TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP. HCM

KHOA CÔNG NGHỆ THÔNG TIN

CƠ SỞ TRÍ TUỆ NHÂN TẠO

BÁO CÁO LAB1: SEARCH

SINH VIÊN THỰC HIỆN:

1712919 Lê Văn Vũ

GV LÝ THUYẾT/ HD THỰC HÀNH:

Thầy **Lê Hoài Bắc**

Thầy **Hoàng Xuân Trường**

MỤC LỤC

A-MỨC ĐỘ HOÀN THÀNH

STT	Thuật toán	Múc độ hoàn thành (%)		
		test1.txt	test2.txt	test3.txt
1	BFS	100	95	95
2	DFS	100	100	100
3	UCS	100	95	95
4	A*	50		
Điểm tự đánh giá toàn bộ:		**8.0/10.0**		

B-CƠ SỞ LÝ THUYẾT:

I. THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG BFS:

1. Kiến thức cơ bản về thuật toán BFS.

+***Breadth First Search*** là một thuật toán duyệt hoặc tìm kiếm một phần tử trên một cấu trúc dữ liệu dạng cây hay một đồ thị.

+ Nó khác với DFS đó là nó sẽ ưu tiên theo chiều ngang, nghĩa là duyệt từ trái qua phải hết rồi mới duyệt tiếp xuống dưới cho từng phần tử

Để giải quyết thuật toán trên, ta sẽ giải 3 bài toán cơ bản sau:

a, Tìm số thành phần liên thông của đồ thị:

Ý tưởng: Ta sẽ dùng BFS duyệt toàn bộ đồ thị từ 1 đỉnh u cho trước. sử dụng một biến đếm để đếm số thành phần liên thông của đồ thị. Sau khi giải quyết xong vấn đề đếm số thành phần liên thông ta có thể giải quyết các bài toán tiếp sau nó như:

+ Chứng minh đồ thị này là đồ thị liên thông yếu hoặc liên thông mạnh

b, Kiểm tra cạnh cầu, đếm số cạnh cầu của đồ thị

Ý tưởng : Trước hết ta duyệt BFS 1 lần đồ thị để tìm ra số thành phần liên thông ban đầu.Ta xóa lần lượt các cạnh của đồ thị, sau đó dùng BFS duyệt tìm số thành phần liên thông của đồ thị. Nếu số thành phần liên thông tăng chứng tỏ cạnh bị xóa là cạnh cầu.

c, Kiểm tra đỉnh trụ, đếm số đỉnh trụ của đồ thị:

(Đỉnh trụ của đồ thị là đỉnh mà khi xóa nó đi thì số thành phần liên thông của đồ thị tăng lên)

Ý tưởng: Trước hết ta duyệt bfs 1 lần đồ thị để tìm ra số thành phần liên thông ban đầu . Mỗi lần duyệt ta xóa 1 đỉnh của đồ thị. Dùng BFS duyệt tìm số thành phần liên thông của đồ thị. Nếu số thành phần liên thông của đồ thị tăng lên thì đỉnh bị xóa là đỉnh trụ.

2. Các tính chất thuật toán breath first search.

– Sử dụng cấu trúc dữ liệu hàng đợi để lưu trữ node.

– Là cấu trúc dữ liệu dạng đồ thị, hay dạng cây.

– Độ phức tạp thời gian là: O(|V| + |E|) V là số đỉnh duyệt qua và E là số cạnh duyệt qua.

– Độ phức tạp không gian là O(|V|). Số đỉnh là số phần tử cần lưu trữ vào bộ nhớ.

II. THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU DFS:

1. Kiến thức cơ bản:

Thuật toán Depth First Search (DFS – Tìm kiếm theo chiều sâu) là một dạng thuật toán duyệt hoặc tìm kiếm trên cây hoặc đồ thị. Trong lý thuyết khoa học máy tính, thuật toán DFS nằm trong chiến lược tìm kiếm mù (tìm kiếm không có định hướng, không chú ý đến thông tin, giá trị được duyệt) được ứng dụng để duyệt hoặc tìm kiếm trên đồ thị.

a, Ý tưởng thuật toán

Từ đỉnh (nút) gốc ban đầu.

- Duyệt đi xa nhất theo từng nhánh.
  - Khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo.
- Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.

b, Thuật giải

Một số quy ước:

Open: *là tập hợp các đỉnh chờ được xét ở bước tiếp theo theo ngăn xếp (ngăn xếp: dãy các phần tử mà khi thêm phần tử vào sẽ thêm vào đầu dãy, còn khi lấy phần tử ra sẽ lấy ở phần tử đứng đầu dãy).*

Close: *là tập hợp các đỉnh đã xét, đã duyệt qua.*

s: *là đỉnh xuất phát, đỉnh gốc ban đầu trong quá trình tìm kiếm.*

g: *đỉnh đích cần tìm.*

p: *đỉnh đang xét, đang duyệt.*

Trình bày thuật giải:

1. Tập Open chứa đỉnh gốc s chờ được xét.
2. Kiểm tra tập Open có rỗng không.

- Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét p.
  - Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm.
- Nếu tập Open rỗng, tiến đến bước 4.

3. Đưa đỉnh p vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét.
   - Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open. Quay lại bước 2.

4. Kết luận không tìm ra đỉnh đích cần tìm.

2. Độ phức tạp:
Độ phức tạp thời gian: O(|V|+|E|) với |V| là số đỉnh của đồ thị, |E| là số cạnh

III. THUẬT TOÁN TÌM KIẾM DFS:

1. Ý tưởng thuật toán:
Định nghĩa: Hàng đợi ưu tiên PQ là cấu trúc dữ liệu lưu trữ các phần tử cùng với độ ưu tiên của nó và khi lấy phần tử ra khỏi hàng đợi sẽ căn cứ vào độ ưu tiên nhỏ nhất.

Cho một trạng thái n, ký hiệu g(n) là tổng chi phí đường đi ngắn nhất (hiện có) từ trạng thái ban đầu S đến trạng thái n. Thuật toán UCS sử dụng một hàng đợi ưu tiên (Priority Queue – PQ) để lưu trữ và duyệt các trạng thái trên đường đi. Thuật toán dùng thêm một danh sách CLOSE để lưu trữ các trạng thái đã được xét.

Mã giả:

Khởi tạo: PQ rỗng, CLOSE rỗng.
Đưa trạng thái ban đầu START vào PQ, độ ưu tiên g(START) = 0
Lặp đến khi PQ rỗng
  Lấy một trạng thái n (có g thấp nhất) ra khỏi PQ. Đưa n vào CLOSE.

  Nếu n là trạng thái đích GOAL thì "đã tìm thấy". Dừng thuật toán.

  Nếu không, với mỗi trạng thái con n' chưa xét (n' không thuộc CLOSE) của n:

    Tính độ ưu tiên: g(n') = g(n) + cost(n, n')

    đưa (n', g(n')) vào PQ
Cuối lặp
Thông báo không có đường đi từ START đến GOAL.

2. Độ phức tạp thời gian:
Nếu hệ số phân nhánh là b, mỗi khi ta mở rộng một nút thì gặp thêm k nút. Do đó, có
- ✓ 1 nút ở cấp 0,
- ✓ b nút ở cấp 1,
- ✓ b_2 nút ở cấp độ 2,
- ✓ b_3 nút ở cấp độ 3,
- ✓ ...
- ✓ b_k nút ở mức k.

Vì vậy, giả sử rằng tìm kiếm dừng lại sau khi đạt đến cấp độ k. Khi điều này xảy ra, tổng số nút đã

truy cập sẽ là: $1 + b + b_2 + ... + b_k = (b_k + 1 - 1) / (b - 1)$ (*)

(*) xảy ra khi $b_k + 1 / (b - 1) = O(b_k)$, vì vậy nếu nút mục tiêu nằm trong lớp k, thì phải mở rộng tổng số nút $O(b_k)$ để tìm nút mong muốn.

Nếu C là chi phí đích và mỗi bước giúp ta tiến gần hơn đến mục tiêu (goal), thì số bước cần thực hiện là $C / \varepsilon + 1$. Lý do cho +1 là bắt đầu ở khoảng cách 0 và kết thúc ở C / ε, vì vậy thực hiện các bước ở khoảng cách xa: $0, \varepsilon, 2\varepsilon, 3\varepsilon, ..., (C / \varepsilon) \varepsilon$

Và có $1 + C / \varepsilon$ tổng số bước ➔ có $1 + C / \varepsilon$ lớp
➔ Tổng số trạng thái bạn cần mở rộng là $O(b (1 + C / \varepsilon))$.

IV. THUẬT GIẢI A*:

Heuristic và hàm Heuristic là gì?

Heuristic là phương pháp giải quyết vấn đề dựa trên phỏng đoán, ước chừng, kinh nghiệm, trực giác để tìm ra giải pháp gần như là tốt nhất và nhanh chóng.

Hàm Heuristic là hàm ứng với mỗi trạng thái hay mỗi sự lựa chọn một giá trị ý nghĩa đối với vấn đề dựa vào giá trị hàm này ta lựa chọn hành động.

A* là gì?

A* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Từ trạng thái hiện tại A* xây dựng tất cả các đường đi có thể đi dùng hàm ước lược khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

A* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.

Thứ tự ưu tiên cho một đường đi đươc quyết định bởi hàm Heuristic được đánh giá $f(x) = g(x) + h(x)$

- g(x) là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.
- h(x) là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.

Thuật giải A*

1. Open: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
2. Close: tập các trạng thái đã được xét đến.
3. Cost(p, q): là khoảng cách giữa p, q.
4. g(p): khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.
5. h(p): giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
6. f(p) = g(p) + h(p)

    - Bước 1:
        - Open: = {s}
        - Close: = {}
    - Bước 2: while (Open !={})

- Chọn trạng thái (đỉnh) tốt nhất p trong Open (xóa p khỏi Open).
- Nếu p là trạng thái kết thúc thì thoát.
- Chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.
    - Nếu q đã có trong Open
        - Nếu g(q) > g(p) + Cost(p, q)
            - g(q) = g(p) + Cost(p, q)
            - f(q) = g(q) + h(q)
            - prev(q) = p (đỉnh cha của q là p)
    - Nếu q chưa có trong Open
        - g(q) = g(p) + cost(p, q)
        - f(q) = g(q) + h(q)
        - prev(q) = p
        - Thêm q vào Open
    - Nếu q có trong Close
        - Nếu g(q) > g(p) + Cost(p, q)
            - Bỏ q khỏi Close
            - Thêm q vào Open

- Bước 3: Không tìm được.

V. SO SÁNH UCS VÀ A*:

Uniform-cost search là tìm kiếm không có thông tin: không sử dụng bất kỳ domain knowledge nào. UCS mở rộng nút có chi phí thấp nhất và làm như vậy theo mọi hướng vì không có thông tin về mục tiêu được cung cấp.

UCS có thể được xem như một hàm *f (n) = g (n)*, trong đó g (n) là chi phí đường dẫn (đơn giản là một chi phí để đạt được nút n)

Best-first search là tìm kiếm có thông tin: nó sử dụng hàm heuristic để ước tính mức độ gần nhất của trạng thái hiện tại với mục tiêu *(có đang tiến gần đến mục tiêu không?)*.

Hàm chi phí *f(n) = g(n)* được kết hợp với chi phí để đi từ n đến mục tiêu, h (n) (hàm heuristic ước tính chi phí đó)

Nên *f(n) = g(n) + h(n)* **(trong đó thuật toán A * là Một ví dụ về điển hình thuật toán Best-first search).**

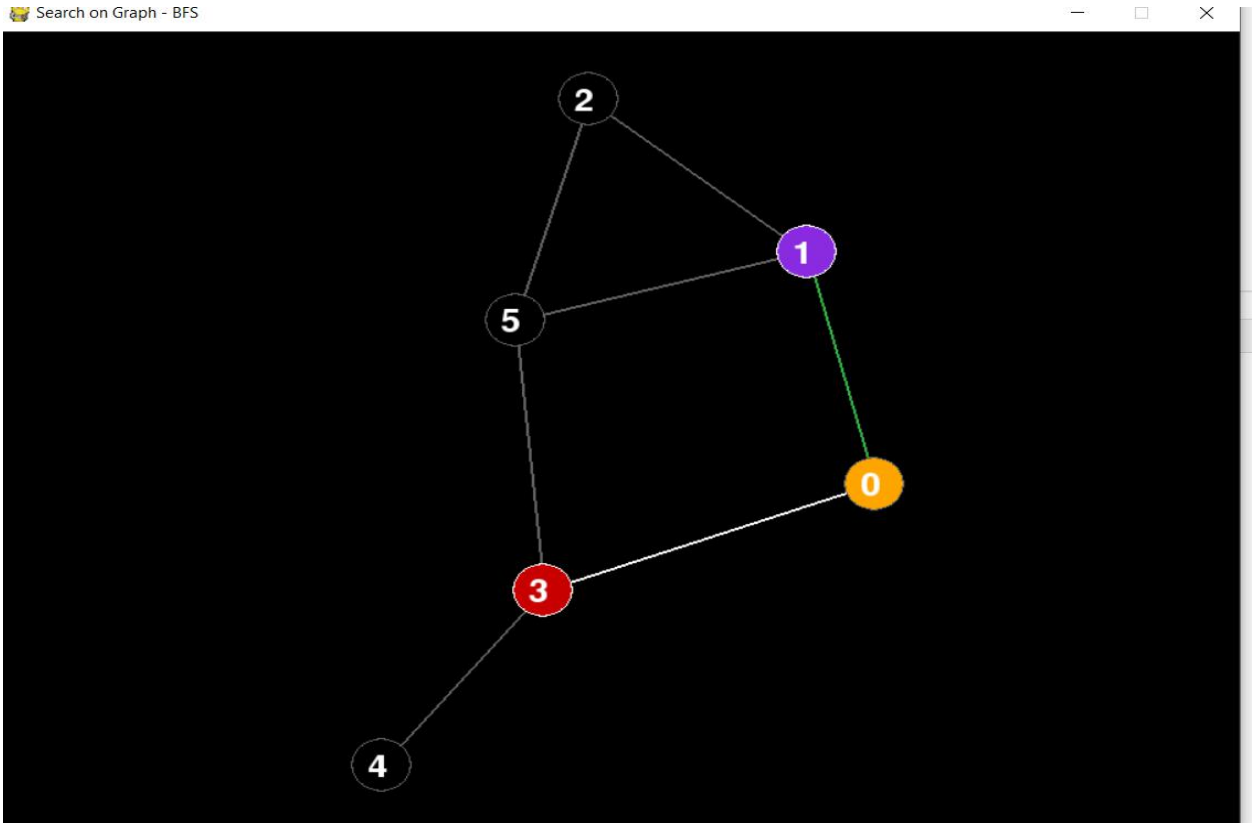
➔Cả hai phương pháp đều có danh sách các nút được mở rộng, nhưng **Best-first search** sẽ cố gắng giảm thiểu số lượng các nút mở rộng đó *(chi phí đường dẫn + hàm heuristic)*.

C-THỰC HÀNH:

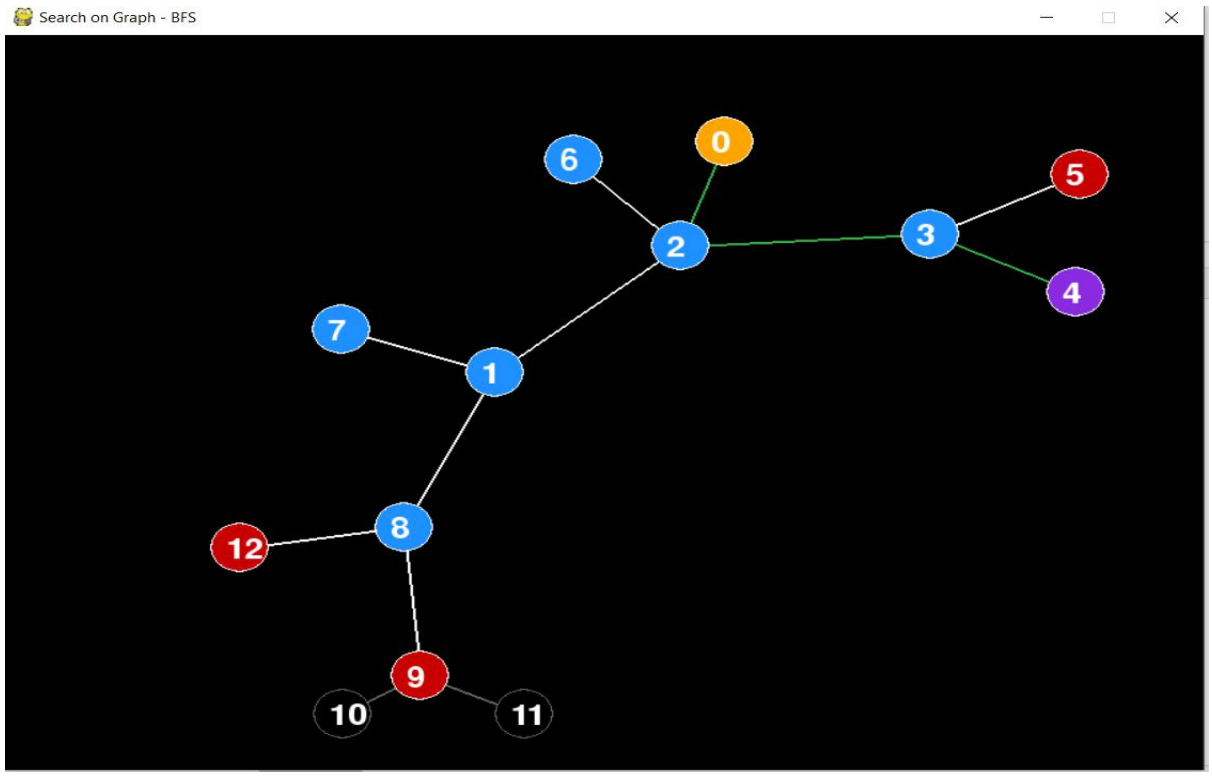
I. Dữ liệu test: nội dung cái file input như sau:
testcase1.txt, testcase2.txt và testcase3.txt nằm trong folder **video**

II . Kết quả:

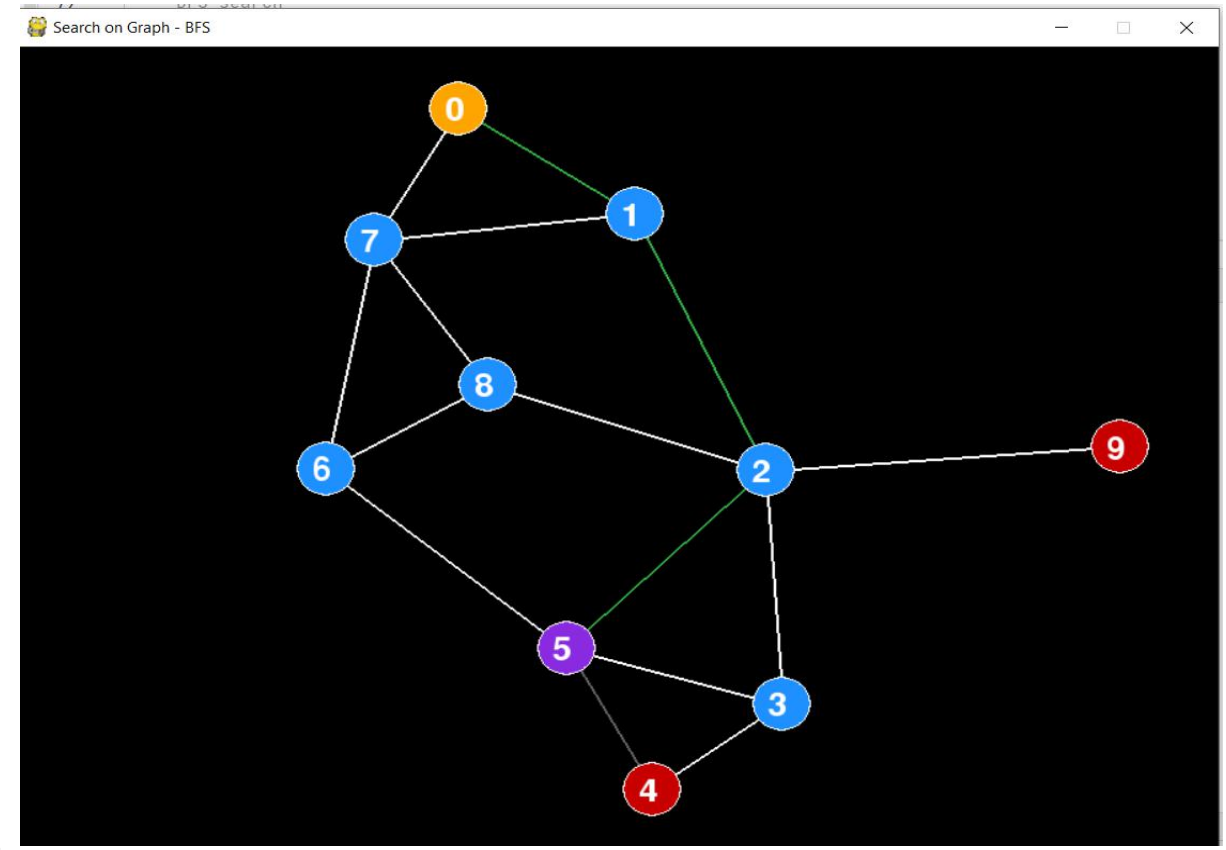
a, BFS:



testcase1.txt

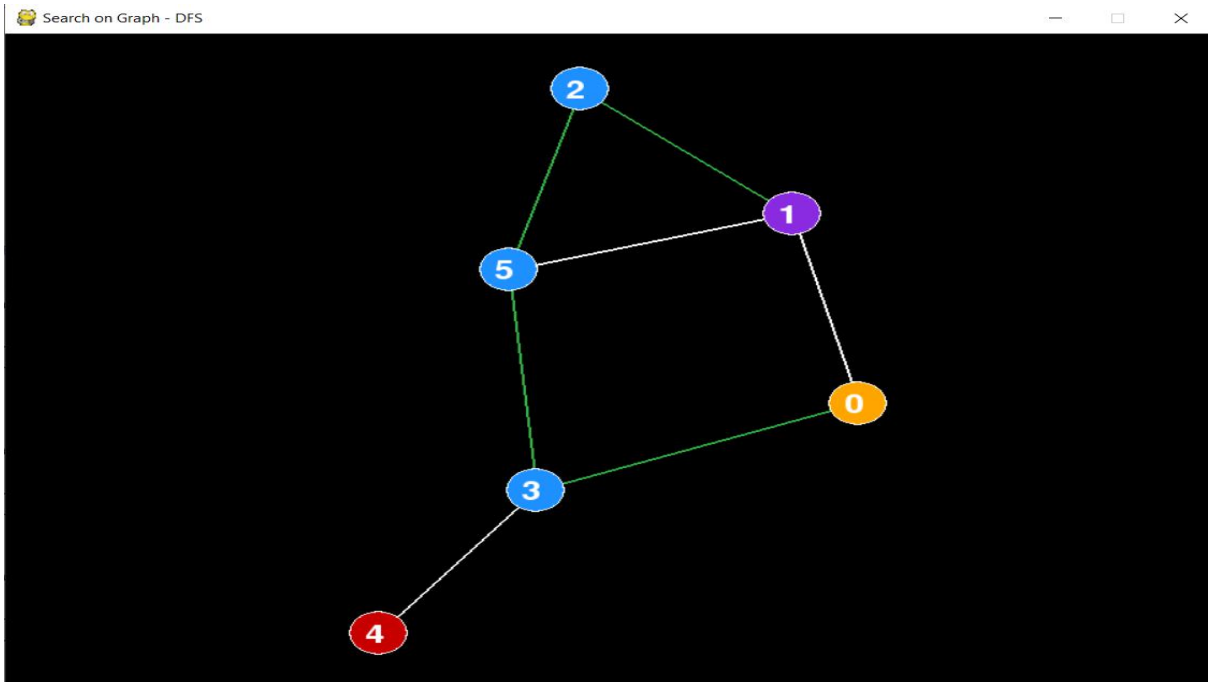


Testcase2.txt

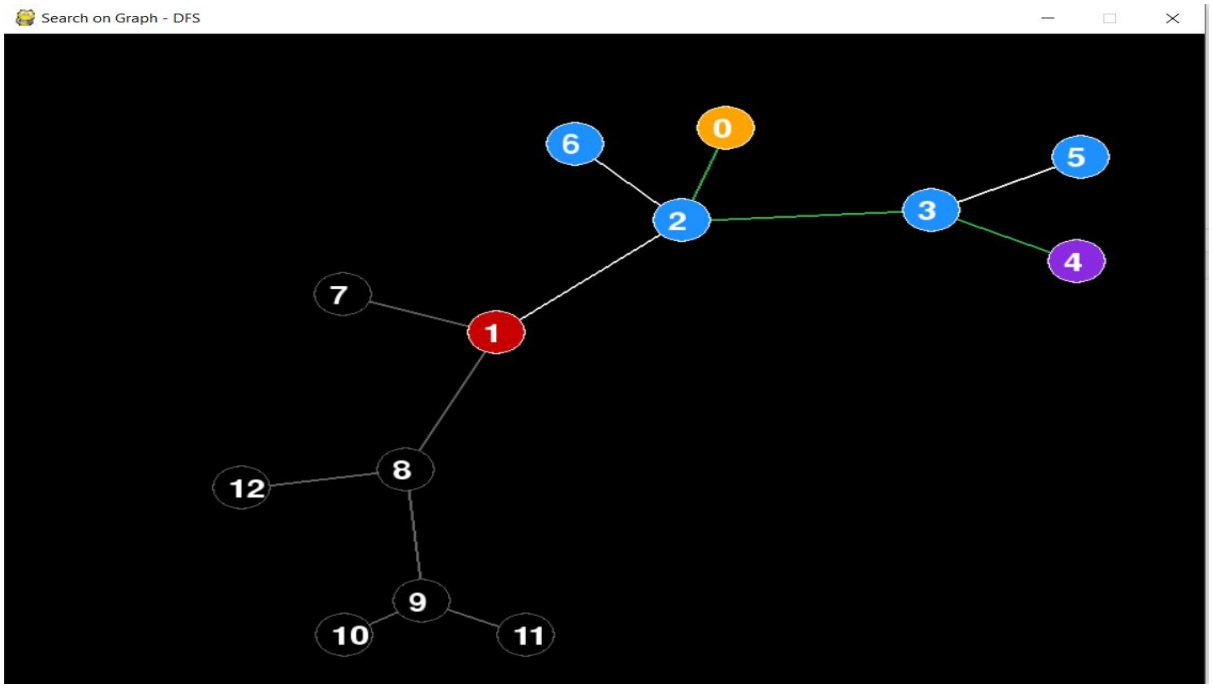


Testcase3.txt

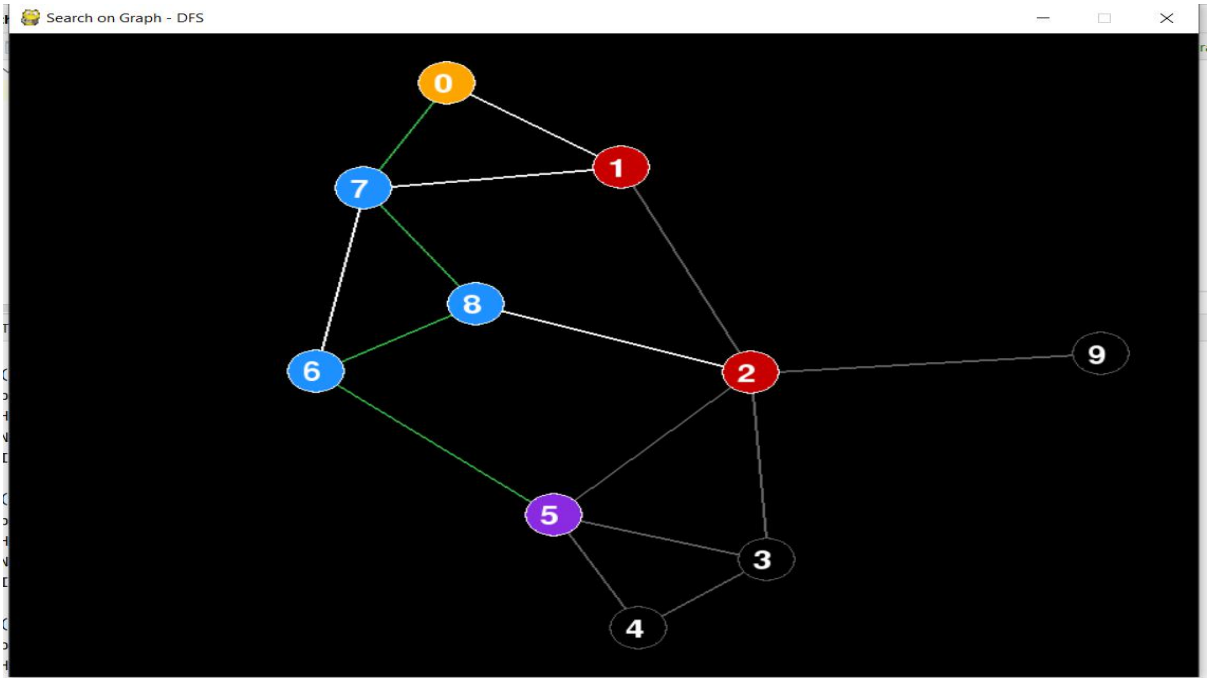
b, DFS:



Testcase1.txt

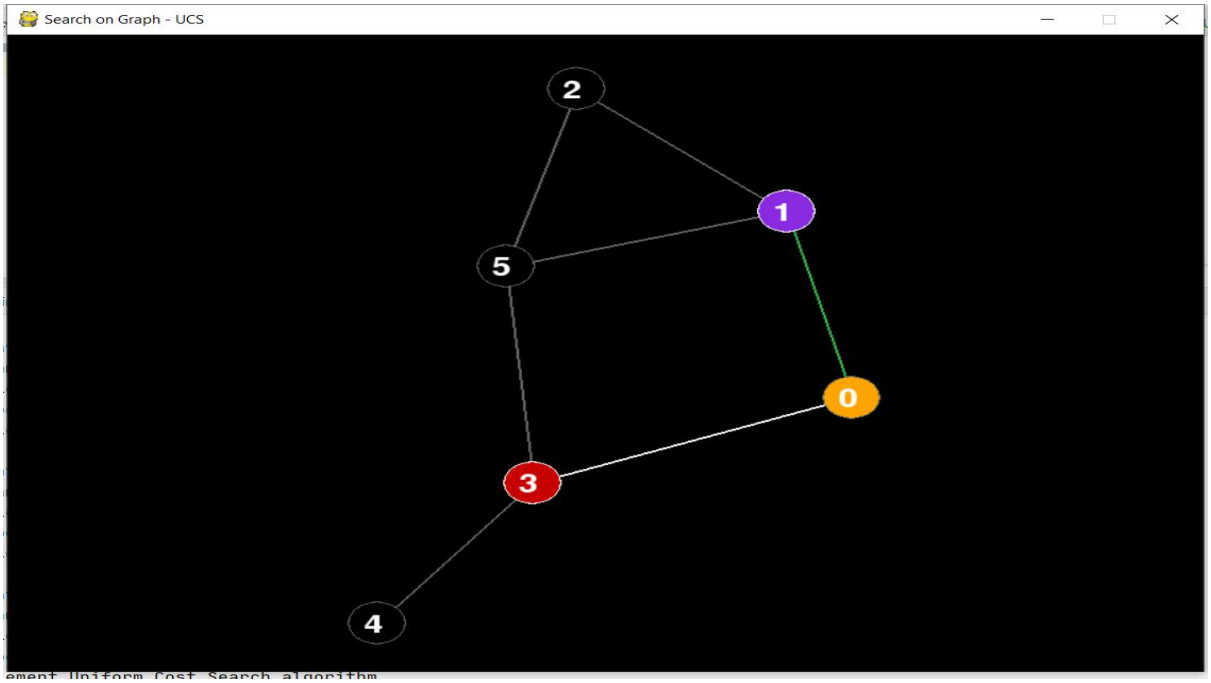


Testcase2.txt

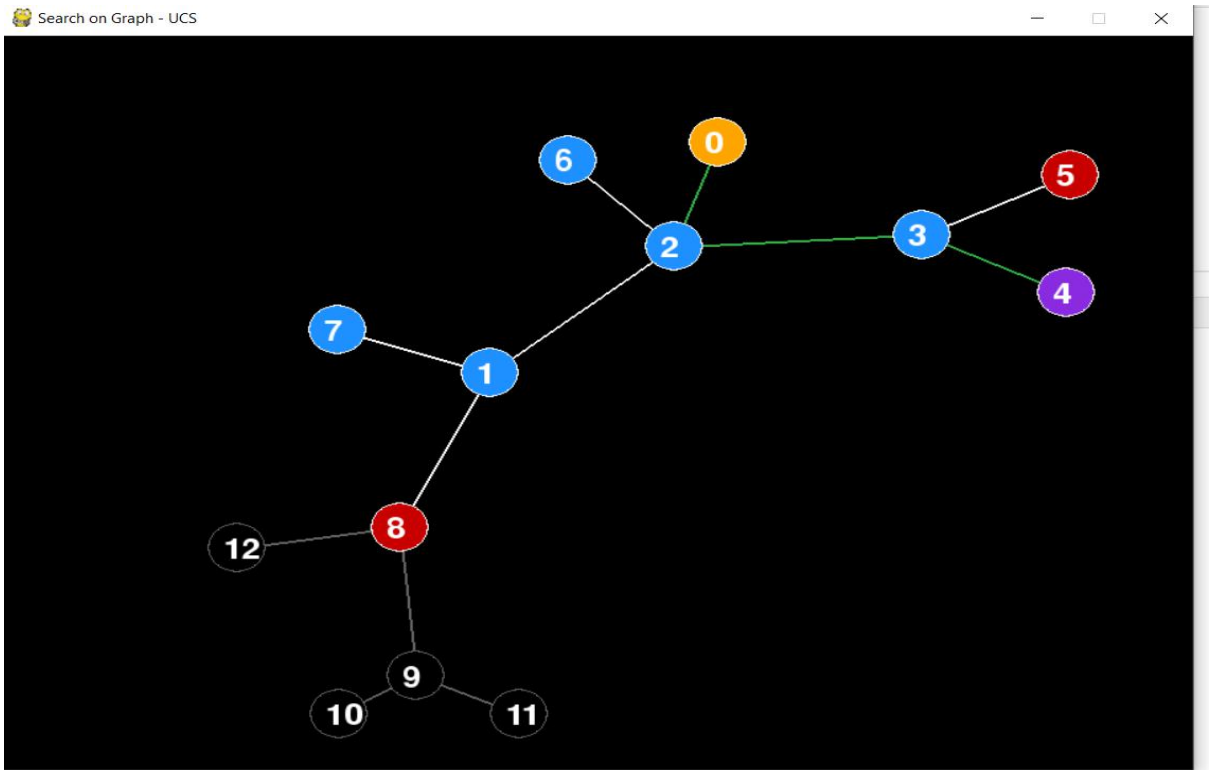


Testcase3.txt

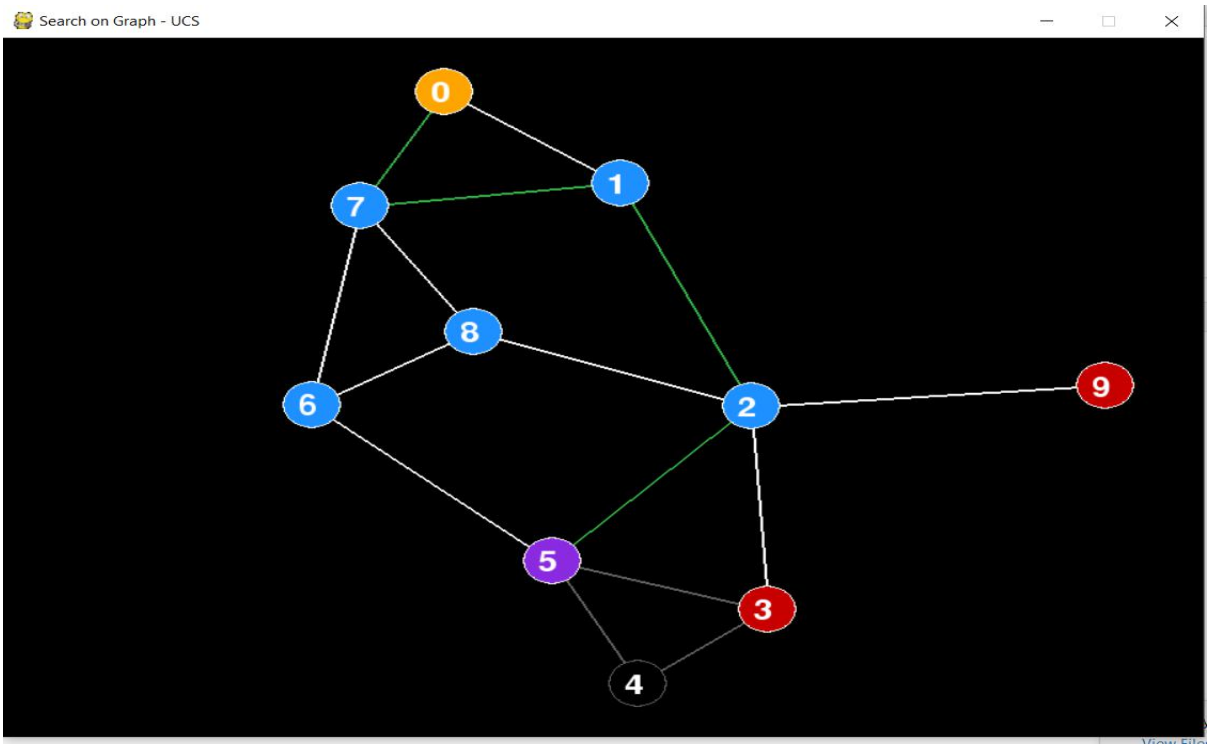
c, UCS:



Testcase1.txt



Testcase2.txt



Testcase3.txt

d, A*:

Nguồn tham khảo:

https://www.stdio.vn/giai-thuat-lap-trinh/thuat-toan-depth-first-search-APnHi1
https://viblo.asia/p/cac-thuat-toan-co-ban-trong-ai-phan-biet-best-first-search-va-uniform-cost-search-ucs-Eb85omLWZ2G
https://stackoverflow.com/questions/19204682/time-complexity-of-uniform-cost-search
https://www.academia.edu

https://www.ics.uci.edu/~eppstein/161/960215.html

https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

Và nhiều nguồn khác