TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TPHCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

GV hướng dẫn: thầy Nguyễn Thái Vũ

Thực hiện: Nhóm AI00

TPHCM, tháng 7, năm 2022

NỘI DUNG

I. Thông tin nhóm

MSSV	Họ và tên
18127008	Lê Mạnh Hoàng
19127197	Hoàng Thị Quỳnh Liên
19127273	Huỳnh Thị Mỹ Thanh

II. Finding path by search algorithm

1. *Breadth – first search (BFS)*

*** Ý tưởng**

• Dùng 1 hàng đợi queue để lưu các điểm hiện tại có thể lấy ra để mở rộng

• Dùng mảng visited để lưu các điểm được mở.

• Trong khi hàng đợi còn phần tử:

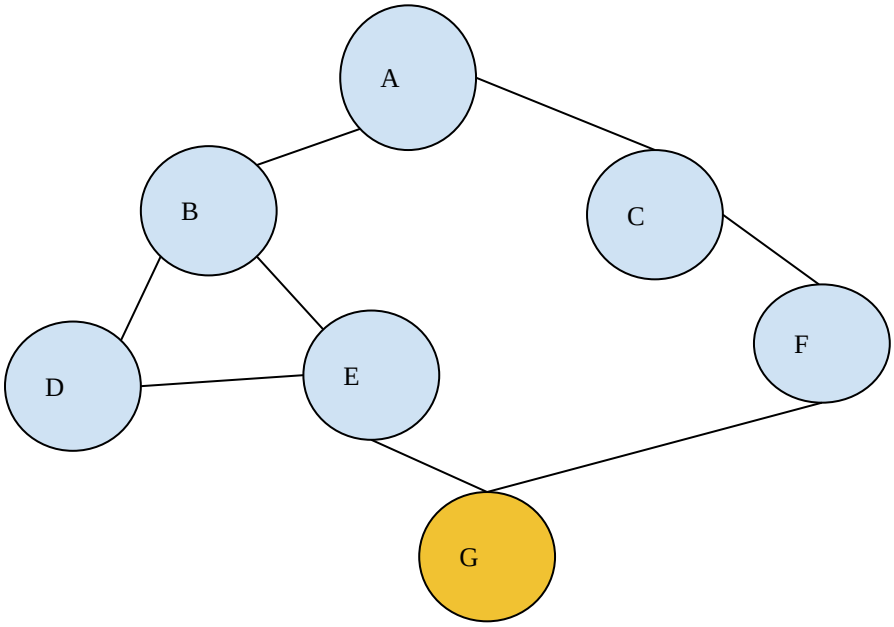
o Lấy 1 điểm ra

▪ Nếu điểm đó là goal thì dừng thuật toán.

▪ Ngược lại mở rộng điểm hiện tại ra các hướng có thể.

▪ Nếu điểm được mở rộng chưa nằm trong visited và queue thì thêm vào queue, đánh dấu và cập nhật đường đi cùng chi phí.

*** Ví dụ**



Bước 1:

- Visited:

A	B	C	D	E	F	G
0	0	0	0	0	0	0

- Queue: Empty

Bước 2:

- Visited:

A	B	C	D	E	F	G
1	0	0	0	0	0	0

- Queue: A

Bước 3:

- Visited:

A	B	C	D	E	F	G
1	0	0	0	0	0	0

- Queue: Empty

- Output: A

Bước 4:

- Visited:

A	B	C	D	E	F	G
1	1	1	0	0	0	0

- Queue: B, C

- Output: A

Bước 5:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	0	0

- Queue: C, D, E

- Output: A, B

Bước 6:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	1	0

- Queue: D, E, F

- Output: A, B, C

Bước 7:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	1	1

- Queue: E, F, G

- Output: A, B, C, D

Bước 8:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	1	1

- Queue: F, G

- Output: A, B, C, D, E

Bước 9:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	1	1

- Queue: G

- Output: A, B, C, D, E, F

Bước 10:

- Visited:

A	B	C	D	E	F	G
1	1	1	1	1	1	1

- Queue: Empty

- Output: A, B, C, D, E, F, G

⇒ A → B → C → D → E → F → G

*** Kết luận, ưu điểm và nhược điểm**

- Ưu điểm:

- Tối ưu cho việc tìm khoảng cách ngắn nhất (không phải chi phí).

- Nhược điểm:

- Tìm kiếm một cách máy móc (khi không có thông tin hỗ trợ cho quá trình tìm kiếm, không nhận ra ngay lời giải).

- Không phù hợp với không gian bài toán kích thước lớn. Đối với loại bài toán này, phương pháp tìm rộng đối mặt với các nhu cầu:

  + Cần nhiều bộ nhớ theo số nút cần lưu trữ.

