ y tế và chẩn đoán chính xác hơn nhờ các suy luận thông kế vượt trội so với đánh giá lâm sàng của chuyên gia con người.

* Nhược điểm:

+ Nó có thể gây nguy hiểm khi sử dụng bởi kẻ xấu, hoặc khi không được sử dụng cẩn thận an toàn và trong các tình huống không lường trước được.

+ Nhờ công nghệ AI quá hiện đại, có thể thế hệ mai máy móc sẽ thay thế con người và từ đó con người sẽ bị thất nghiệp.

**CHƯƠNG 2: TỔNG QUAN VỀ THUẬT TOÁN A STAR**

* 1. **Ý tưởng thuật toán A Star**

Xét bài toán tìm đường-bài toán mà A\* thường được sử dụng để giải. Thuật toán A\* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các [thuật toán tìm kiếm có thông tin](https://vi.wikipedia.org/w/index.php?title=Thu%E1%BA%ADt_to%C3%A1n_t%C3%ACm_ki%E1%BA%BFm_c%C3%B3_th%C3%B4ng_tin&action=edit&redlink=1" \o "Thuật toán tìm kiếm có thông tin (trang chưa được viết)) (*informed tìm kiếm thuật toán*), nó chỉ xây dựng các tuyến đường "có vẻ" dẫn về phía đích.

Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A\* sử dụng một "đánh giá heuristic" về khoảng cách từ điểm bất kỳ cho trước tới đích. Trong trường hợp tìm đường đi, đánh giá này có thể là khoảng cách đường chim bay - một đánh giá xấp xỉ thường dùng cho khoảng cách của đường giao thông.

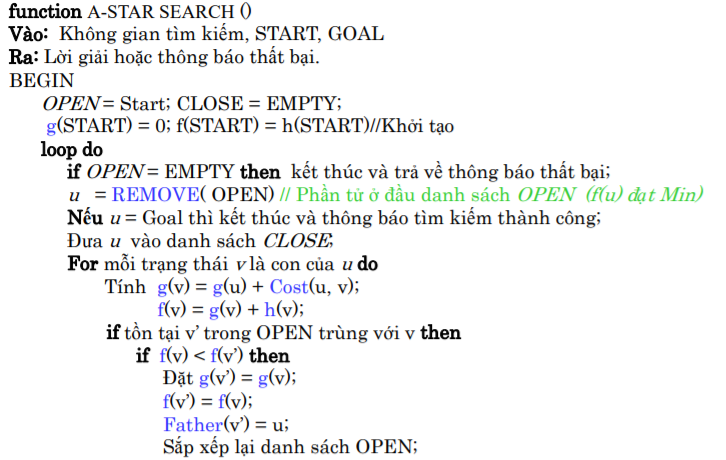
Điểm khác biệt của A\* đối với [tìm kiếm theo lựa chọn tốt nhất](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_l%E1%BB%B1a_ch%E1%BB%8Dn_t%E1%BB%91t_nh%E1%BA%A5t" \o "Tìm kiếm theo lựa chọn tốt nhất) là nó còn tính đến khoảng cách đã đi qua. Điều đó làm cho A\* "đầy đủ" và "tối ưu", nghĩa là, A\* sẽ luôn luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như vậy. A\* không đảm bảo sẽ chạy nhanh hơn các thuật toán tìm kiếm đơn giản hơn. Trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay lại. Trong trường hợp đó, việc thử các nút theo thứ tự "gần đích hơn thì được thử trước" có thể gây tốn thời gian.

* 1. **Thuật toán A Star**

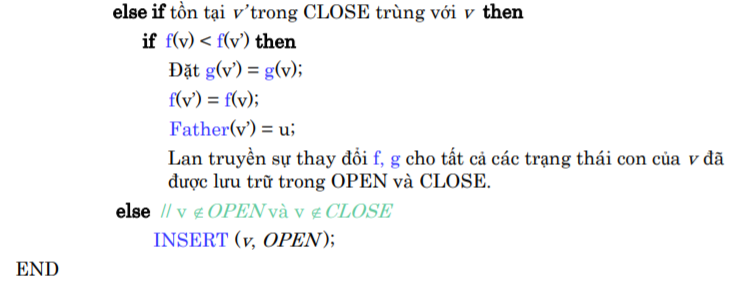
A\* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một [hàng đợi ưu tiên](https://vi.wikipedia.org/w/index.php?title=H%C3%A0ng_%C4%91%E1%BB%A3i_%C6%B0u_ti%C3%AAn&action=edit&redlink=1" \o "Hàng đợi ưu tiên (trang chưa được viết)) (*priority queue*). Thứ tự ưu tiên gán cho một đường đi {\displaystyle x}được quyết định bởi hàm {\displaystyle f(x)=g(x)+h(x)}.

Trong đó, {\displaystyle g(x)}là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. {\displaystyle h(x)}là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ {\displaystyle x}. Ví dụ, nếu "chi phí" được tính là khoảng cách đã đi qua, [khoảng cách đường chim bay](https://vi.wikipedia.org/wiki/Kho%E1%BA%A3ng_c%C3%A1ch_%C4%91%C6%B0%E1%BB%9Dng_chim_bay" \o "Khoảng cách đường chim bay) giữa hai điểm trên một bản đồ là một đánh giá heuristic cho khoảng cách còn phải đi tiếp.

Hàm {\displaystyle f(x)}có giá trị càng thấp thì độ ưu tiên của {\displaystyle x}*càng cao (do đó có thể sử dụng một cấu trúc*[*heap tối thiểu*](https://vi.wikipedia.org/w/index.php?title=Heap_(c%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u)&action=edit&redlink=1)*để cài đặt hàng đợi ưu tiên này)*.



Hình 2. 1: Thuật toán A\*



Hình 2. 2: Thuật toán A\*

* 1. **Độ phức tạp**

[Độ phức tạp thời gian](https://vi.wikipedia.org/wiki/L%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%99_ph%E1%BB%A9c_t%E1%BA%A1p_t%C3%ADnh_to%C3%A1n) của A\* phụ thuộc vào đánh giá heuristic. Trong trường hợp xấu nhất, số nút được mở rộng theo [hàm mũ](https://vi.wikipedia.org/w/index.php?title=Th%E1%BB%9Di_gian_l%C5%A9y_th%E1%BB%ABa&action=edit&redlink=1" \o "Thời gian lũy thừa (trang chưa được viết)) của độ dài lời giải, nhưng nó sẽ là [hàm đa thức](https://vi.wikipedia.org/w/index.php?title=Th%E1%BB%9Di_gian_%C4%91a_th%E1%BB%A9c&action=edit&redlink=1" \o "Thời gian đa thức (trang chưa được viết)) khi hàm heuristic *h* thỏa mãn điều kiện sau:{\displaystyle |h(x)-h^{\*}(x)|\leq O(\log h^{\*}(x))}

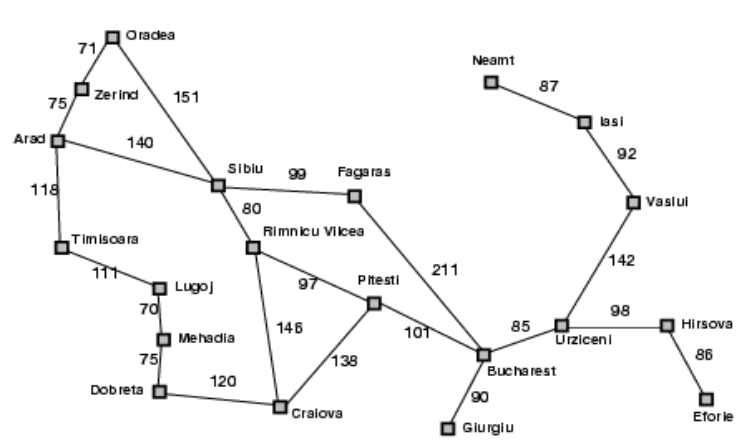
* Trong đó {\displaystyle h^{\*}} là heuristic tối ưu, nghĩa là hàm cho kết quả là chi phí chính xác để đi từ {\displaystyle x}tới đích. Nói cách khác, sai số của *h* không nên tăng nhanh hơn [logarit](https://vi.wikipedia.org/wiki/L%C3%B4garit" \o "Lôgarit) của "heuristic hoàn hảo" {\displaystyle h^{\*}} - hàm trả về khoảng cách thực từ *x* tới đích.
* Độ phức tạp là: O(bdelta)
* Trong đó: delta = h\* - h. Với h\* là chi phí thực, h là ước lượng heuristic
* Vấn đề sử dụng bộ nhớ của A\* còn rắc rối hơn độ phức tạp thời gian. Trong trường hợp xấu nhất, A\* phải ghi nhớ số lượng nút tăng theo hàm mũ. Một số biến thể của A\* đã được phát triển để đối phó với hiện tượng này, một trong số đó là A\* lặp sâu dần (*iterative deepening A\**), A\* bộ nhớ giới hạn (*memory-bounded A\** - MA\*) và A\* bộ nhớ giới hạn đơn giản (*simplified memory bounded A\**).
* Một thuật toán tìm kiếm có thông tin khác cũng có tính chất tối ưu và đầy đủ nếu đánh giá heuristic của nó là thu nạp được (*admissible*). Đó là [tìm kiếm đệ quy theo lựa chọn tốt nhất](https://vi.wikipedia.org/w/index.php?title=T%C3%ACm_ki%E1%BA%BFm_%C4%91%E1%BB%87_quy_theo_l%E1%BB%B1a_ch%E1%BB%8Dn_t%E1%BB%91t_nh%E1%BA%A5t&action=edit&redlink=1" \o "Tìm kiếm đệ quy theo lựa chọn tốt nhất (trang chưa được viết)) (*recursive best-first search* - RBFS).
  1. **Ưu – nhược điểm**
* Ưu điểm:

+ Một thuật giải linh động, tổng quát, trong đó hàm chứa cả [tìm kiếm chiều sâu](https://www.stdio.vn/article/thuat-toan-depth-first-search-APnHi1), [tìm kiếm chiều rộng](https://www.stdio.vn/article/thuat-toan-breadth-first-search-sBPnH) và những nguyên lý Heuristic khác. Nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói A\* chính là thuật giải tiêu biểu cho Heuristic.

* Nhược điểm:

+ A\* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua.

* 1. **Ví dụ minh họa**



Hình 2.3 Biểu đồ bài toán

- Sau đây là từng bước hoạt động của thuật toán A\* trong việc tìm đường đi ngắn nhất từ Oradea đến Hirsova:

- Ban đầu :

OPEN = {(Oradea,g= 0,h’= 0,f’= 0)}

CLOSE = {}

- Do trong OPEN chỉ chứa một thành phố duy nhất nên thành phố này sẽ là thành phố tốt nhất. Nghĩa là Tmax = Oradea.Ta lấy Orades ra khỏi OPEN và đưa vào CLOSE.

OPEN = {}

CLOSE = {(Oradea,g= 0,h’= 0,f’= 0)}

- Từ Oradea có thể đi đến được 2 thành phố là Sibiu và Zerind. Ta lần lượt tính giá trị f’, g và h’ của 2 thành phố này. Do cả 2 nút mới tạo ra này chưa có nút cha nên ban đầu nút cha của chúng đều là Oradea.

h’(Sibiu) = 253

g(Sibiu) = g(Oradea)+cost(Oradea,Sibiu)

= 0+151= 151

f’(Sibiu) = g(Sibiu)+h’(Sibiu)

= 151+253 = 404

Cha(Sibiu) = Oderea

Tương tự ta có f’(Zerind) = 71 +374 = 445

Cha(Zerind) = Oderea

Sau đó thêm 2 nút vào OPEN và cập nhật Sbiu vào CLOSE

.

- Do f’ của Sibiu nhỏ nhất nên lúc này Tmax = Sibiu , g=151. Từ Sibiu đi đến được 4 thành phố là Arad, Oderea, Vilcea, Fagaras.

- Xét tương tự như trên ta lần lượt có: f’(Arad) = 140+151+366 = 657 , f’(Vilcea) = 80+151+193 = 424, f’(Fagaras) = 99+151+178 = 428, f’(Oderea) = 151+151+380 = 682

- Lúc này do Oderea đã có trong CLOSE nên ta ko cập nhật vào OPEN và thêm các thành phố mới vào OPEN

- Do f’ của Vilcea nhỏ nhất nên Tmax = Vilcea, g=231. Từ Vilcea đi được đến 3 thành phố là Piesti, Craiova, Sibiu

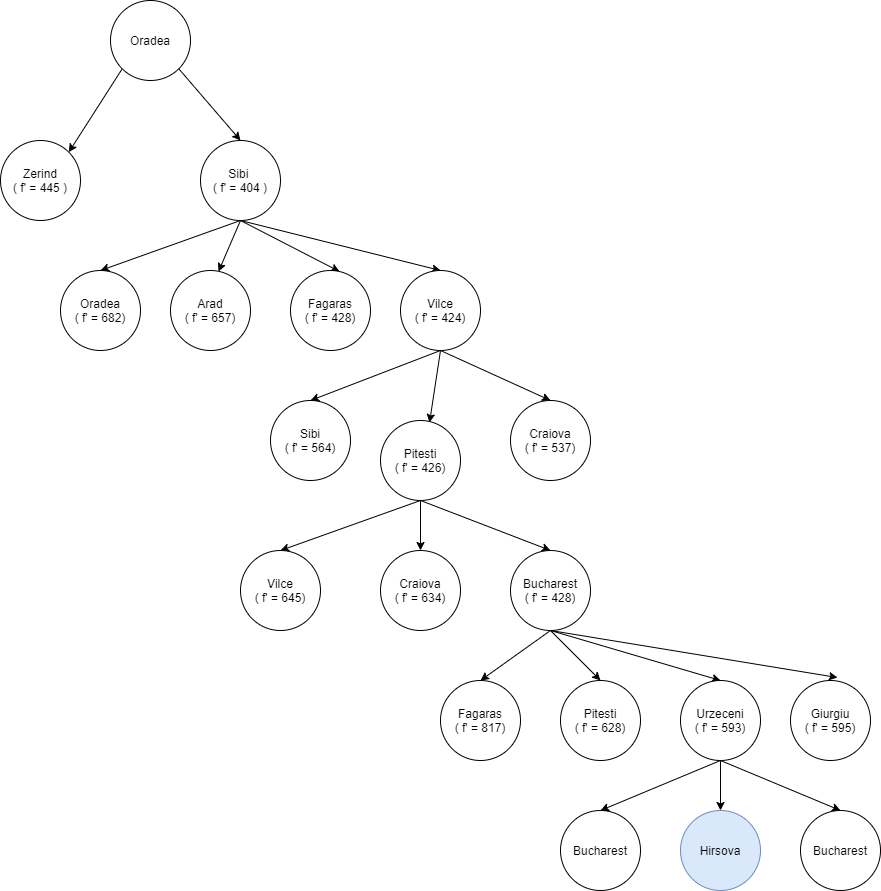
Tiếp tục làm như trên ta có: f’(Piesti) = 231+97+98 = 426, f’(Craiova) = 231+145+160 =536, f’(Sibiu) = 231+80+253 = 564.

- Đến đây, Sibiu đã có trong CLOSE nên ko cho vào OPEN, cập nhật thêm các thành phố mới vào trong OPEN và Piesti vào CLOSE

- Lặp lại các bước như trên ta có thêm Burchaset(g=429), Urziceni(g=514) và cuối cùng là Hirsova(g=612).

- Đến đây ta dừng lại , và có được đường đi ngắn nhất từ Oderea đến HIrsova là tập hợp trong CLOSE:

- Oderea, Sibiu, Vlicea, Burchaset, Urziceni, Hirsova với g=612

. 

Hình 2.4 Minh họa bài toán

**CHƯƠNG 3: ỨNG DỤNG THUẬT TOÁN VÀO BÀI TOÁN**

* 1. **Mô tả bài toán**

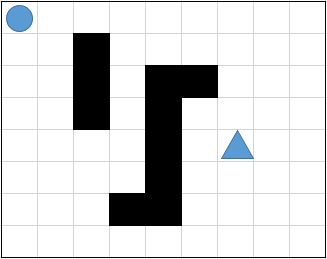
Mô hình và các chức năng:

* + Khi người chơi khởi động chương trình game lên thì màn hình sẽ hiện thị lên một giao diện game. Bên tay trái gồm cái nút lệnh cho người chơi lựa chọn đó là nút bắt đầu chơi, nút hiện thị kết quả và núi thoát game. Khi người chơi lựa chọn vào nút bắt đầu chơi thì màn hình game sẽ bắt đầu chạy và thời gian chơi sẽ bắt đầu được tính.
  + Nhiệm vụ của người chơi đó là di chuyển nhân vật của mình tới được vị trí đích bằng cách tháo tách mũi tên đi lên, đi xuống, sang phải và sang trái để có thể vừa tìm được đường đi và đến đích với tổng thời gian là ngắn nhất.
  + Nếu người chơi không thế tìm thấy được đường đi để tới được điểm đích thì có thể sử dụng trợ giúp từ chương trình để ta có thể thuận tiền đến điểm đích một cách nhanh chóng nhất.
  + Nếu người chơi muốn thoát khỏi game đang chơi thì chọn vào nút thoát từ chương trình bên góc trên cùng bên phải
  1. **Các bước giải quyết bài toán**

\* Ví dụ :

Cho một bàn cờ kích thước n x n. Hãy tìm đường đi ngắn nhất từ điểm 〇 đến điểm △ biết rằng:

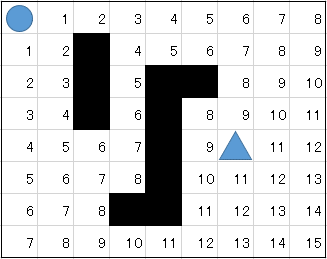
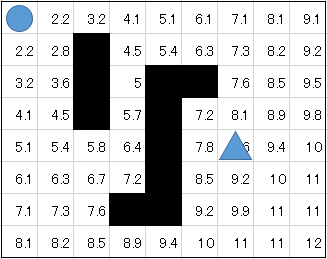
* Các ô trắng là đường đi
* Các ô đen là vật cản, không đi được
* Tại một ô bất kỳ, chỉ có thể đi lên, xuông, trái, phải nếu không bị cản. Không được đi chéo



Hình 2.6 Bàn cờ với kích thước 9\*8

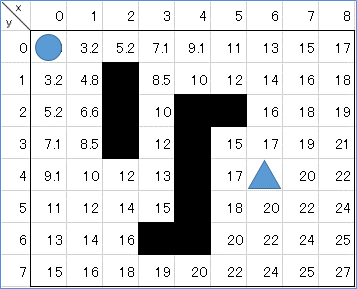
- Trong bài toán trên ta thấy rằng đường đi ngắn nhất là đường chéo nối từ 〇 đến △ chiều dài đường chéo được tính bằng công thức h(x) =

- Khi di chuyển qua mỗi ô thì giá trị của ô sẽ tăng lên 1 đơn vị

Hình 2.7 g(x) Hình 2.8 h(x) =

- Vì là f(x) = g(x) + h(x) nên ta có bảng như sau:



Hình 2.9 f(x) = g(x) + h(x)

- Tiếp theo từ vị trí 0 sẽ duyệt các ô lân cận nó để tìm ra đường ngắn nhất

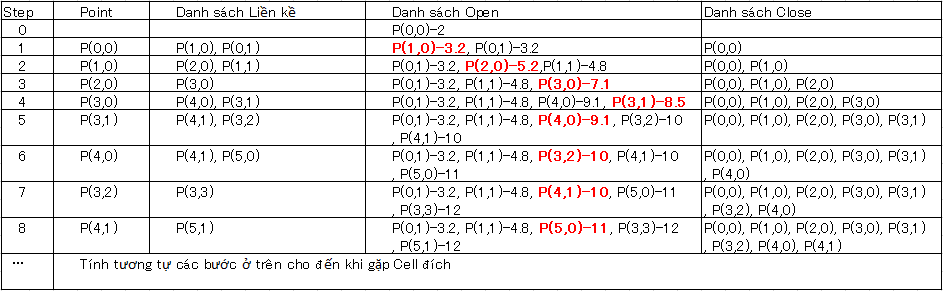
Ví dụ:

- Xét lại điểm P(0,0) thì có 2 điểm liên kề nó là bên phải P(1,0) và bên dưới P(0,1) và giá trị f(x) = g(x) + h(x).

- Vì 2 giá trị bằng nhau nên chọn điểm nào cũng được. Ở Step 2 thì đang chọn điểm P(1,0)

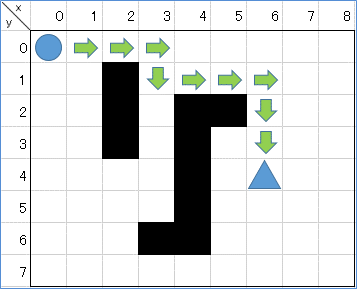
- Điểm P(1,0) có 2 điểm liền kề có giá trị lần lượt là: P (2,0) =5,2 ; P(1,1)=4,8 cộng với P(0,1) đang có sẵn thì ta có 3 điểm P(0,1)=3,2 ; P (2,0) =5,2 ; P(1,1)=4,8.

- Giá trị P (2,0) =5,2 là giá trị gần với giá trị 8.6 nhất. nên P(2,0) được chọn làm giá trị xét tiếp theo. tương tự các bước trên ta được bảng:



Bảng 2.1 Bảng tìm đường đi

- Sau khi chạm được đến điểm △ thì ta được sơ đồ đường đi ngắn nhất như sau:



Hình 2.10 Đường đi ngắn nhất