KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐAI HỌC KHOA HỌC TỰ NHIÊN

PROJECT 1: SEARCH
MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO

Giáo viên hướng dẫn: Hoàng Xuân Trường

Lớp: 18_21

Sinh viên thực hiện:

Tên: Nguyễn Công Bình Phương

MSSV: 18120517

2020-2021

MỤC LỤC

A. GIỚI THIỆU ĐỒ ÁN

Đồ án 1 môn Cơ sở trí tuệ nhân tạo
Giáo viên hướng dẫn: Hoàng Xuân Trường

Đồ án về nghiên cứu, cài đặt và trình bày các thuật toán tìm kiếm trên đồ thị. Nhiệm vụ chỉ cần viết code của các hàm BFS, DFS, UCS, AStar

B. BÁO CÁO ĐỒ ÁN

I. MỨC ĐỘ HOÀN THÀNH

- Viết code cho mỗi thuật toán (hoàn thành)

- Biểu diễn tìm kiếm trên giao diện (hoàn thành)

- yêu cầu test_case (hoàn thành)

- thuật toán khác (hoàn thành)

=> Đánh giá chung (99%): 9.5/10

II. NỘI DUNG

1. BFS (BREAD FIRST SEARCH)

a. thuật toán

- Ý tưởng: Bắt đầu ở đỉnh gốc, từ đỉnh gốc đó duyệt hết lần lượt danh sách các đỉnh kề với nó. Sau đó tiếp tục quá trình duyệt qua các đỉnh kề với đỉnh vừa xét cho đến khi đạt kết quả cần tìm.

- Thuật toán dùng hàng đợi để lưu trữ thông tin trung gian trong quá trình tìm kiếm:

+ B1: Cho đỉnh đầu tiên vào hàng đợi
+ B2: Lấy đỉnh đầu tiên ra khỏi hàng đợi và xem xét nó
* Nếu đỉnh đó là đỉnh đích thì thông báo đường đi và kết thúc chương trình

* Nếu không phải thì duyệt tất cả các đỉnh kề với nó mà chưa xét vào hàng đợi

+ B3:

* Nếu hàng đợi rỗng, thông báo kết quả không tìm thấy đường đi

* Nếu hàng đợi không rỗng, thì quay về bước 2

b. đặc tính

- Chi phí không gian: gọi V là tập hợp đỉnh của đồ thì và |V| là số đỉnh thì không gian cần dùng của thuật toán là O(|V|)

- Chi phí thời gian: gọi V tập tập hợp đỉnh, E là tập hợp cạnh. Độ phức tạp thời gian là O(|V| + |E|)

c. chi tiết cài đặt

- Cài đặt không sử dụng trọng số giữa các cạnh. các cài đặt như phần thuật toán ở trên, thứ tự xét các đỉnh tiếp theo sẽ theo thứ tự được trữ trong list. (tuần tự)

- Về code phần thể hiện hình ảnh (đồ thị). Sẽ được cài đặt theo qui trình sau :

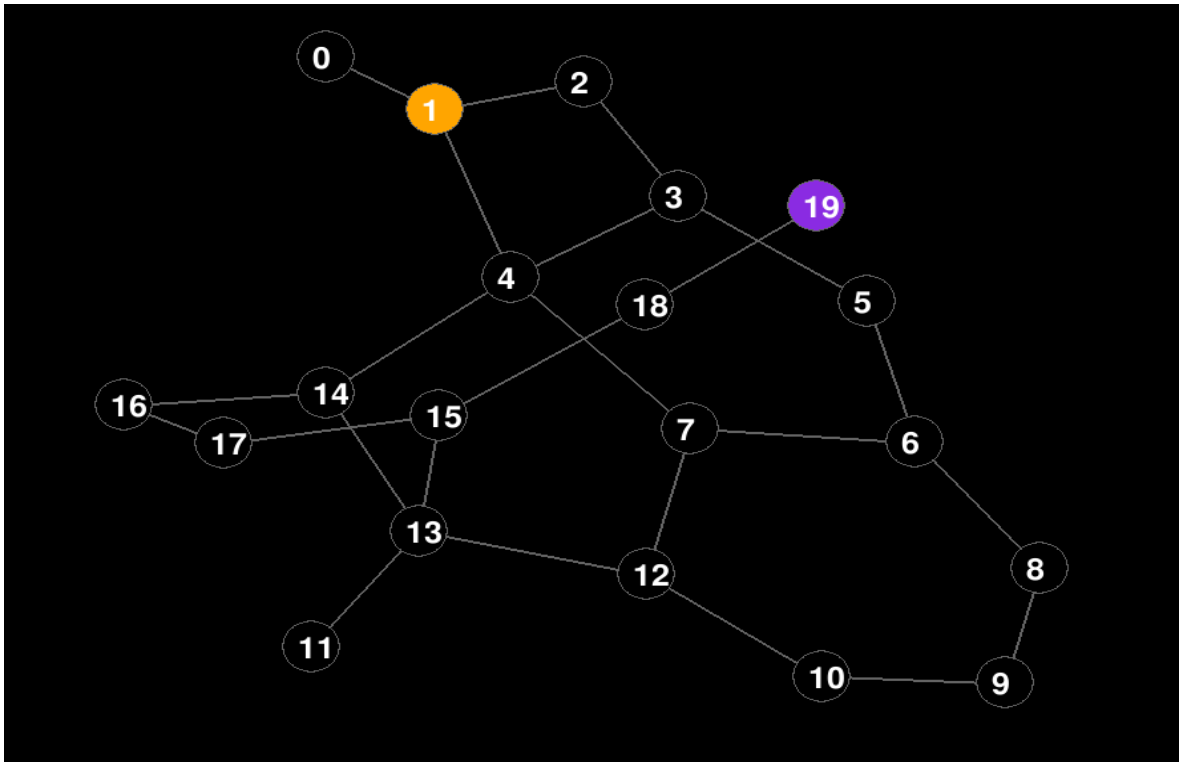
+ Trước khi cho vòng lặp tiềm kiếm, tô màu điểm bắt đầu là màu cam, điểm đích là màu tím

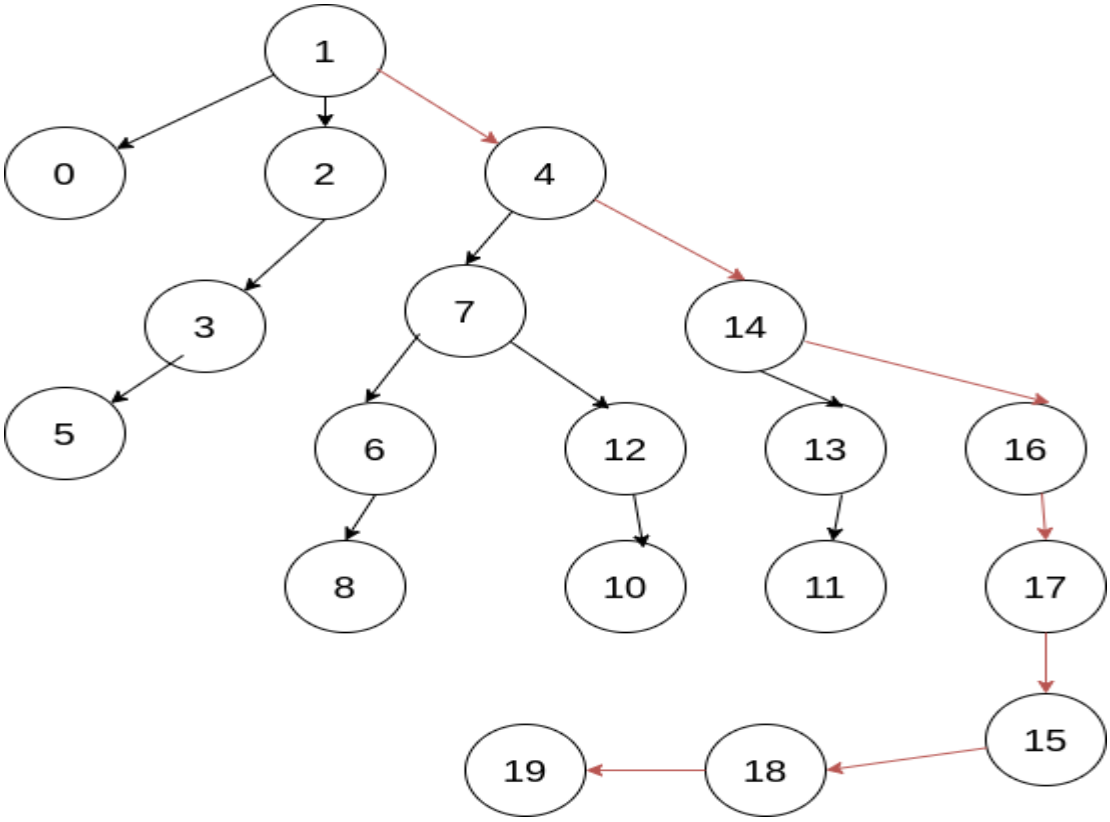
+ Vào vòng lặp, mỗi khi lấy ra một đỉnh (đỉnh hiện tại), tô điểm đó màu xanh dương. Các điểm xét kế tiếp (các điểm kề và "hợp lệ") sẽ được tô màu đỏ và cạnh giữa nó vơí điểm hiện tại sẽ được tô thành màu cam, lúc này update hình ảnh (khi gọi hàm update hình ảnh thì mới hiển thị mọi thứ ta đã tô) rồi sau đó tô điểm hiện tại thành màu vàng (xem như điểm đã xét), và cạnh xét xong thành màu trắng. Đường đi tìm được sẽ tô thành màu xanh lá cây. Nếu không tìm thấy đường đi thì thông báo kết quả ra màn hình terminal(**)

(**) **Lưu ý:** cách xử lí hình ảnh này sẽ được áp dụng tương tự vào các thuật toán tiếp theo, chỉ riêng thuật toán DFS sẽ được xử lí khác (có thêm phần quay lui lại đỉnh trước trong các trường hợp "ngõ cụt").

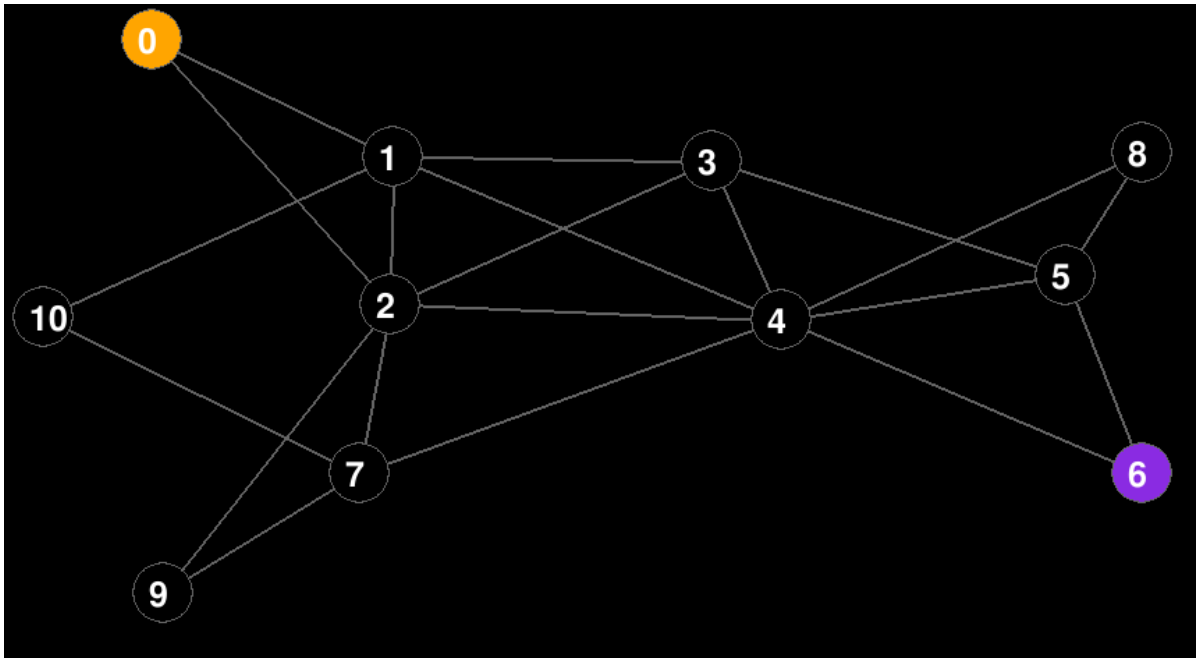
- Ở đây sẽ đưa ra minh họa cây qui trình trong mỗi test case:

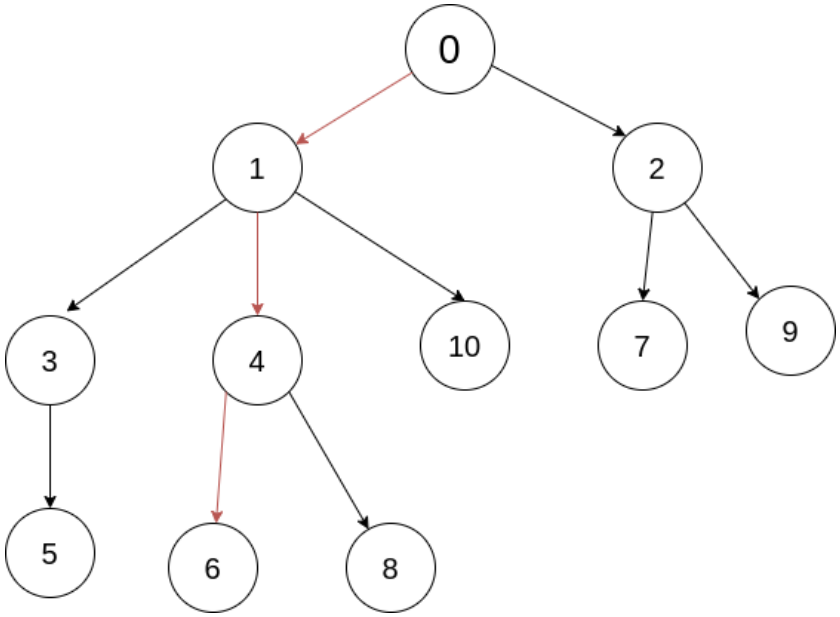
+ Test case 1:


0
1
2
3
19
4
18
5
16
14
15
7
6
17
13
12
8
11
10
9

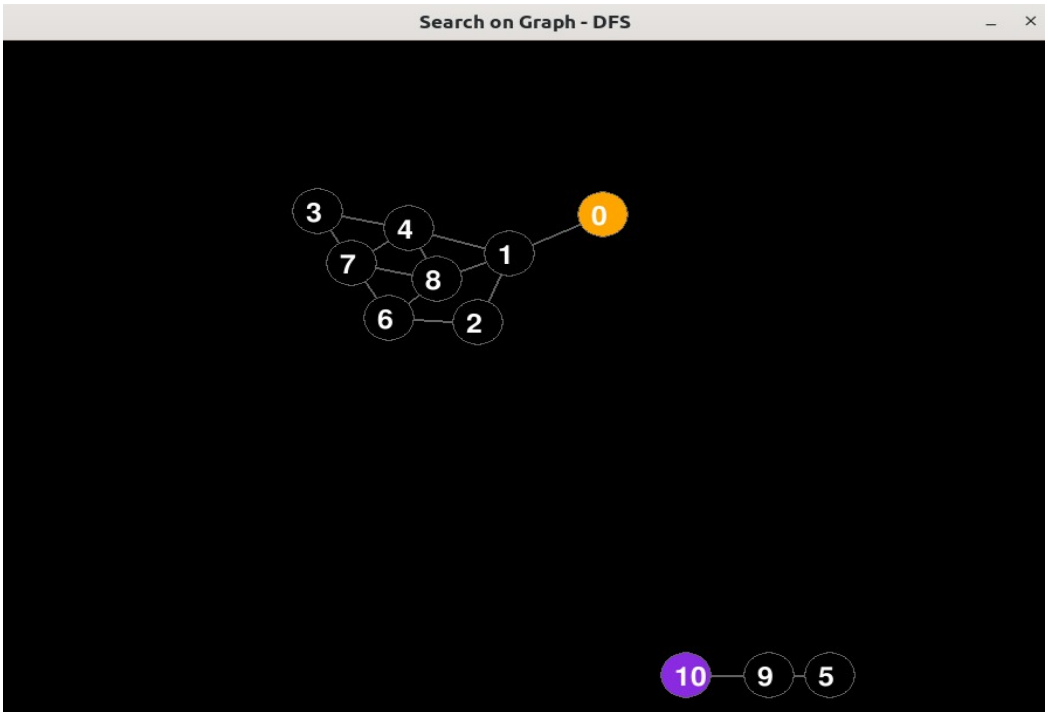


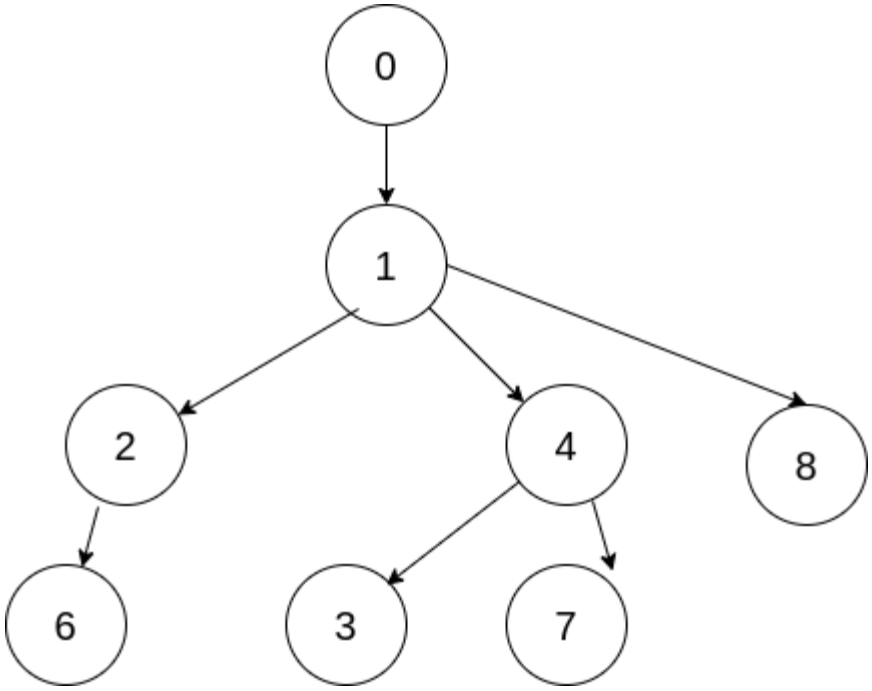
+ Test case 2:





+ Test case 3:





→ không có đường đi

=> **Nhận xét thuật toán:** Thuật toán tìm kiếm chiều rộng luôn tìm ra được đường đi ngắn nhất có thể nếu đồ thì không có chi phí giữa các cạnh. Có điều thuật toán tìm kiếm này quét gần hết mọi khả năng, chi phí cao, thời gian chậm.

2. DFS (DEPTH FIRST SEARCH)

a. thuật toán

- Ý tưởng: Bắt đầu ở đỉnh gốc, từ đỉnh gốc duyệt đến đỉnh tiếp theo kề với nó mà chưa có trong quy trình. Nếu hết đường đi thì quay lại nút trước đó tìm nút khác (phát triển xa nhất trong mỗi nhánh) tới khi tìm thấy đỉnh đích.

- Thuật toán: dùng stack cho quá trình tìm kiếm và danh sách các đỉnh đã duyệt qua:

+ B1: Cho điểm đầu tiên vào stack, tiện thể đánh dấu điểm đầu đã được duyệt, thêm vào đường đi

+ B2: Nếu stack rỗng, trả về thất bại

+ B3: lấy giá trị cuối của stack ra (chỉ lấy giá trị, không lấy nó ra khỏi stack). Kiểm tra xem đã ghé thăm chưa, nếu chưa cho vào đường đi và nếu nó là đích thì thông báo ra màn hình kết quả

+ Bước 4: Xét điểm tiếp theo kề cận với điểm đang xét thỏa mãn nếu chưa ghé thăm cho vào stack. Nếu không có điểm kề cận nào thỏa mãn (ngõ cụt) thì lấy điểm hiện tại ra khỏi stack và bỏ điểm cuối cùng trong đường đi. Quay lại bước 2.

b. đặc tính

- Chi phí không gian: gọi V là tập hợp đỉnh của đồ thì và |V| là số đỉnh thì không gian cần dùng của thuật toán là O(|V|) (trong trường hợp xấu nhất). Xét tổng thể thì sẽ thấp hơn thuật toán BFS mặt dù cũng độ phức tạp O(|V|).

- Chi phí thời gian: gọi V tập tập hợp đỉnh, E là tập hợp cạnh. Độ phức tạp thời gian là O(|V| + |E|) (tương đường với BFS).

c. chi tiết cài đặt

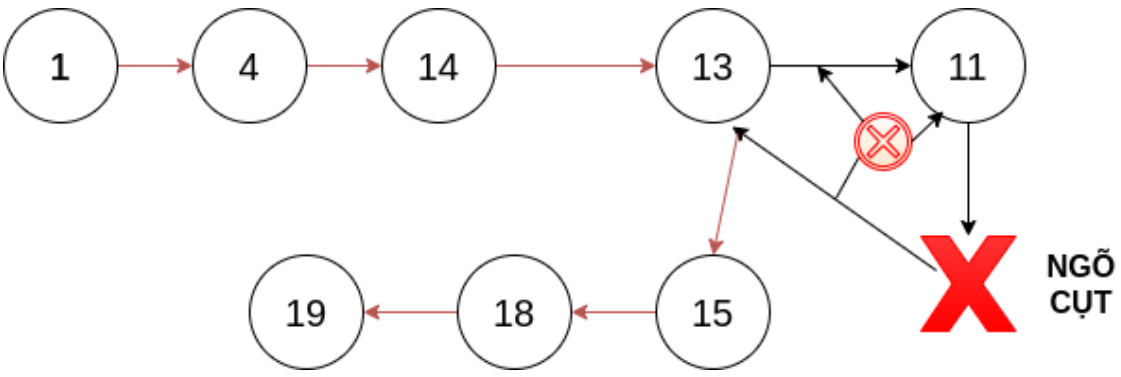
- Thuật toán này cũng sẽ không sử dụng chi phí đi đường đi. Điều muốn chú ý trong thuật toán mình code chính là cách xét các điểm tiếp theo. Giả sử điểm A bao gồm các điểm tiếp theo theo thứ tự la B, C, D. Theo cách xét tiền thư tự thì sẽ từ B đến D, nếu B hợp lệ thì sẽ chọn B là điểm kế tiếp, nhưng làm như vậy thì mỗi lần chạy code sẽ giống nhau. Vì có rất nhiều đường đi nên ở đây sẽ hoàn toàn xét ngẫu nhiên. Để làm được điều này thì danh sách B, C, D sẽ đảo thứ tự ngẫu nhiên (có thể là C, D, B hay D,B,C, …). Trong code sẽ reorder lại danh sách đỉnh bằng method **sample()** của thư viện **random**.

- Về code thể hiện hình ảnh: Về phần xét điểm hiện tại, cạnh, các điểm tiếp theo, cạnh đã xét và các điểm đã duyệt qua thì trong thuật toán này sẽ thêm thể hiện về mặt hình ảnh khi bị "ngõ cụt" và phải quay lại. Mỗi khi đỉnh đang xét bị "ngõ cụt" thì điểm đó sẽ đổi màu lại như mặc định (màu bạc) và cạnh giữa nó với điểm trước đó cũng về mặc định. Và điểm trước đó thành điểm đang xét (màu xanh). (* *Xem video demo để hiểu rõ hơn)*

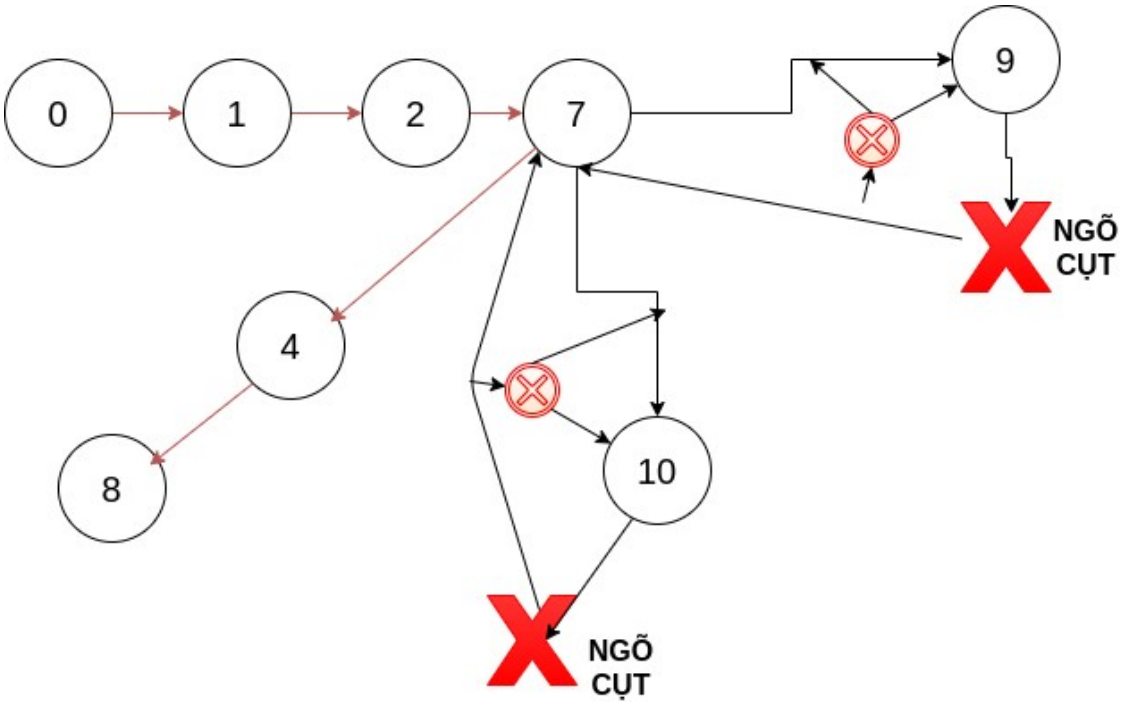
- Minh họa cho mỗi testcase: Vì thuật toán này mỗi lần chạy sẽ cho ra mỗi đường đi khác nhau nên sẽ minh họa một đường đi cụ thể

Lưu ý: các thuật toán đều dùng chung testcase nên hình ảnh đồ thì các testcase đã có ở phía trên. Phần này chỉ thể hiện đường đi của thuật toán thôi. Tương tự cho các thuật toán sau.

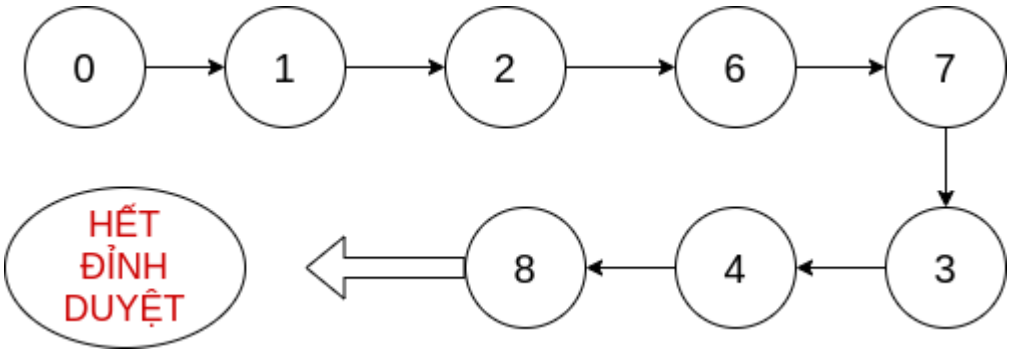
+ Testcase 1:



+ Testcase 2:



+ Testcase 3:



3. UCS (UNIFORM-C-SEARCH)

a. thuật toán

- Ý tưởng: uniform cost search (tìm kiếm chi phí đều) dành cho bài toán có chi phí đường đi. Được xem như là biến thể của thuật toán Djikstra. Từ đỉnh gốc sẽ mở rộng ra mọi đường đi, lưu các đường đi và sắp xếp tăng dần theo chi phí. Cứ thể mở rộng và thay thế những đường đi có trước dựa vào đường đi có chi phí thấp nhất tính từ đỉnh gốc cho đến khi có kết quả hoặc duyệt hết đồ thị.

- Thuật toán: Dùng **hàng đợi ưu tiên** để lưu trữ, lí do dùng hàng đợi ưu tiên bởi vì hàng đợi ưu tiên tự sắp xếp thứ tự các thành phần từ nhỏ tới lớn, và khi lấy ra thì sẽ lấy ra giá trị nhỏ nhất => Rất hợp lí với thuật toán của thuật toán này :

+ B1: Khởi tạo các giá trị **frontier** = [**start**], **explored** = (), **G**[**start**] = 0

+ B2: Nếu **frontier** rỗng thì trả về thất bại, không tìm được đường đi

+ B3: Lấy ra đỉnh **current** của **frontier** sao cho **G**[**current**] đạt min.

* Nếu **current** là điểm đích thì dừng chương trình thông báo kết quả

* Thêm **current** vào **explored**

+ B4: Xét các điểm **q** kề cận với **current**

***** Nếu **q** không có trong **explored** và không có trong **frontier** thì thêm **q** vào frontier và cập nhật **G**[q]

* Nếu **q** không có trong **explored** và có trong **frontier** và cập nhật lại **G**[**q**] nếu giá trị **G**[**q**] nhỏ hơn

+B5: quay lại bước 2

b. đặc tính

- Chi phí thời gian: gọi **C** là chi phí cho giải pháp tối ưu, và **z** là mỗi bước để tiến gần hơn đến nút mục tiêu, **b** là hệ số phân nhánh. Khi đó số bước là **C**/**z** + 1. Do đó độ phức tạp về mặt thời gian trong trường hợp xấu nhất là **O(b^(1+C/z))**.

- Chi phí về mặt không gian: **O(b^(1+C/z))**

c. chi tiết cài đặt

- Thuật toán này sử dụng chi phí đường đi giữa 2 điểm. Cách tính chi phí này dựa vào hình vẽ của đồ thì trên màn hình, chi phí chính là khoảng cách giữa 2 đỉnh trên đồ thị (Sử dụng công thức tính khoảng cách dựa vào tọa độ).

- Sau đây là cách thuật toán hoạt động theo các test case:

  * Lưu ý: Xem lại hình vẽ testcase 1 ở phía trên mục BFS

+ Test case 1:

* bảng chi phí giữa các đỉnh(đã làm tròn):

Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí
(0, 1)	93	(7, 12)	126.4	(16, 17)	81.5
(1, 2)	114.3	(8, 9)	99.5	(18, 19)	153.4
(1, 4)	152	(9, 10)	138.1	(6, 8)	141.6
(2, 3)	119.4	(10, 12)	157.5	(15, 18)	180.76
(3, 4)	143.1	(11, 13)	126.4	(6, 7)	169,3
(3, 5)	167	(12, 13)	174.7	(15, 17)	163.6
(4, 7)	185.3	(13, 14)	135.5	(5, 6)	123.3

(4, 14)	169,5	(13, 15)	98.1	(14, 16)	152.3

* Quá trình: Tổng hợp các lần duyệt của các giá trị **current**, **frontier** và **explored**

current: 1 **frontier**{(0: 93.1),(2: 114.3),(4: 152.1),} **explored**{1}
- current: 0 **frontier**{(2: 114.3),(4: 152.1),} **explored**{0, 1}
- current: 2 **frontier**{(4: 152.1),(3: 233.7),} **explored**{0, 1, 2}
- current: 4 **frontier**{(3: 233.7),(7: 337.4),(14: 321.6),} **explored**{0, 1, 2, 4}
- current: 3 **frontier**{(7: 337.4),(14: 321.6),(5: 400.8),} **explored**{0, 1, 2, 3, 4}-
- current: 14 **frontier**{(7: 337.4),(5: 400.8),(13: 457.1),(16: 473.9),}
explored{0, 1, 2, 3, 4, 14}
- current: 7 **frontier**{(5: 400.8),(13: 457.1),(16: 473.9),(6: 506.8),(12: 463.8),}
explored{0, 1, 2, 3, 4, 7, 14}
- current: 5 **frontier**{(13: 457.1),(16: 473.9),(6: 506.8),(12: 463.8),}
explored{0, 1, 2, 3, 4, 5, 7, 14}
- current: 13 **frontier**{(16: 473.9),(6: 506.8),(12: 463.8),(11: 583.4),(15: 555.2),}
explored{0, 1, 2, 3, 4, 5, 7, 13, 14}
- current: 12 {**frontier**(16: 473.9),(6: 506.8),(11: 583.4),(15: 555.2),(10: 621.4),}
explored{0, 1, 2, 3, 4, 5, 7, 12, 13, 14}
- current: 16 **frontier**{(6: 506.8),(11: 583.4),(15: 555.2),(10: 621.4),(17: 555.5),}
explored{0, 1, 2, 3, 4, 5, 7, 12, 13, 14, 16}
- current: 6 **frontier**{(11: 583.4),(15: 555.2),(10: 621.4),(17: 555.5),(8: 648.5),}
explored{0, 1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 16}
- current: 15 **frontier**{(11: 583.4),(10: 621.4),(17: 555.5),(8: 648.5),(18: 736.0),}
explored{0, 1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16}
- current: 17 **frontier**{(11: 583.4),(10: 621.4),(8: 648.5),(18: 736.0),} **explored** {0, 1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15, 16, 17}
- current: 11 **frontier**{(10: 621.4),(8: 648.5),(18: 736.0),} **explored**{0, 1, 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17}
- current: 10 **frontier**{(8: 648.5),(18: 736.0),(9: 759.5),} **explored**{0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17}
- current: 8 **frontier**{(18: 736.0),(9: 747.9),} **explored**{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17}
- current: 18 **frontier**{(9: 747.9),(19: 889.4),} **explored**{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18}
- current: 9 **frontier**{(19: 889.4),} **explored**{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
→ đường đi tìm được: [19, 18, 15, 13, 14, 4, 1]

+ Test case 2:

* Bảng chi phí giữa các đỉnh

Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí
0 1	192.3	0 2	255.6	1 2	106
1 3	229	1 4	302.5	1 10	276.1
2 3	253	2 4	281.2	2 7	123
2 9	263.5	3 4	124.5	3 5	266.9
4 5	206.5	4 6	281.4	4 7	322.3
4 8	285.4	5 6	152.3	5 8	103.7
7 9	165.1	7 10	253.1		

* Quá trình chạy thuật toán

- **current**: 0 **frontier**:{(1: 192.3),(2: 255.6),} **explored**: {0}
- **current**: 1 **frontier**:{(2: 255.6),(3: 421.3),(4: 494.9),(10: 468.4),} **explored**{0, 1}
- **current**: 2 **frontier**:{(3: 421.3),(4: 494.9),(10: 468.4),(7: 378.6),(9: 519.1),} **explored**{0, 1, 2}
- **current**: 7 **frontier**:{(3: 421.3),(4: 494.9),(10: 468.4),(9: 519.1),} **explored**{0, 1, 2, 7}
- **current**: 3 **frontier**:{(4: 494.9),(10: 468.4),(9: 519.1),(5: 688.2),} **explored**{0, 1, 2, 3, 7}
- **current**: 10 **frontier**:{(4: 494.9),(9: 519.1),(5: 688.2),} **explored**{0, 1, 2, 3, 7, 10}
- **current**: 4 **frontier**:{(9: 519.1),(5: 688.2),(6: 776.2),(8: 780.3),} **explored**{0, 1, 2, 3, 4, 7, 10}
- **current**: 9 **frontier**:{(5: 688.2),(6: 776.2),(8: 780.3),} **explored**{0, 1, 2, 3, 4, 7, 9, 10}
- **current**: 5 **frontier**:{(6: 776.2),(8: 780.3),} **explored** {0, 1, 2, 3, 4, 5, 7, 9, 10}

→ đường đi tìm được: [6, 4, 1, 0]

+ Test case 3:

* Bảng chi phí giữa các đỉnh

Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí	Cặp đỉnh	Chi phí
(1, 0)	88.1	(7, 3)	56.9	(8, 6)	55.2
(2, 1)	70.4	(7, 4)	59.1	(8, 7)	74.5
(4, 1)	89.3	(7, 6)	61.1	(9, 5)	52.0

(4, 3)	79.6	(8, 1)	66.5	(10, 9)	71.0
(6, 2)	76.1	(8, 4)	53.7		

* Quá trình chạy thuật toán

- current: 0 **frontier**{(1: 88.1),} **explored**{0}

- current: 1 **frontier**{(2: 158.5),(4: 177.4),(8: 154.6),} **explored**{0, 1}

- current: 8 **frontier**{(2: 158.5),(4: 177.4),(6: 209.9),(7: 229.2),} - **explored**{0, 1, 8}

- current: 2 **frontier**{(4: 177.4),(6: 209.9),(7: 229.2),} **explored**{0, 1, 2, 8}

- current: 4 **frontier**{(6: 209.9),(7: 229.2),(3: 257.1),} **explored**{0, 1, 2, 4, 8}

- current: 6 **frontier**{(7: 229.2),(3: 257.1),} **explored**{0, 1, 2, 4, 6, 8}

- current: 7 **frontier**{(3: 257.1),} **explored**{0, 1, 2, 4, 6, 7, 8}

- current: 3 **frontier**{} **explored**{0, 1, 2, 3, 4, 6, 7, 8}

→ không tim được đường đi

4. A*(A-STAR)

a. thuật toán

- Ý tưởng: Thuật toán tìm kiếm A-star dùng cho đồ thị có chi phí. Dùng đánh giá heuristic để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút

- Hàm heuristic trong bài này: Vì trong bài này không cho trước chi phí heuristic. Để trực quan về mặt hình ảnh được thể hiện trong đồ thị. Hàm đánh giá heuristic chính là "đường chim bay" giữa hai đỉnh. Hay nói đúng hơn chính là sử dụng công thức toán tính khoảng cách giữa hai điểm, giống với hàm chi phí nhưng khác ở chỗ chính là 2 điểm này là 2 điểm bất kì.

- Thuật toán:

Các biến:

1. **Openset**: tập các trạng thái được sinh ra nhưng chưa được xét
2. **Closedset**: tập các trạng thái đã được xét đến
3. g(p): khoảng cách từ trạng thái ban đầu đến trạng thái hiện tại
4. h(p): hàm heuristic ước lượng khoảng cách từ điểm hiện tại đến đích
5. f(p) = g(p) + h(p)

+ Bước 1: Khởi tạo **Openset** = {start}, **Closedset** = {}

+ Bước 2: Nếu openset rỗng thông báo không có đường đi và kết thúc chương trình

+ Bước 3: Lấy ra trạng thái đỉnh tốt nhất là p trong tập **openset**. trạng thaí đỉnh tốt nhất chính là đỉnh có hàm **f(p)** đạt giá trị nhỏ nhất. Thêm **p** vào trong **closedSet**. Sau đó duyệt các đỉnh kề **a** với đỉnh của p:

* Nếu **a** có trong **closedset** thì sang bước 4

* Nếu **a** đã có trong **openset** thì tính: **_g(a) = g(p) + cost(p,a), nếu _g(a) < g(a)** thì cập nhật lại giá trị *g(a) = _g(a)*

* Nếu **a** chưa có trong **openset** thì tính g(a) và thêm a vào **openset**

+ Bước 4: Quay lại bước 2

b. đặc tính

- Chi phí thời gian: Tùy thuộc vào hàm heuristic. Trong trường hợp xấu nhất, số nút được mở rộng theo hàm mũ của độ dài lời giải, nhưng nó sẽ à hàm đa thức khi hàm heuristic h thỏa mãn điều kiện sau:

|h(x) – h*(x)| <= O(log(h*(x)))

với h*(x) là hàm heuristic tối

- Chi phí không gian: Tùy vào hàm heuristic, O(b^m) với m là độ sâu của cây cần duyệt và mỗi trạng thái khi được phát triển sẽ sinh ra b trạng thái.

c. chi tiết cài đặt

- Testcase 1:

+ Bảng đánh giá heuristic:

đỉnh	h_value	đỉnh	h_value	đỉnh	h_value	đỉnh	h_value
0	389.6	5	88.6	10	395	15	334.1
1	298.2	6	211.4	11	530	16	547.1
2	203.6	7	209.7	12	334.5	17	488.4
3	104.3	8	347.3	13	404.9	18	153.4
4	237.7	9	424.5	14	401	19	0

+ Các giá trị của đỉnh đang xét, **openset**, **closed** qua các lần lặp:

1 {0, 2, 4} {1}

---

2 {0, 3, 4} {1, 2}

---

3 {0, 4, 5} {1, 2, 3}

---

5 {0, 4, 6} {1, 2, 3, 5}

---

6 {0, 4, 7, 8} {1, 2, 3, 5, 6}

----
7 {0, 4, 8, 12} {1, 2, 3, 5, 6, 7}
----
4 {0, 8, 12, 14} {1, 2, 3, 4, 5, 6, 7}
----
12 {0, 8, 10, 13, 14} {1, 2, 3, 4, 5, 6, 7, 12}
----
0 {8, 10, 13, 14} {0, 1, 2, 3, 4, 5, 6, 7, 12}
----
8 {9, 10, 13, 14} {0, 1, 2, 3, 4, 5, 6, 7, 8, 12}
----
9 {10, 13, 14} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 12}
----
10 {13, 14} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
----
14 {13, 16} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14}
----
13 {11, 15, 16} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14}
----
15 {11, 16, 17, 18} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15}
----
18 {11, 16, 17, 19} {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 18}
----
19 → Trả về kết quả đường đi $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 12 \rightarrow 13 \rightarrow 15 \rightarrow 18 \rightarrow 19$

- Testcase 2:
+ Bảng đánh giá heuristic

đỉnh	h_value	đỉnh	h_value	đỉnh	h_value	đỉnh	h_value
0	776	3	381.7	7	562.0	6	0
1	583.9	4	281.4	8	230.0	10	796.9
2	553.4	5	152.3	9	708.2		

+ Các giá trị của **đỉnh đang xét**, **openset**, **closed** qua các lần lặp:

0 {1, 2} {0}
----
1 {2, 3, 4, 10} {0, 1}
----

4 {2, 3, 5, 6, 7, 8, 10} {0, 1, 4}

---

6 → trả về kết quả đường đi 0->1->4->6

- Testcase 3:

* Bảng giá trị heuristic

Đỉnh	0	1	2	3	4	5	6	7	8	9	10
h_value	441.7	426.6	378.5	540.3	484.9	123.0	422.3	483.0	431.2	71.0	0.0

* giá trị của **đỉnh đang xét, openset**, **closed** qua các lần lặp:

0 {1} {0}
1 {8, 2, 4} {0, 1}
2 {4, 6, 8} {0, 1, 2}
8 {4, 6, 7} {0, 1, 2, 8}
6 {4, 7} {0, 1, 2, 6, 8}
7 {3, 4} {0, 1, 2, 6, 7, 8}
4 {3} {0, 1, 2, 4, 6, 7, 8}
3 {} {0, 1, 2, 3, 4, 6, 7, 8}
→ không tìm thấy đường đi

*** SO SÁNH SỰ KHÁC BIỆT GIỮA UNIFORM COST SEARCH VÀ BEST FIRST SEARCH**

Cả hai thuật toán đều nằm trong bài toán tìm đường đi có chi phí và mở rộng dựa theo chi phí tốt nhất.

Thuật toán Uniform cost search tìm kiếm đường đi kiểu "mù quáng" nghĩa là mở rộng đường đi theo đường ngắn nhất tính đến điểm đầu (không dựa theo đích đến). Còn thuật toán A thì không, dùng hàm heuristic để ước lượng đường đi tới đích và chọn chi phí nhỏ nhất theo tổng khoảng cách từ điểm xét đến đỉnh đầu và đến đích. Cả hai thuật toán đều có danh sách mở rộng nhưng thuật toán A* sẽ giảm thiểu số lượng trong danh sách mở rộng đó.*

=> Tóm lại:

- giống giau: đều dùng hàng đợi ưu tiên, duyệt chiều rộng

- khác nhau:

+ UCS: tìm kiếm mù, không sử dụng hàm đánh giá, chi phí tính từ đỉnh đầu đến đỉnh hiện tại

+ A: tìm kiếm kinh nghiệm, sử dụng hàm đánh giá, chi phí là tổng khoảng cách từ đỉnh đầu đến đỉnh hiện tại và từ đỉnh hiện tại đến đích.*

5. ALGORITHM BONUS - BeFS (BEST FIRST SEARCH)

a. thuật toán

- Ý tưởng: Thuật toán Best First Search là thuật toán tìm kiếm theo bề rộng được hướng dẫn bởi hàm đánh giá heuristic. Tư tưởng của thuật toán này là việc tìm kiếm bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo có giá trị của hàm đánh giá là thấp nhất so với các nút còn lại nằm trong hàng đợi.

- Hàm heuristic trong thuật toán này chính là hàm heuristic đã sử dụng ở thuật toán A*

- Thuật toán:

Các biến:

1. **Openset**: tập các trạng thái được sinh ra nhưng chưa được xét
2. **Closedset**: tập các trạng thái đã được xét đến
3. h(a): hàm heuristic đánh giá tại đỉnh a

Bước 1: Khởi tạo **Openset** = (**start**), **closedset** = ()

Bước 2: Nếu **Openset** rỗng, thông báo không tìm được đường đi và kết thúc chương trình

Bước 3: Lấy giá trị ưu tiên p ra khỏi Openset (là giá trị có h(p) nhỏ nhất).

* Nếu p là đích thì thông báo kết quả và kết thúc chương trình

* Thêm **p** vào tập **closedset**

* Thêm các đỉnh kề cận **q** của **p** mà không thuộc **Closedset** và cả **Openset** vào Openset

Bước 4: quay lại bước 2

b. đặc tính

- Chi phí thời gian: O(nlog(n)) n là số
- Chi phí không gian:

c. chi tiết cài đặt

các giá trị **current**, **openset** *và* ***closedset*** *trong các lần duyệt*

- Testcase 1:

1	{0, 2, 4}	{1}
2	{0, 3, 4}	{1, 2}
3	{0, 4, 5}	{1, 2, 3}
5	{0, 4, 6}	{1, 2, 3, 5}
6	{0, 4, 7, 8}	{1, 2, 3, 5, 6}
7	{0, 4, 8, 12}	{1, 2, 3, 5, 6, 7}
4	{0, 8, 12, 14}	{1, 2, 3, 4, 5, 6, 7}
12	{0, 8, 10, 13, 14}	{1, 2, 3, 4, 5, 6, 7, 12}
8	{0, 9, 10, 13, 14}	{1, 2, 3, 4, 5, 6, 7, 8, 12}
0	{9, 10, 13, 14}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 12}
10	{9, 13, 14}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12}
14	{9, 13, 16}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14}
13	{9, 11, 15, 16}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14}
15	{9, 11, 16, 17, 18}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15}
18	{9, 11, 16, 17, 19}	{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 18}

19 → Trả về kết quả đường đi: 0->1->2->3->5->6->7->4->14->13->15->18->19
- Testcase 2:
0 {1, 2} {0}
2 {1, 3, 4, 7, 9} {0, 2}
4 {1, 3, 5, 6, 7, 8, 9} {0, 2, 4}
6 → Trả về kết quả đường đi: 0->2->4->6
- Testcase 3:
0 {1} {0}
1 {8, 2, 4} {0, 1}
2 {4, 6, 8} {0, 1, 2}
6 {4, 7, 8} {0, 1, 2, 6}
8 {4, 7} {0, 1, 2, 6, 8}
7 {3, 4} {0, 1, 2, 6, 7, 8}
4 {3} {0, 1, 2, 4, 6, 7, 8}
3 {} {0, 1, 2, 3, 4, 6, 7, 8}
→ không tìm thấy đường đi

TÀI LIỆU THAM KHẢO

1. https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng
2. https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_s%C3%A2u
3. https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_chi_ph%C3%AD_%C4%91%E1%BB%81u
4. https://en.wikipedia.org/wiki/A*_search_algorithm
5. 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
6. https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G