  + Cần nhiều công sức xử lý các nút, nhất là khi các nhánh cây dài, số nút tăng.

  + Dễ thực hiện các thao tác không thích hợp, thừa, đưa đến việc tăng đáng kể số nút phải xử lý.

- Không hiệu quả nếu lời giải ở sâu. Phương pháp này không phù hợp cho trường hợp có nhiều đường dẫn đến kết quả nhưng đều sâu.

2. *Uniform – cost search (UCS)*

*** Ý tưởng:**

f(n) = g(n)

• Hàm tìm điểm tối ưu nhất sẽ tìm điểm mà tại đó tổng đường đi từ start đến điểm hiện tại là nhỏ nhất.

• Dùng 1 list để lưu các đỉnh đã được mở, thuật toán sẽ dừng khi đã tìm được đích hoặc không còn đỉnh nào mở được nữa (không có đường đi):

o Lấy 1 điểm được cho là tối ưu nhất có thể đến được đích.

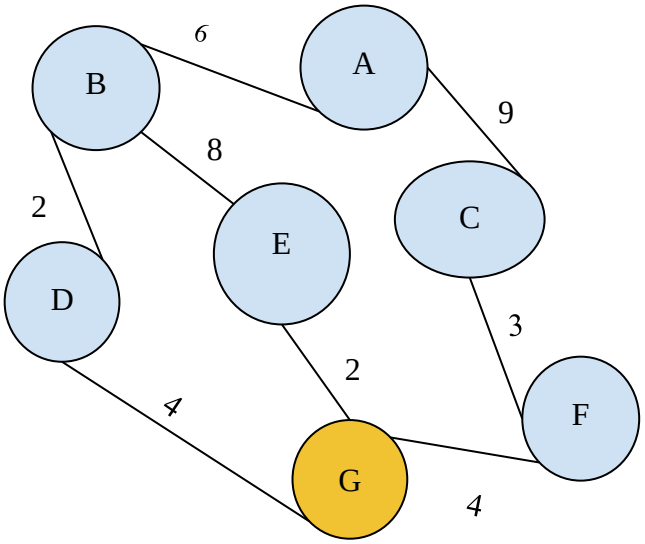
o Kiểm tra nếu là đích thì trả về kết quả.

o Nếu không phải là đích: với điểm lấy được, ta mở rộng ra các hướng có thể

▪ Với mỗi điểm mở được:

• Nếu điểm chưa được duyệt hoặc chi phí đường đi mới tốt hơn thì cập nhật lại đường đi mới và các chi phí.

*** Ví dụ**



Bước 1:

- Chọn A là node bắt đầu
- Tính: g(B) = 6, g(C) = 9
- g(B) < g(C) → Chọn node B

Bước 2:

- Tính: g(D) = 6 + 2 = 8, g(E) = 6 + 8 = 14
- g(D) < g(E) → Chọn node D

Bước 3:

- Tính g(G) = 6 + 2 + 4 = 12

⇒ A → B → D → G

*** Kết luận, ưu điểm và nhược điểm**

- Ưu điểm: Chọn đường đi có chi phí thấp nhất.

- Nhược điểm: chỉ quan tâm đến chi phí nên có thể mắc kẹt trong vòng lặp vô tận.

3. *Greedy – best first search (GBFS)*

*** Ý tưởng**

f(n) = h(n)

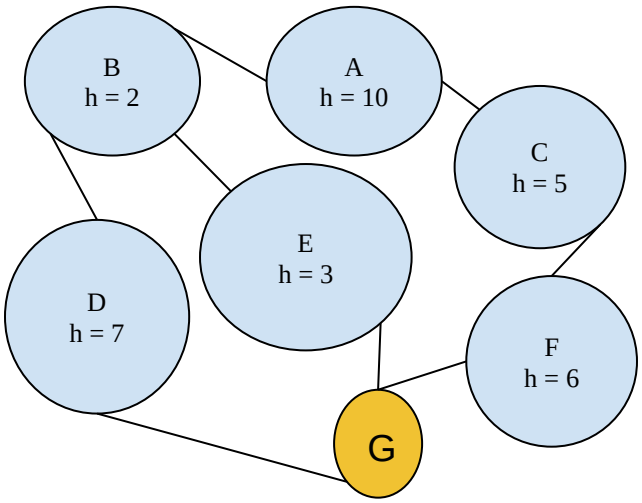
•Hàm heuristic (khoảng cách manhattan) làm hàm ước lượng.

