TRƯỜNG ĐH KHOA HỌC TỰ NHIÊN, ĐHQG - HCM
KHOA TOÁN-TIN HỌC

——————o0o——————

ĐỒ ÁN CUỐI KỲ

Các thuật toán tìm kiếm trên đồ thị

Tác giả:
Huỳnh Lê Minh Thư
Nguyễn Thị Lan Diệp

Giáo viên hướng dẫn:
Nguyễn Bảo Long

Ngày hoàn thành
Ngày 11 tháng 12 năm 2022

Tên học phần:
Thực hành Cấu trúc dữ liệu và giải thuật

Lời cảm ơn

Lời đầu tiên, chúng em xin gửi lời cảm ơn đến thầy Bùi Tiến Lên - giảng viên lý thuyết và thầy Nguyễn Bảo Long - giảng viên thực hành môn Cấu trúc Dữ liệu và Giải thuật, lớp 21KDL1. Cảm ơn thầy đã đưa đề tài, tận tình hướng dẫn cũng như cung cấp tài liệu, thông tin khoa học cần thiết trong quá trình học tập và cả trong đồ án cuối kỳ này.

Đề tài "Tìm hiểu các thuật toán tìm kiếm trên đồ thị"là đề tài thú vị, vô dùng bổ ích và có tính thực tế cao. Thông qua quá trình thực hiện đồ án, chúng em đã hiểu sâu hơn, kỹ hơn về các thuật toán tìm kiếm trên đồ thị. Có được những kiến thức, kinh nghiệm mới mẻ và có thể sẽ sử dụng đến trong quá trình sau này của chúng em.

Trong quá trình thực hiện đồ án, do thời gian không nhiều và trình độ của nhóm có hạn. Tuy đã cố gắng hết sức nhưng chắc chắn bài báo cáo khó tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác.Chúng em rất mong nhận được góp ý và chia sẻ từ quý thầy để nhóm có thể cải thiện đồ án, rút kinh nghiệm để thực hiện các dự án sau này tốt hơn!

Chúng em xin chân thành cảm ơn!

Thành phố Hồ chí Minh, ngày 10 tháng 12 năm 2022

Các thành viên nhóm

Mục lục

Chương 1

Tổng quan về đồ án

1.1 Mô tả về đồ án

- Tên đồ án: Các thuật toán tìm kiếm trên đồ thị
- Đối tượng thực hiện: nhóm 2 sinh viên học lớp 21KDL1 môn Thực hành Cấu trúc dữ liệu và giải thuật
- Thời gian: từ ngày 24/11/2022 đến hết ngày 11/12/2022

1.1.1 Bài toán đặt ra

. Đồ thị là một đối tượng tổ hợp (combinatorial object) được nghiên cứu và ứng dụng rất nhiều trong thực tế ví dụ như bài toán tìm đường đi trên bản đồ giao thông, bài toán làm lịch,... Vì tính thiết yếu của nó mà đồ án này sẽ giới thiệu một số cách tìm kiếm trên đồ thị.

Thuật toán tìm đường đi trên đồ thị cơ bản được chia thành 2 loại: tìm kiếm mù và tìm kiếm kinh nghiệm (heuristics). Trong đò án này, thuật toán tìm kiếm mù là DFS, BFS, UCS, Dijkstra và tìm kiếm kinh nghiệm là A*.

1.1.2 Yêu cầu cụ thể

Tìm hiểu và Trình bày thuật toán

Trình bày về 4 thuật toán DFS, BFS, UCS, A* . Nội dung gồm có:

1. Ý tưởng chung
2. Mã giả
3. Đưa ra đánh giá về thuật toán (tính đầy đủ, tính tối ưu, độ phức tạp)
4. Ví dụ đơn giản (đồ thị chứa 5-6 nodes) để minh họa cho mã giả. Ví dụ cho các thuật toán nên lấy cùng một đồ thị để thấy được ưu nhược điểm của từng thuật toán.
5. Không sử dụng đệ quy trong cài đặt

So sánh

1. So sánh sự khác biệt giữa 2 thuật toán DFS, BFS

2. So sánh sự khác biệt giữa 2 thuật toán UCS, Dijkstra

Cài đặt

1. Nhiệm vụ là tìm kiếm đường đi giữa 2 node trong đồ thị như hình vẽ
2. Đọc code được cung cấp, hoàn thành các hàm chưa được cài đặt: DFS, BFS, UCS
3. Với mỗi thuật toán, chụp màn hình lại kết quả cuối và mô tả ngắn gọn quá trình tìm kiếm vào báo cáo. Nêu nhận xét về kết quả thu được
4. Quá trình chạy phải được quay lại trong 1 video. Cần phân đoạn rõ ràng từng thuật toán. Upload video lên Youtube và đính kèm đường link vào file link.txt . Mỗi video không nên dài quá 3 phút. Có thể tăng tốc video bằng phần mềm.
5. Tuân thủ các quy định về màu sắc cho các loại node và màu của đường đi (Constants.py).

1.1.3 Phần mềm thực hiện

Dưới đây là các phần mềm được sử dụng trong suốt đồ án:

1. Microsoft Word: soạn thảo nội dung tìm hiểu
2. OverLeaf: soạn thảo báo cáo bằng Latex
3. Onedrive: lưu trữ, chia sẻ dữ liệu
4. PyCharm: dùng để code và run code Python
5. Messenger: dùng để liên lạc, trao đổi thông tin
6. Video Editor Movavi: edit video
7. R Studio: vẽ biểu đồ nhận xét
8. ...

1.1.4 Giải thích các file trong bài nộp

Bài nộp là tệp nén có dạng 21280110-21280123.zip gồm 3 file dưới đây

1. file link.txt chứa đường link video demo
2. file report.pdf chứa báo cáo tìm hiểu
3. thư mục ./source chứa source code.

1.2 Mục tiêu

Hiểu sâu hơn về các thuật toán tìm kiếm trên đồ thị. Sử dụng ngôn ngữ lập trình Python để thực hiện các yêu cầu liên quan đến cài đặt các thuật toán tìm kiếm. Trau dồi kỹ năng làm việc nhóm. Thành thạo sử dụng các phần mềm liên quan.

Chương 2

Kế hoạch thực hiện

2.1 Thông tin nhóm

Họ và tên	MSSV	Email
Huỳnh Lê Minh Thư	21280110	21280110@student.hcmus.edu.vn
Nguyễn Thị Lan Diệp	21280123	21280123@student.hcmus.edu.vn

BẢNG 2.1: Thông tin thành viên

2.2 Phân công công việc

Đồ án được thực hiện trong 2.5 tuần, từ ngày 23/11/2022 đến hết ngày 11/12/2022. Phân chia công việc và thời hạn deadline được thực hiện hai ngày sau khi nhận được đề tài đồ án từ giảng viên.

STT	TÊN CÔNG VIỆC	NGƯỜI THỰC HIỆN	DEADLINE	TIẾN ĐỘ
1	Tìm hiểu DFS, BFS UCS	Minh Thư	28/11/2022	Hoàn thành
2	Tìm hiểu A*, Dijkstra	Lan Diệp	28/11/2022	Hoàn thành
3	Làm ví dụ cho DFS, BFS, UCF	Minh Thư	30/11/2022	Hoàn thành
4	Làm ví dụ cho A*, Dijkstra	Lan Diệp	30/11/2022	Hoàn thành
5	So sánh DFS, BFS	Lan Diệp	03/12/2022	Hoàn thành
6	So sánh UCS, Dijkstra	Minh Thư	03/12/2022	Hoàn thành
7	Check lại phần tìm hiểu	Cả hai	04/12/2022	Hoàn thành
8	Cài đặt thuật toán BFS, DFS, UCS	Minh Thư	08/12/2022	Hoàn thành
9	Cài đặt thuật toán A Star	Lan Diệp	08/12/2022	Hoàn thành
10	Viết báo cáo	Cả hai	10/12/2022	Hoàn thành
11	Kiểm tra lại bài	Cả hai	11/12/2022	Hoàn thành
12	Nộp bài	Minh Thư	11/12/2022	Hoàn thành

BẢNG 2.2: Phân công công việc

2.3 Đánh giá bài làm

Trong quá trình làm đồ án, các thành viên trong nhóm tham gia và hoàn thành công việc đầy đủ. Tuy nhiên do thời gian có hạn, một số task bị trễ deadline nhưng không ảnh hưởng đến các task còn lại.

Quá trình thực hiện có gặp một số khó khăn. Một số thuật toán không nhiều nguồn tài liệu uy tín. Việc tràn bộ nhớ, đứng máy do có một số sai sót trong quá trình cài đặt. Tuy nhiên các thành viên trong nhóm

đã tìm hiểu và xử lý tốt các vấn đề phát sinh để hoàn thiện đồ án.
Dưới đây là bảng đánh giá theo thang điểm 10 với từng yêu cầu được đề ra.

Yêu cầu		**Tỉ lệ điểm**	**Đánh giá**
Tìm hiểu và trình bày được các thuật toán	Ý tưởng chung	10%	10
	Mã giả.	10%	10
	Đưa ra đánh giá về thuật toán	10%	10
So sánh các thuật toán	DFS vs BFS	10%	10
	UCS Dijkstra	10%	10
Cài đặt được các thuật toán	Cài đặt DFS, BFS, UCS, AStar	30%	10
Tìm hiểu thêm các thuật toán	Dijkstra và AStar	10%	10

BẢNG 2.3: Bảng đánh giá bài làm

Đánh giá điểm bài làm: 10/10 (với sai số ≤ 0.5)

Chương 3

Tìm hiểu và trình bày thuật toán

3.1 BFS

3.1.1 Ý tưởng

Thuật toán Breadth-first search – BFS (tìm kiếm đồ thị ưu tiên chiều rộng) là một trong những thuật toán tìm kiếm cơ bản và thiết yếu trên đồ thị. Thuật toán BFS bắt đầu tìm kiếm từ nút gốc của cây và mở rộng tất cả các nút kế thừa ở cấp hiện tại trước khi chuyển sang các nút của cấp tiếp theo. Các bước của thuật toán như sau:

- Sử dụng 1 danh sách OPEN để lưu các đỉnh được sinh ra và chờ duyệt. Đỉnh nào được sinh ra trước sẽ được duyệt trước nên danh sách OPEN được xử lí như queue (hàng đợi)
- Đỉnh đã được duyệt sẽ không được duyệt lại nên ta dùng mảng phụ CLOSED để lưu vết đường đi tiện cho việc truy vết.
- Khi danh sách OPEN không rỗng:
  - Lấy đỉnh đầu v ra khỏi OPEN và cho vào CLOSED. Nếu v là đích thì tìm kiếm thành công $\rightarrow$ thoát
  - Tìm tất cả các đỉnh con của v không thuộc OPEN và CLOSED cho vào cuối OPEN
  - Lặp lại các bước trên đến khi OPEN không còn phần tử

3.1.2 Mã giả

Algorithm 1 Thuật toán BFS

```
1:  procedure BFS(g, u, x)
2:      OPEN ← u
3:      CLOSED ← ∅
4:      while OPEN ≠ ∅ do
5:          v ⟵ DEQUEUE(OPEN)
6:          if v = x then
7:              return suceed
8:          else:
9:              v ⟵ ENQUEUE(CLOSED)
10:             neighbors ⟵ GET_NEIGHBOR(v)
11:             for i ∈ neighbors do          ▷ Tập hợp các node v có thể đi đến
```

12: **if** $i \notin OPEN$ & $i \notin CLOSED$ **then**
13: $i \longleftarrow$ ENQUEUE($OPEN$)
14: $father[i] \leftarrow v$ ▷ Node cha của i là v
15: **return** failed

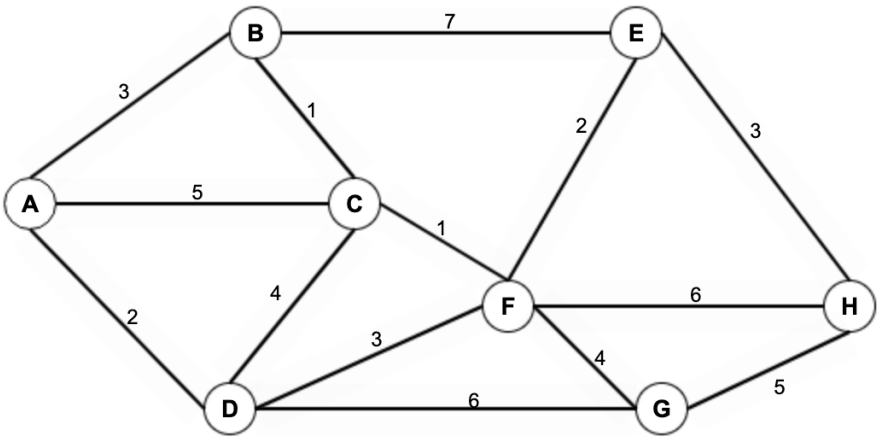
3.1.3 Đánh giá

Tính đầy đủ	Có tính đầy đủ vì luôn tìm thấy một lời giải nếu bài toán có lời giải.
Tính tối ưu	Giải thuật BFS luôn luôn trả về đường đi ngắn nhất (đi qua ít cạnh nhất) nên có tính tối ưu cho việc tìm khoảng cách ngắn nhất
Độ phức tạp	Về thời gian là O(V+E) Về không gian là O(V)

BẢNG 3.1: Đánh giá thuật toán BFS

3.1.4 Ví dụ

Cho đồ thị như hình vẽ:



Thuật toán BFS tìm đường đi ngắn nhất từ 1 đỉnh tới đỉnh bất kỳ. Ở ví dụ này, chúng tôi chọn đỉnh đầu là A, đỉnh cuối là H.

Current Node	Open Set	Closed Set
	A	$\emptyset$
A	D, C, B	A
D	G, F, C, B	A, D
G	H, F, C, B	A, D, G
H	F, C, B	A, D, G

BẢNG 3.2: Mô tả cách chạy của thuật toán BFS

Vậy đường đi từ A đến H là A $\rightarrow$ D $\rightarrow$ G $\rightarrow$ H

3.2 DFS

3.2.1 Ý tưởng

Thuật toán Depth-first search – DFS (tìm kiếm đồ thị ưu tiên chiều sâu) là một trong những thuật toán tìm kiếm cơ bản và thiết yếu trên đồ thị. DFS duyệt sâu theo một nhánh con của một đỉnh, khi không còn nút con, DFS quay lại nút cha và tiếp tục duyệt. Các bước của thuật toán như sau:

1. Sử dụng 1 danh sách OPEN đêr lưu các đỉnh được sinh ra và chờ duyệt. Đỉnh được sinh ra trước tiên sẽ được duyệt sau cùng trong số các đỉnh chờ duyệt nên danh sách OPEN được xử lí như stack (ngăn xếp)

2. Đỉnh đã được duyệt sẽ không được duyệt lại nên ta dùng mảng phụ CLOSED để lưu vết đường đi tiện cho việc truy vết.

3. Khi danh sách OPEN không rỗng:

   (a) Lấy đỉnh đầu v ra khỏi OPEN và cho vào CLOSED. Nếu v là đích thì tìm kiếm thành công $\rightarrow$ thoát

   (b) Tìm tất cả các đỉnh con của v không thuộc OPEN và CLOSED cho vào đầu OPEN

   (c) Lặp lại các bước trên đến khi OPEN không còn phần tử

3.2.2 Mã giả

Algorithm 2 Thuật toán DFS

```
1:  procedure DFS(g, u, x)
2:      OPEN ← u
3:      CLOSED ← ø
4:      while OPEN ≠ ø do
5:          v ⟵ POP(OPEN)
6:          if v = x then
7:              return suceed
8:          else:
9:              v ⟵ PUSH(CLOSED)
10:             neighbors ⟵ GET_NEIGHBOR(v)
11:             for i ∈ neighbors do
12:                 if i ∉ OPEN & i ∉ CLOSED then
13:                     i ⟵ PUSH(OPEN)
14:                     father[i] ← v
15:         return failed
```

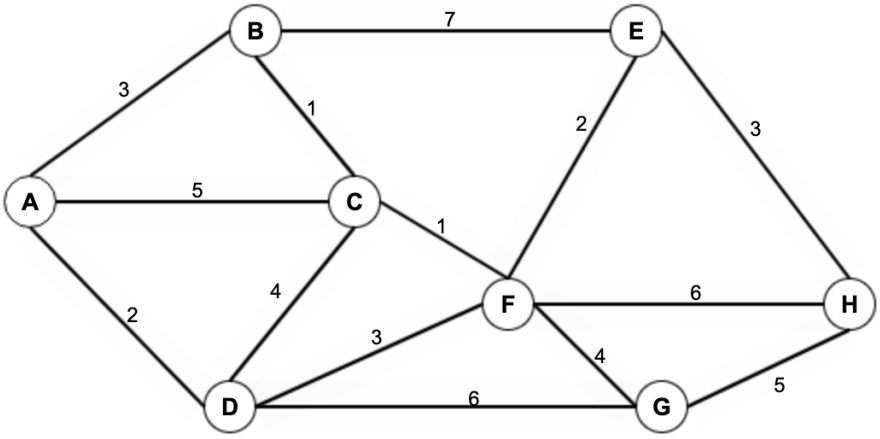
3.2.3 Đánh giá

Tính đầy đủ	Có tính đầy đủ vì luôn tìm thấy một lời giải nếu bài toán có lời giải.
Tính tối ưu	Chưa có tính tối ưu, vì nó có thể tạo ra một số lượng lớn các bước hoặc chi phí cao để đạt đến nút mục tiêu.
Độ phức tạp	Về thời gian là O(V+E) Về không gian là O(V)

BẢNG 3.3: Đánh giá thuật toán DFS

3.2.4 Ví dụ

Cho đồ thị như hình vẽ:



Thuật toán DFS tìm đường đi từ 1 đỉnh tới đỉnh bất kỳ. Ở ví dụ này, chúng tôi chọn đỉnh đầu là A, đỉnh cuối là H.

Current Node	Open Set	Closed Set
	A	∅
A	B, C, D	A
B	C, D, E	A, B
C	D, E, F	A, B, C
D	E, F, G	A, B, C, D
E	F, G, H	A, B, C, D, E
F	G, H	A, B, C, D, E, F
G	H	A, B, C, D, E, F, G
H	∅	A, B, C, D, E, F, G

BẢNG 3.4: Mô tả cách chạy của thuật toán DFS

Vậy đường đi từ A đến H là A $\rightarrow$ B $\rightarrow$ E $\rightarrow$ F $\rightarrow$ G $\rightarrow$ H

3.3 UCS

3.3.1 Ý tưởng

Thuật toán Uniform cost search – UCS (tìm kiếm chi phí cực tiểu) là một thuật toán duyệt, tìm kiếm trên một cấu trúc cây, hoặc đồ thị có trọng số (chi phí). Việc tìm kiếm bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo với trọng số hay chi phí thấp nhất tính từ nút gốc. Các bước của thuật toán như sau:

1. Sử dụng 1 danh sách OPEN để lưu các đỉnh được sinh ra và chờ duyệt. UCS sẽ duyệt nút có chi phí (trọng số) thấp nhất nên danh sách OPEN được xử lí như priority queue (hàng đợi ưu tiên)

2. Đỉnh đã được duyệt sẽ không được duyệt lại nên ta dùng mảng phụ CLOSED để lưu vết đường đi tiện cho việc truy vết.

3. Khi danh sách OPEN không rỗng:

   (a) Lấy đỉnh đầu v (có chi phí thấp nhất) ra khỏi OPEN và cho vào CLOSED. Nếu v là đích thì tìm kiếm thành công → thoát

   (b) Tìm tất cả các đỉnh con của v không thuộc OPEN và CLOSED cho vào OPEN theo thứ tự tăng dần chi phí

   (c) Lặp lại các bước trên đến khi OPEN không còn phần tử

3.3.2 Mã giả

Algorithm 3 Thuật toán UCS

```
1:  procedure UCS(g, u, x)
2:      OPEN ← u
3:      CLOSED ← ø
4:      while OPEN ≠ ø do
5:          a ⟵ MINCOST(OPEN)                ▷ Chi phí nhỏ nhất của các node trong OPEN
6:          v ⟵ POP(OPEN)
7:          if v = x then
8:              return suceed
9:          else:
10:             v ⟵ push(CLOSED)
11:             neighbors ⟵ GET_NEIGHBOR(v)
12:             for i ∈ neighbors do
13:                 if i ∉ CLOSED then
14:                     cost_temp ← a+ D(v, i)      ▷ d(v, i) là chi phí đi node hàng xóm i của v
15:                     if cost_temp < cost[i] then
16:                         i ⟵ PUSH(OPEN)
17:                         father[i] ← v
18:         return failed
```

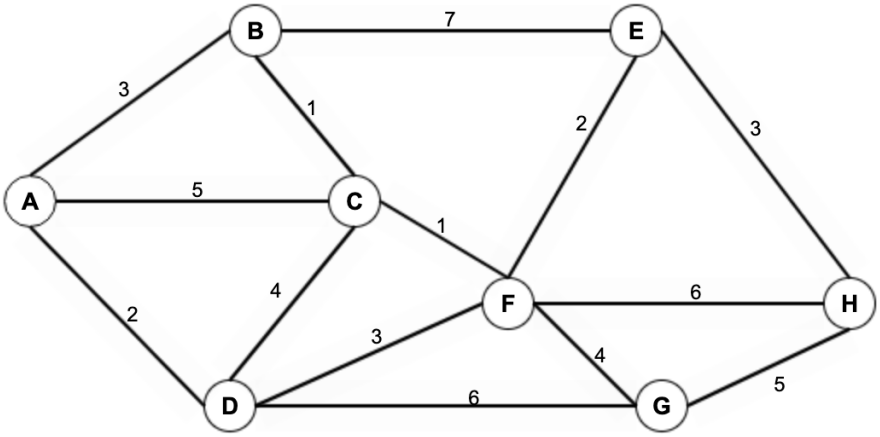
3.3.3 Đánh giá

Tính đầy đủ	Có tính đầy đủ vì luôn tìm thấy một lời giải nếu bài toán có lời giải nếu số node là hữu hạn.
Tính tối ưu	Có tính tối ưu với đồ thị có trọng số thì UCS đảm bảo tìm được lời giải có chi phí thấp nhất. Trong trường hợp đồ thị có chi phí ở mỗi bước là như nhau, UCS trở thành BFS.
Độ phức tạp	Về thời gian là O(V+E) Về không gian là O(V+E)

BẢNG 3.5: Đánh giá thuật toán UCS

3.3.4 Ví dụ

Cho đồ thị như hình vẽ:



Thuật toán UCS tìm đường đi có chi phí thấp nhất từ 1 đỉnh tới đỉnh bất kỳ. Ở ví dụ này, chúng tôi chọn đỉnh đầu là A, đỉnh cuối là H.

Current Node	Open Set	Closed Set
	A(0, A)	∅
A(0, A)	B(3, A), C(5, A), D(2, A)	A(0, A)
D(2, A)	B(3, A), C(5, A), F(5, D), G(8, D)	A(0, A) , D(2, A)
B(3, A)	C(4, B), F(5, D), G(8, D), E(10, B)	A(0, A) , D(2, A), B(3, A)
C(4, B)	F(5, D), G(8, D), E(10, B)	A(0, A) , D(2, A), B(3, A), C(4, B)
F(5, D)	G(8, D), E(7, F), H(11, F)	A(0, A) , D(2, A), B(3, A), C(4, B), F(5, D)
E(7, F)	G(8, D), H(10, E)	A(0, A) , D(2, A), B(3, A), C(4, B), F(5, D), E(7, F)
G(8, D)	H(10, E)	A(0, A) , D(2, A), B(3, A), C(4, B), F(5, D), E(7, F), G(8, D)
H(10, E)	∅	A(0, A) , D(2, A), B(3, A), C(4, B), F(5, D), E(7, F), G(8, D)

BẢNG 3.6: Mô tả cách chạy của thuật toán UCS

Vậy đường đi từ A đến H là A $\rightarrow$ D $\rightarrow$ F $\rightarrow$ E $\rightarrow$ H với tổng chi phí là 10

3.4 Dijkstra

3.4.1 Ý tưởng

Thuật toán Dijkstra cho phép chúng ta tìm đường đi ngắn nhất giữa hai đỉnh bất kỳ của đồ thị. Có thể giải quyết bài toán tìm đường đi ngắn nhất trên đồ thị vô hướng lẫn có hướng miễn là trọng số không âm. Nó khác với cây khung nhỏ nhất vì khoảng cách ngắn nhất giữa hai đỉnh có thể không bao gồm tất cả các đỉnh của đồ thị. Các bước cơ bản của thuật toán như sau:

1. Bước 1: Từ đỉnh gốc, khởi tạo khoảng cách tới chính nó là 0, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là ∞ . Ta được danh sách các khoảng cách tới các đỉnh.

2. Bước 2: Chọn đỉnh a có khoảng cách nhỏ nhất trong danh sách này và ghi nhận. Các lần sau sẽ không xét tới đỉnh này nữa.

3. Bước 3: Lần lượt xét các đỉnh kề b của đỉnh a. Nếu khoảng cách từ đỉnh gốc tới đỉnh b nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật giá trị và đỉnh kề a vào khoảng cách hiện tại của b.

4. Bước 4: Sau khi xét tất cả đỉnh kề b của đỉnh a. Lúc này ta được danh sách khoảng cách tới các điểm đã được cập nhật. Quay lại Bước 2 với danh sách này.

Khi thuật toán kết thúc, chúng ta có thể quay ngược từ đỉnh đích đến đỉnh nguồn để tìm đường dẫn.

3.4.2 Mã giả

Cho một đồ thị có hướng G(V,E), một hàm trọng số w: E $\rightarrow$ R+ và một đỉnh s. Tìm đường đi ngắn nhất từ s tới mọi đỉnh trong G

Algorithm 4 Thuật toán Dijkstra

```
1:  procedure DIJKSTRA(G(V,E),s)
2:      dist[s] ← 0                                  ▷ Distance from source to source is set to 0
3:      for each vertex v in Graph do                ▷ Initializations
4:          if v ≠ s then
5:              dist[v] ← ∞      ▷ Unknown distance function from source to each node set to infinity
6:          add v to Q                               ▷ All nodes initialy in Queue
7:
8:          while Q ≠ ∅ do
9:              v = vertex in Q with min dist[v]  ▷ In the first run-through, this vertex is the sourse node
10:             remove v from Q
11:             for each neighbor u of v do      ▷ where neighbor u has not yet been removed from Q
12:                 dist[u] = min(dist[u], dist[v] + length(v,u))
13:         return dist
14:
```

3.4.3 Đánh giá

Bây giờ ta sẽ thực thi thuật toán trong giả mã ở trên. Như đã phân tích, vòng lặp while thực hiện V1 lần lặp. Hai thao tác tốn kém nhất của mỗi lần lặp là:

- Lấy đỉnh v có giá trị d[v] nhỏ nhất ra khỏi B
- Cập nhật nhãn d[u] của các hàng xóm u của v.

Ta có thể thực thi tập hợp Q mảng. Do đó, thao tác đầu tiên có thể thực hiện trong thời gian O(V) còn thao tác thứ 2 có thể thực hiện trong thi gian O(deg(v)) trong đó deg(v) là bậc của đỉnh v trong G (nếu sử dụng cấu trúc danh sách kề). Như vậy, ta có thể thực thi thuật toán trong thời gian:

$$O(V^2 + \sum_{v \in G} deg(v)) = O(V^2 + E) = O(V^2) \tag{3.1}$$

Tuy nhiên, nếu biểu diễn B bằng một Heap nhị phân, thao tác đầu tiên chính là ExtractMin và thao tác thứ 2 là DecreaseKey. Do mỗi thao tác ExtractMin và DecreaseKey mất thời gian O(logn), thời gian để

thực hiện thuật toán Dijkstra với Heap nhị phân là:

$$O(Vlog(V) + \sum_{v \in G} log(V)deg(v)) = O((V+E)log(V)) \tag{3.2}$$

Từ đó ta suy ra: Thuật toán Dijkstra tìm đường đi ngắn nhất từ một đỉnh tới mọi đỉnh khác trong đồ thị với thời gian O((V + E)logV).

Mã giả của thuật toán khi sử dụng Heap nhị phân

Algorithm 5 Thuật toán DijkstraHeapA

```
1: procedure DIJKSTRAHEAPA(G(V,E), w, s)
2:     for v ← 1 to V do
3:         d[v] ← +∞
4:     d[s] ← 0
5:     for every neighbor v of s do
6:         d[v] ← w(s → v)
7:     B ← BuildHeap(V{s})                    ▷ B contains all vertices except s
8:     while B ≠ ∅ do
9:         u → ExtractMin(B)                  ▷ d[u] = δ(s,u)
10:        for every neighbor v of u do
11:            d[v] ← min(d[v], d[u] + w(u ← v))
12:            DecreaseKey(B, v, d[v])
    =0
```

Đánh giá

Tính đầy đủ	Chưa có tính đầy đủ. Vì thuật toán không áp dụng cho đồ thị có trọng số âm
Tính tối ưu	Tuỳ từng trường hợp. Thuật toán Dijkstra là tối ưu với đồ thị dày $(E = O(V^2))$ nhưng với đồ thị thưa thì thuật toán này khá chậm. Tuy nhiên, ý tưởng của Dijkstra sau này đã được cải tiến - sử dụng Fibonaci Heap và có thời gian chạy O(E+VlogV) .
Độ phức tạp	Trung bình là O((V+E)logV)

BẢNG 3.7: Bảng đánh giá thuật toán Dijkstra

Ngoài lề: Nếu ta biểu diễn B bằng Fibonacci Heap, thao tác ExtractMin có thời gian O(logn) (khấu trừ) và thao tác DecreaseKey có thời gian O(1). Do đó, tổng thời gian của thuật toán Dijkstra với Fibonacci Heap là:

$$O(Vlog(V) + \sum_{u \in G} log(V)deg(u)) = O((Vlog(V) + E) \tag{3.3}$$

Để in ra đường đi ngắn nhất, mỗi khi cập nhật lại giá trị $d[v] \leftarrow min(d[v], d[u] + w(u \rightarrow v))$ trong thuật toán trên, ta sẽ đánh dấu u là hàng xóm làm thay đổi nhãn d[v] của v. Ta sẽ dùng một mảng P và đánh dấu P[v]=u. Như vậy, thuật toán cuối cùng như sau:

Algorithm 6 Thuật toán DijkstraHeapB

```
1: procedure DIJKSTRAHEAPB(G(V,E), w, s)
2:     for v ← 1toV do
3:         d[v] ← +∞
4:     d[s] ← 0
```

```
5:    P[v] ← s
6:    B ← BuildHeap(V{s})                          ▷ B contains all vertices except s
7:    while B ≠ ∅ do
8:        u ← ExtractMin(B)                        ▷ d[u] = δ(s,u)
9:        for every neighbor v of u do
10:           if d[v] > d[u] + w(u → v)) then
11:               d[v] ← d[u] + w(u → v)
12:               d[v] ← u
13:           DecreaseKey(B,v,d[v])
```

Algorithm 7 FindReverseShortestPath

```
procedure FINDREVERSESHORTESTPATH(G(V,E), w, s, t)
    DijkstraHeapB(D,w,s)
    print t
    while P[t] ≠ t do
        t ← P[t]
        print t
```

Với thuật toán Dijkstra, ta vẫn sử dụng mảng H[1,2,...,n] để thực thi Heap. Tuy nhiên, mỗi phần tử H[i] sẽ lưu một đỉnh của đồ thị (chứ không phải khóa) còn khóa sẽ được lưu trong mảng d. Như vậy, khóa của nút thứ i của Heap là d[H[i]]. Để thực hiện DecreaseKey, như đã nói ở đoạn văn trước, ta sẽ lưu một mảng pos[1,2,...,V] trong đó pos[v] sẽ lưu vị trí của đỉnh vtrong Heap H. Hay nói cách khác, pos[v]=i khi và chỉ khi H[i]=v(4)

Giả mã của Heap:

Algorithm 8 DecreaseKey

```
procedure DECREASEKEY(Heap H, Vertex u, Key k)
    D[u] ← k
    UpHeapify(pos[u])
```

Algorithm 9 ExtractMin

```
procedure EXTRACTMIN(Heap H)
    n ← lengthofHeapH
    tmp ← H[1]
    H[1] ← H[n]
    pos[H[n]] ← 1                    ▷ update the position of the vertex H[n]
    DownHeapify(1)
    return tmp
=0
```

Algorithm 10 DownHeapify

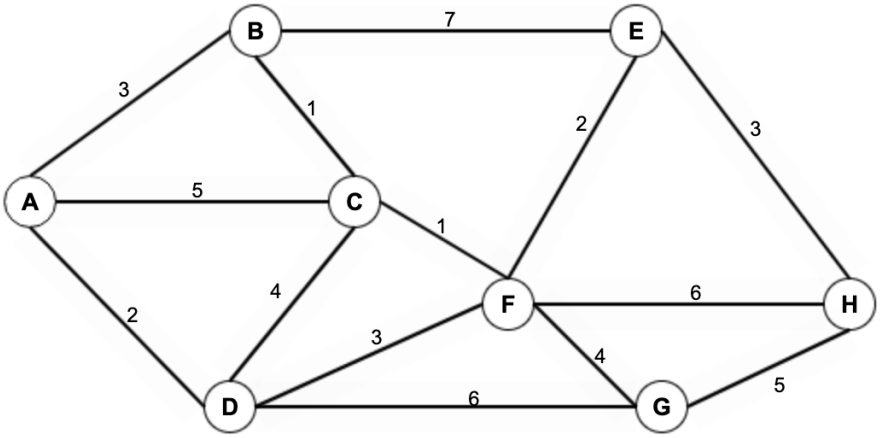
```
procedure DOWNHEAPIFY(Heap Node u)
    m ← 2u                           ▷ v is the left child of u
    if m ≤ n then                    ▷ u is not a leaf
        if d[H[m]] > d[H[2u + 1]] then
            m ← 2u + 1
```

if $d[H[u]] > d[H[m]]$ **then**
$swap(H[u], H[m])$
update pos H[u] and H[m] after swap
DownHeapify(m)

Algorithm 11 Parent

procedure PARENT(u)
if u is even **then**
return $u/2$
else
return $(u-1)/2$

3.4.4 Ví dụ



Thuật toán Dijkstra tìm đường đi ngắn nhất từ 1 đỉnh tới đỉnh bất kỳ. Ở ví dụ này, chúng tôi chọn đỉnh đầu là A, đỉnh cuối là H.

Current Node	Open Set	Close Set
	A(0, A)	∅
A	B(3A), C(5A), D(2A)	A
D	B(3A), C(5A), F(5D), G(8D)	D
B	C(4B), E(10B), F(5D), G(8D)	B
C	E(10B), F(5D), G(8D)	C
F	E(7F), G(8D), H(11F)	F
E	H(10E), G(8D)	E
G	H(10E)	G
H	∅	H

BẢNG 3.8: Mô tả cách chạy của thuật toán Dijkstra

Vậy đường đi từ A đến H là $A \to D \to F \to E \to H$ với tổng chi phí là 10.

3.5 A*

3.5.1 Giới thiệu chung

Thuật toán A Star là một thuật toán tìm kiếm được sử dụng để tìm đường dẫn ngắn nhất giữa điểm ban đầu và điểm cuối cùng.

Đây là một thuật toán tiện dụng thường được sử dụng để đi qua bản đồ để tìm con đường ngắn nhất được thực hiện. A* ban đầu được thiết kế như một bài toán truyền qua đồ thị, để giúp chế tạo một robot có thể tìm thấy hướng đi của riêng mình. Nó vẫn là một thuật toán phổ biến rộng rãi cho truyền đồ thị.

Nó tìm kiếm các đường dẫn ngắn hơn trước tiên, do đó làm cho nó trở thành một thuật toán tối ưu và hoàn chỉnh. Một thuật toán tối ưu sẽ tìm thấy kết quả chi phí thấp nhất cho một vấn đề, trong khi một thuật toán hoàn chỉnh tìm thấy tất cả các kết quả có thể có của một vấn đề.

Một khía cạnh khác làm cho A * trở nên mạnh mẽ là việc sử dụng các biểu đồ có trọng số trong quá trình triển khai nó. Biểu đồ có trọng số sử dụng các con số để biểu thị chi phí thực hiện từng đường dẫn hoặc quá trình hành động. Điều này có nghĩa là các thuật toán có thể đi theo con đường với chi phí thấp nhất và tìm ra tuyến đường tốt nhất về khoảng cách và thời gian.

3.5.2 Ý tưởng

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 được 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, nghĩa là tổng trọng số của các cạnh đã đi qua.
- h(x) là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ x. Ví dụ, nếu "chi phí"được tính là khoảng cách đã đi qua, 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 f(x) có giá trị càng thấp thì độ ưu tiên của x càng cao (do đó có thể sử dụng một cấu trúc heap tối thiểu để cài đặt hàng đợi ưu tiên này)
Chú thích:

- 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.

3.5.3 Mã giả

Algorithm 12 AStar

procedure ASTAR(G(V,E),p,q)
$f(p) = g(p) + h(p)$

$Open \leftarrow s$
$Close \leftarrow ø$
while $Open \neq \emptyset$ **do**
$Open \longleftarrow$ POP$_b est$(P) ▷ XOÁ TRẠNG THÁI (ĐỈNH) TỐT NHẤT P KHỎI OPEN
IF $p == x$ **THEN** ▷ X LÀ TRẠNG THÁI KẾT THÚC
RETURN
$Close \longleftarrow$ ADD(p)
$q \longleftarrow$ GET_NEIGHBOR(p)
12:
IF $q \in Open$ **THEN**
IF $g(q) > g(p) + Cost(p,q)$ **THEN** ▷ G(P): KHOẢNG CÁCH TỪ TRẠNG THÁI ĐẦU ĐẾN TRẠNG THÁI P
$g(q) = g(p) + Cost(p,q)$
$f(q) = g(q) + h(q)$ ▷ H(P): GIÁ TRỊ ĐƯỢC LƯỢNG GIÁ TỪ TRẠNG THÁI HIỆN TẠI ĐẾN ĐÍCH
$prev(q) = p$ ▷ P LÀ CHA CỦA Q
IF $q \notin Open$ **THEN**
$g(q) = g(p) + cost(p,q)$
$f(q) = g(q) + h(q)$
$prev(q) = p$
$Open \longleftarrow$ ADD(q)
IF $q \in Close$ **THEN**
24: **IF** $g(q) > g(p) + Cost(p,q)$ **THEN**
$Close \longleftarrow$ REMOVE(q)
$Open \longleftarrow$ ADD(q)

3.5.4 Tính toán giá trị Heuristics

Phương pháp 1

Heuristics chính xác
Mặc dù chúng ta có thể thu được các giá trị chính xác của h, nhưng làm như vậy thường mất rất nhiều thời gian. Các cách để xác định giá trị chính xác của h được liệt kê dưới đây.

- Cách 1: Trước khi sử dụng Thuật toán tìm kiếm A *, hãy tính toán trước khoảng cách giữa mỗi cặp ô.
- Cách 2: Sử dụng công thức khoảng cách / Khoảng cách Euclide, có thể trực tiếp xác định giá trị chính xác của h trong trường hợp không có các ô bị chặn hoặc vật cản.

Do các cách của phương pháp này tống nhiều thời gian. Vì thế nó thường ít được sử dụng. Ở đây, chúng tôi chỉ giới thiệu sơ lược, không đi sâu vào cách làm.

Phương pháp 2

Xấp xỉ Heuristics: Sử dụng các kỹ thuật khác nhau để xấp xỉ giá trị của h. (ít tốn thời gian hơn). Để xác định h, thường có ba heuristics xấp xỉ:

1. **Khoảng cách Manhattan:** Khoảng cách Manhattan là tổng các giá trị tuyệt đối của sự khác biệt giữa tọa độ x và y của các ô hiện tại và ô mục tiêu. Công thức được tóm tắt dưới đây

$$h = |currCell.x - goal.x| + |currCell.y - goal.y| \tag{3.4}$$

   Chúng ta phải sử dụng phương pháp heuristic này khi chúng ta chỉ được phép di chuyển theo bốn hướng trên, trái, phải và dưới.

2. **Khoảng cách đường chéo:** Nó không gì khác hơn là giá trị tuyệt đối lớn nhất của sự khác biệt giữa tọa độ x và y của ô hiện tại và ô mục tiêu. Điều này được tóm tắt dưới đây trong công thức sau

$$dx = |currCell.x - goal.x|$$
$$dy = |currCell.y - goal.y|$$
$$h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)$$

   trong đó D là độ dài của mọi nút (mặc định = 1) và D2 là độ dài đường chéo. Chúng tôi sử dụng phương pháp heuristic này khi chúng tôi chỉ được phép di chuyển theo tám hướng, giống như các nước đi của Nhà vua trong Cờ vua.

3. **Khoảng cách Euclide:** