Có 5 thành phần của 1 bài toán tìm kiếm:

1. Initial state (trạng thái ban đầu): trạng thái ban đầu của vấn đề, cần thiết để xác định vị trí của chúng ta so với mục tiêu cụ thể cần đạt nào đó nhìn như thế nào.
2. Action (Hành động): những hành động mà chúng ta có thể thực hiện để giải bài toán khi đang ở một tráng thái cụ thể.
3. Transition (chuyển hóa): tích hợp, thực hiện những hành động ở bước Action khi đó sẽ cho ra 1 trạng thái mới, từ trạng thái này xác định xem bước tiếp theo là gì, xem xét trạng thái này nhìn như thế nào.
4. Goal test (kết quả thử nghiệm): Kiểm tra xem Transition đã thực hiện có đạt được yêu cầu chưa, trạng thái hiện tại có phải trạng thái đích không, nếu được thì tiến hành bước tiếp theo để xác định chi phí.
5. Path costing (chi phí đường đi): Tất cả các chi phí bao gồm chi phí phần mềm, phần cứng và con người cần thiết để đạt kết quả đó, hoặc một chi phí giả định trong bài toán cụ thể.

Nhìn chung để giải quyết một bài toán tìm kiếm thực chất chính là chúng ta đi duyệt đồ thị (graph traversal) và sử dụng các thuật toán như DFS và BFS. Những bài toán liên quan tới tìm kiếm là những bài toán cần được giải quyết có hệ thống.

Khi thực hiện giải quyết bài toán tìm kiếm, chúng ta cần biết được trạng thái đầu tiên đang là gì, trạng thái đích nhìn như thế nào, ở trạng thái hiện tại, làm sao để biết được bước tiếp theo là gì để giải bài toán.

Phân loại bài toán tìm kiếm:

Uninformed Search hay thuật toán tìm kiếm mù là thuật toán chỉ sử dụng những thành phần có sẵn trong định nghĩa của vấn đề, có nghĩa là thuật toán này chỉ biết đâu là trạng thái đích và chưa đích chứ không thể xem xét tình trạng bên trong để ước tính xem ta có gần hơn với lời giải hay chưa.

Một số ví dụ của thuật toán này là Breadth-First Search, Uniform-Cost Search, Depth-First Search, Depth-Limited Search, Iterative Deepening, and Bidirectional Search.

Informed Search khác so với thuật toán tìm kiếm mù ở điểm có sử dụng thêm kiến thức của hàm heuristic, ý nghĩa của hàm này là nó sẽ có tham số đầu vào là một trạng thái nào đó và cho ra kết qủa là một sự ước tính xem trạng thái này như thế nào so với đích đến, có gần hơn hay không. Nhờ hàm heuristic, thuật toán tìm kiếm có thể phân biệt được các trạng thái ở giữa với nhau, đâu là trạng thái hứa hẹn hơn, gần hơn với đích.

Hay nói cách khác, hàm heuristic là một hàm ước tính chi phí của con đường đi ngắn nhất giữa trạng thái của một node cụ thể và trạng thái đích.

Điển hình nhất của thuật toán là thuật toán A\*, ngoài ra còn có các thuật toán như Best-First Search, Recursive Best-First Search, and Simplified Memory-bounded A\*.

DFS

- Ý tưởng chung: Là thuật toán duyệt các node ở mức sâu nhất trước, thuật toán bắt đầu ở node rễ (root), sau đó đi tới node sâu nhất của một cành xác định, trước khi đi tới cành khác.

- Mã giả:

Đặt node bắt đầu vào open\_set tượng trưng cho các node vừa khám phá ra

Đặt node bắt đầu vào stack closed\_set tượng trưng cho các node đã duyệt

while closed\_set không rỗng:

Lấy node đầu stack ra khỏi stack closed\_set

Nếu node là root:

Trả về true, kết thúc thuật toán

Nếu node không là root:

Tìm các node tiếp theo kề với root

Thêm các node đó vào stack closed\_set và open\_set nếu node, đó chưa được khám phá

Nếu duyệt hết đồ thị mà không tìm thấy search\_value:

Trả về false, kết thúc thuật toán

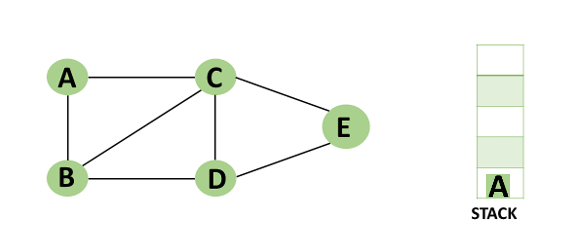
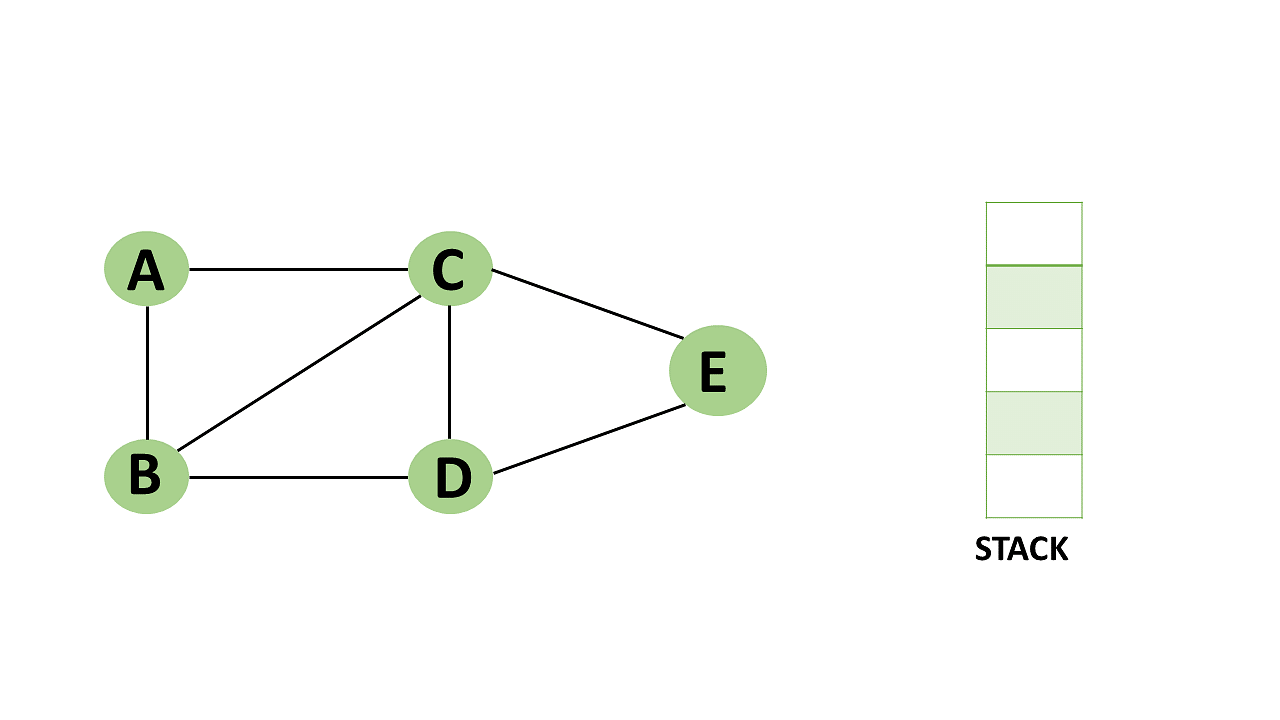
Đánh giá thuật toán:

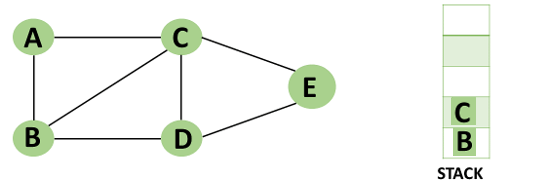
Nếu không gian là hữu hạn thuật toán này có thể giải quyết đầy đủ.

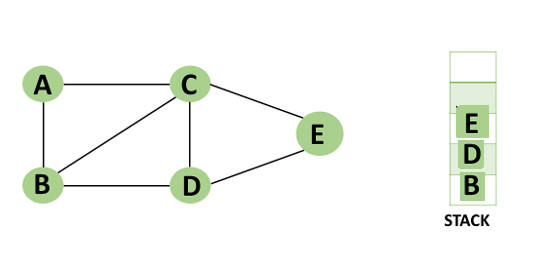
Nếu không gian là hữu hạn thì thuật toán này có thể tìm lời giải tối.

Độ phức tạp thời gian và không gian của thuật toán là O(V) với V là số đỉnh của đồ thị

Ví dụ minh họa:







BFS:

* Ý tưởng chung: Là thuật toán duyệt qua các node theo độ sâu, tất cả các node ở 1 độ sâu phải được mở trước khi tiến tới độ sâu tiếp theo.
* Mã giả:

Đặt node bắt đầu vào open\_set tượng trưng cho các node vừa khám phá ra

Đặt node bắt đầu vào queue closed\_set tượng trưng cho các node đã duyệt

while closed\_set không rỗng:

Lấy node đầu stack ra khỏi stack closed\_set

Nếu node là root:

Trả về true, kết thúc thuật toán

Nếu node không là root:

Tìm các node tiếp theo kề với root

Thêm các node đó vào stack closed\_set và open\_set,nếu node đó chưa được khám phá

Nếu duyệt hết đồ thị mà không tìm thấy search\_value:

Trả về false, kết thúc thuật toán

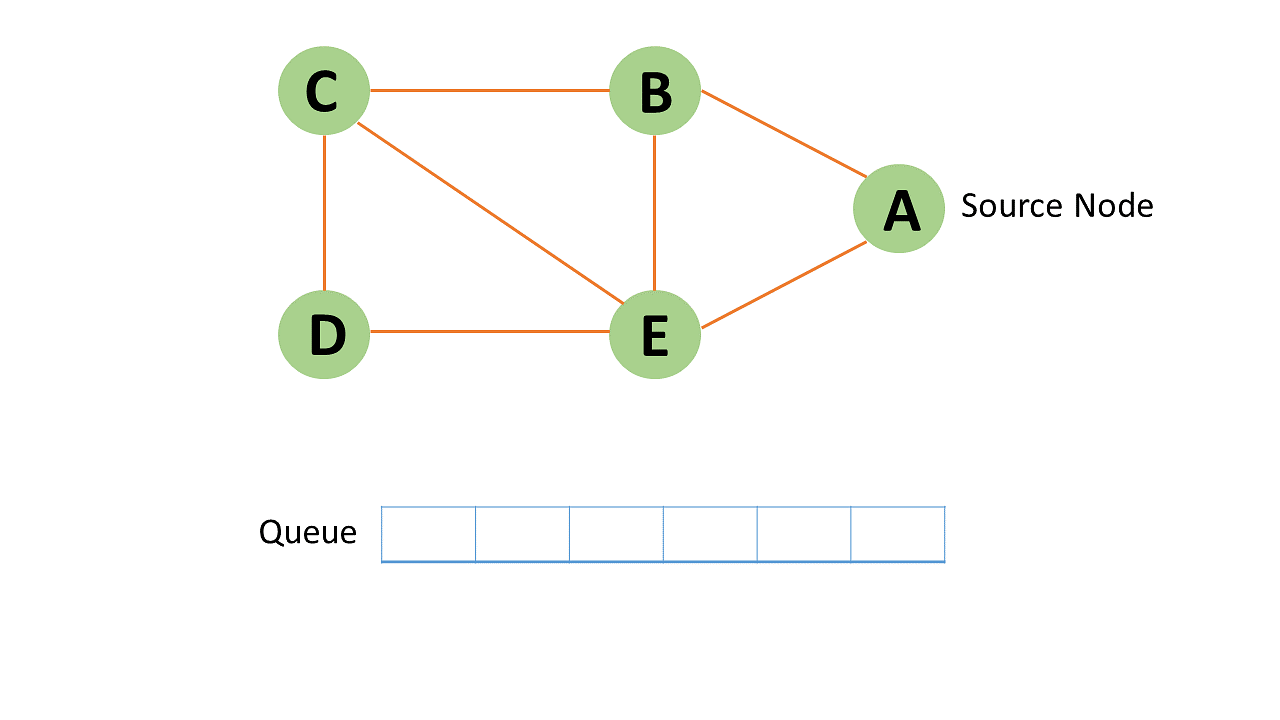
Đánh giá thuật toán:

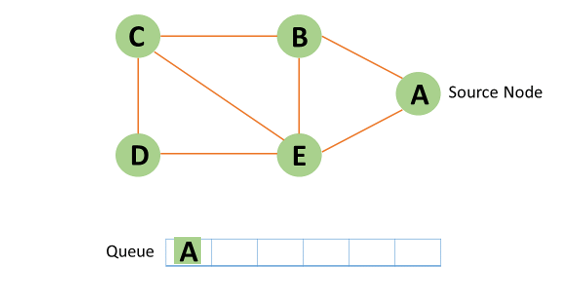
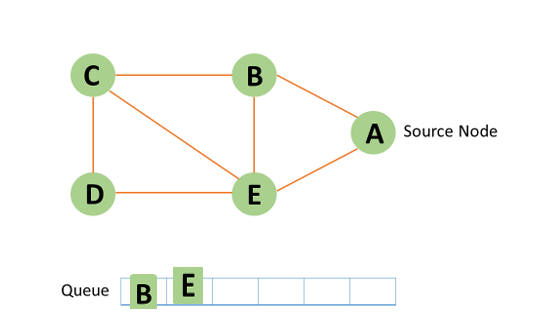
Nếu không gian là hữu hạn thuật toán này có thể giải quyết đầy đủ.

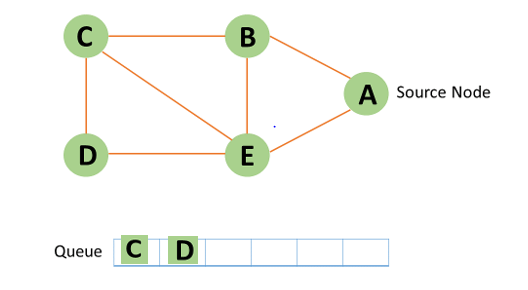
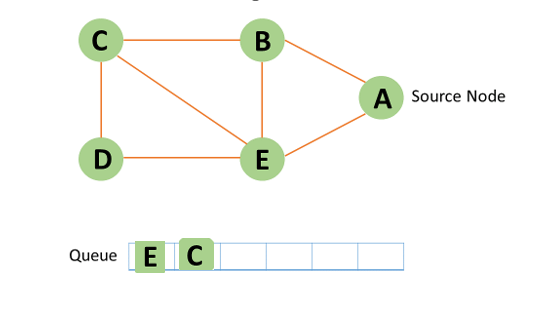
Nếu không gian là hữu hạn thì thuật toán này có thể tìm lời giải tối ưu.

Độ phức tạp thời gian và không gian của thuật toán là O(V) với V là số đỉnh của đồ thị

Ví dụ minh họa:







UCS:

* Ý tưởng chung:

Thuật toán duyệt các node có cạnh nối mang giá trị nhỏ nhất khi so với node ban đầu trước. Khi các cạnh đều có giá trị như nhau thuật toán này sẽ duyệt như BFS.

* Mã giả:

While True:

Nếu open\_set không còn node nào nữa, tức open\_set rỗng nhưng chưa trả về True thì tức đã duyệt qua mọi node và node cần tìm không tồn tại,

Trả về False

Tìm và chọn node đầu tiên có đường đi nhỏ nhất khi so với node ban đầu

Nếu node tìm được là node cần tìm

Trả vể True

Thêm node vừa tìm được vào closed\_set

Tìm các node kề và chi phí đường đi của nó

Nếu node kề này không tồn tại cả trong closed\_set và open\_set

Thêm node này vào cả hai tập closed\_set và open\_set

Nếu node này tồn tại trong open\_set

So sánh chi phí của node kề này với node hiện tại ,nếu node kề có chi phí nhỏ hơn

Thêm lại node này vào open\_set để xét đường đi khác

Pop node hiện tại khỏi open\_set để xét node tiếp theo

Đánh giá thuật toán:

Nếu không gian là hữu hạn thuật toán này có thể giải quyết đầy đủ.

Nếu không gian là hữu hạn thì thuật toán này có thể tìm lời giải tối.

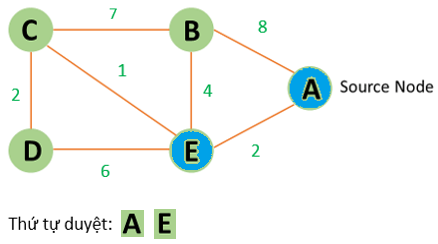
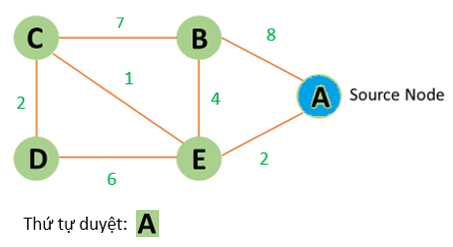
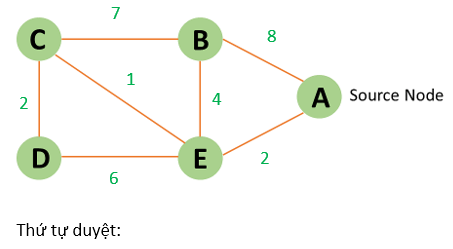
Độ phức tạp thời gian và không gian của thuật toán là O(b(1 + C / ε)) với:

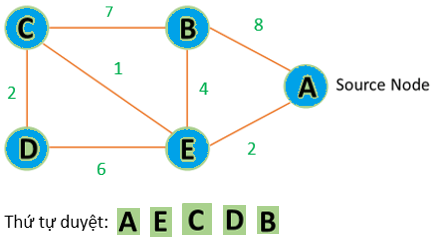
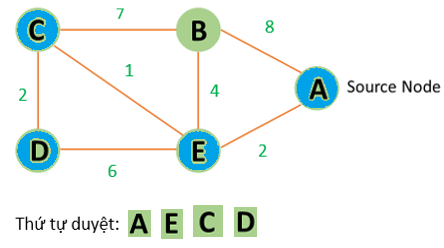
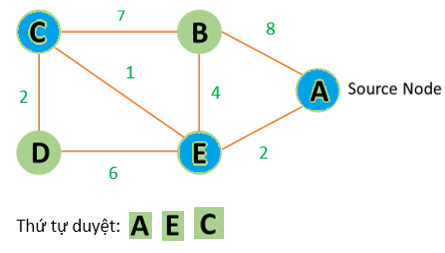
+ B là số node ở từng tầng (hệ số phân nhánh)

+ C là độ dài từ vị trí bắt đầu tới vị trí đích

+ ε là độ dài tối thiểu của 1 cạnh

Ví dụ minh họa:





A star:

* Ý tưởng chung: Là thuật toán có kiến thức nhờ vào hàm Heuristic, hàm Heuristic có công thức là f(n) = g(n) + h(n) trong đó g(n) là chi phí từ node khởi đầu đến node n và h(n) là chi phí đi từ node n tới node đích. Thuật toán A star vô cùng tối ưu vì tại mọi thời điểm trong thuật toán sẽ luôn ưu tiên mở node có f(n) thấp nhất.
* Mã giả:

While True:

Nếu open\_set không còn node nào nữa, tức open\_set rỗng nhưng chưa trả về True thì tức đã duyệt qua mọi node và node cần tìm không tồn tại,

Trả về False

Tìm node có f(n) nhỏ nhất

Nếu node tìm được là node cần tìm

Trả vể True

Thêm node vừa tìm được vào closed\_set

Tìm các node kề và tính toán f(n) của nó

Nếu node kề này không tồn tại cả trong closed\_set và open\_set

Thêm node này vào cả hai tập closed\_set và open\_set

Nếu node này tồn tại trong open\_set

So sánh f(n) của node kề này với node hiện tại ,nếu node kề có f(n) nhỏ hơn

Thêm lại node này vào open\_set để xét đường đi khác

Pop node hiện tại khỏi open\_set để xét node tiếp theo

* Đánh giá thuật toán:

Nếu không gian là hữu hạn và chi phí cố định thì thuật toán này có thể giải quyết đầy đủ.

Thuật toán này có thể tìm lời giải tối ưu khi:

h(n) là một heuristic có thể chấp nhận được

đồ thị Astar có tính nhất quán

Độ phức tạp về thời gian và không gian của thuật toán là O(bd) với:

+ B là hệ số phân nhánh

+ D là độ sâu

* Hàm Heuristic đề xuất:

Hàm sẽ ước tính chi phí giữa node hiện tại và node đích như sau:

Tính khoảng cách giữa node hiện tại và node đích dùng Euclid do đã có thông số tọa độ x, y, sau đó chia cho bán kính của 1 node để ước tính số lượng node nằm giữa node hiện tại và node đích. Cuối cùng là nhân vào chi phí của 1 node (giả sử mỗi node có cùng 1 chi phí trừ node bắt đầu)

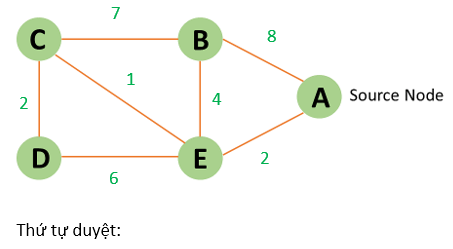
Công thức:

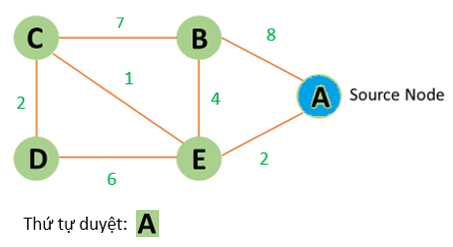
Như vậy ta sẽ ước tính được chi phí tại node n cách node đích là bao nhiêu.

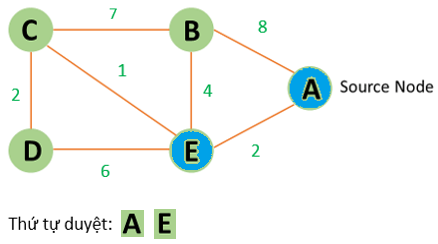
Heuristic này được chọn do không chỉ nó có tính phân biệt cao, mỗi node có một tọa độ x, y khác nhau. Kiến thức của hàm này còn đơn giản và quen thuộc, dễ dàng trong việc lập trình. Hàm này có thể chấp nhận được do có sử dụng công thức toán học Euclid đã được chứng minh chính xác.

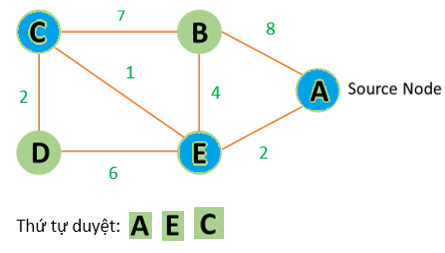
Một heuristic có thể chấp nhận được là khi giá trị trả về của nó không lớn hơn giá trị nhỏ nhất có thể trong thực tế

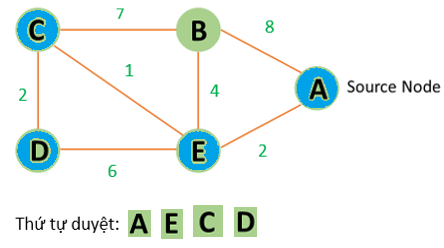
* Ví dụ minh họa:











Dijkstra:

* Ý tưởng chung: Là thuật toán luôn chọn đường đi có chi phí thấp nhất tại mọi thời điểm, khác hoàn toàn so với UCS, UCS là chọn đường đi có chi phí thấp nhất so với node ban đầu trong khi Dijkstra sẽ tìm đường đi có chi phí nhỏ nhất xung quanh node n và chọn đường đi đó và lặp đi lặp lại như vậy ở từng node trong đồ thị, không quan tâm tới node ban đầu.
* Mã giả:

While True:

Nếu open\_set không còn node nào nữa, tức open\_set rỗng nhưng chưa trả về True thì tức đã duyệt qua mọi node và node cần tìm không tồn tại,

Trả về False

Tìm node cuối cùng có chi phí nhỏ nhất

Nếu node tìm được là node cần tìm

Trả vể True

Thêm node vừa tìm được vào closed\_set

Tìm các node kề và tính toán chi phí của nó

Nếu node kề này không tồn tại cả trong closed\_set và open\_set

Thêm node này vào cả hai tập closed\_set và open\_set

Nếu node này tồn tại trong open\_set

So sánh chi phí của node kề này với node hiện tại ,nếu node kề có chi phí nhỏ hơn

Thêm lại node này vào open\_set để xét đường đi khác

Pop node hiện tại khỏi open\_set để xét node tiếp theo

* Đánh giá thuật toán:

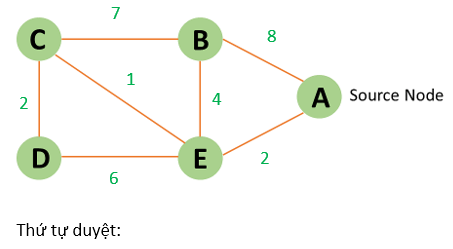
Nếu không gian là hữu hạn thuật toán này có thể giải quyết đầy đủ.

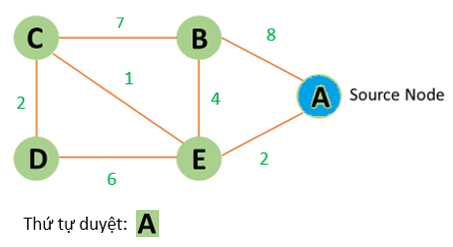
Nếu không gian là hữu hạn thì thuật toán này có thể tìm lời giải tối.

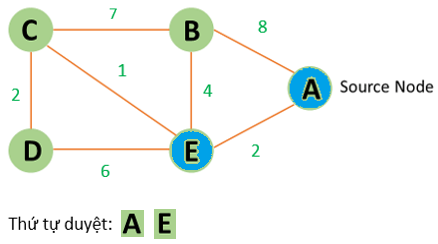
Độ phức tạp thời gian của thuật toán là O(V2) với V là số đỉnh của đồ thị

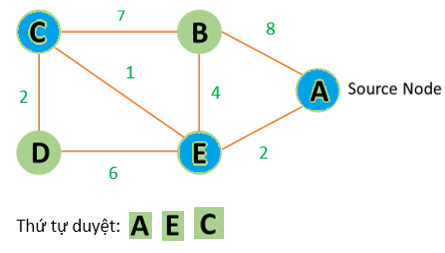
Đọ phức tạp không gian của thuật toán là O(V) với V là số đỉnh của đồ thị

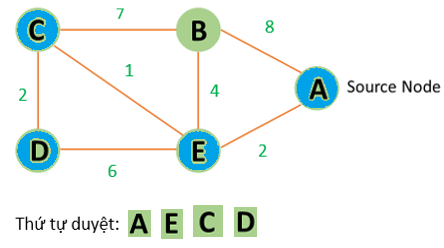
* Ví dụ minh họa:

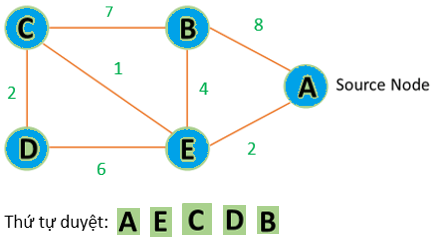












https://towardsdatascience.com/a-small-taste-of-classical-ai-solving-search-problems-aab1a1984a24

<https://www.tutorialandexample.com/problem-solving-in-artificial-intelligence>

https://www.baeldung.com/cs/informed-vs-uninformed-search

<https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm>

<https://medium.com/nerd-for-tech/graph-traversal-in-python-bfs-dfs-dijkstra-a-star-parallel-comparision-dd4132ec323a>

<https://python.plainenglish.io/uniform-cost-search-ucs-algorithm-in-python-ec3ee03fca9f>

https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/