•Hàm tìm điểm tối ưu nhất sẽ tìm điểm mà tại đó heuristic từ điểm hiện tại đến điểm tiếp theo là nhỏ nhất.

• Dùng 1 list để lưu các đỉnh đã được mở, thuật toán sẽ dừng khi đã tìm được đích hoặc không còn đỉnh nào mở được nữa (không có đường đi):

- o Lấy 1 điểm được cho là tối ưu nhất có thể đến được đích.
- o Kiểm tra nếu là đích thì trả về kết quả.
- o Nếu không phải là đích: với điểm lấy được, ta mở rộng ra các hướng có thể
  - ▪ Với mỗi điểm mở được:
    - • Nếu điểm chưa được duyệt hoặc chi phí đường đi mới tốt hơn thì cập nhật lại đường đi mới và các chi phí.

*** Ví dụ**



Bước 1:

- Chọn A là node bắt đầu.
- Tính: h(B) = 2, h(C) = 5
- h(B) < h(C) → Chọn node B

Bước 2:

- Tính h(D) = 7, h(E) = 3
- h(E) < h(D) → Chọn node E

Bước 3:

- h(G) = 0

⇒ A → B → E → G

*** Kết luận, ưu điểm và nhược điểm**

- Ưu điểm:

- Độ phức tạp về thời gian có thể cải thiện nếu hàm heuristic là tốt.

- Nhược điểm:

- Không có tính hoàn chỉnh.
- Độ phức tạp về bộ nhớ lớn (vì lưu giữ tất cả các node trong bộ nhớ).
- Phụ thuộc vào hàm heuristic mới đánh giá được kết quả có cải thiện hay không.

4. *Graph – search A**

*** Ý tưởng**

f(n) = g(n) + h(n)

• Dùng hàm heuristic (khoảng cách manhattan) làm hàm ước lượng.

• Hàm tìm điểm tối ưu nhất sẽ tìm điểm mà tại đó tổng đường đi từ start đến điểm hiện tại và heuristic từ điểm hiện tại đến goal là nhỏ nhất.

• Dùng 1 list để lưu các đỉnh đã được mở, thuật toán sẽ dừng khi đã tìm được đích hoặc không còn đỉnh nào mở được nữa (không có đường đi):

o Lấy 1 điểm được cho là tối ưu nhất có thể đến được đích.

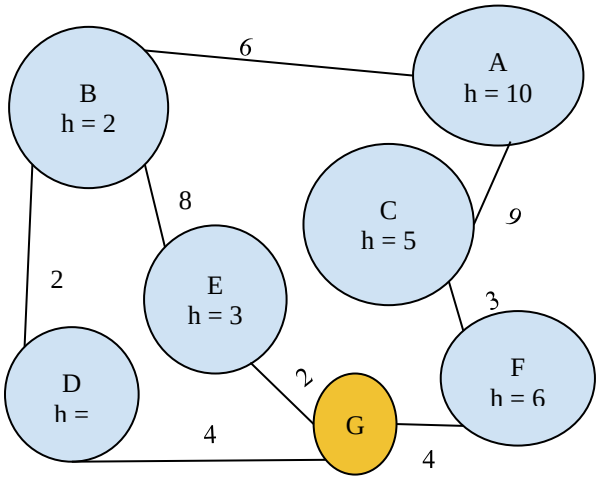
o Kiểm tra nếu là đích thì trả về kết quả.

o Nếu không phải là đích: với điểm lấy được, ta mở rộng ra các hướng có thể

▪ Với mỗi điểm mở được:

• Nếu điểm chưa được duyệt hoặc chi phí đường đi mới tốt hơn thì cập nhật lại đường đi mới và các chi phí.

*** Ví dụ**



Bước 1:

- Chọn A là node bắt đầu
- Tính: f(B) = 6 + 2 = 8, f(C) = 5 + 9 = 14
- f(B) < f(C) → chọn node B

Bước 2:

- Tính: f(D) = (2 + 6) + 7 = 15, f(E) = (6 + 8) + 3 = 17
- f(D) < f(E) → chọn node D

Bước 3:

- f(G) = (6 + 2 + 4) + 0 = 12

⇒ A → B → D → G

*** Kết luận, ưu điểm và nhược điểm**

- Ưu điểm:

- Giải quyết được các vấn đề tìm kiếm phức tạp.
- Không có thuật toán tối ưu nào khác đảm bảo việc mở rộng ít node hơn A*.
- Là một trong những kỹ thuật tìm kiếm heuristic tốt nhất.

- Nhược điểm:

- Hiệu suất của thuật toán A* phụ thuộc vào độ chính xác của heuristic được sử dụng để tính toán hàm h(n).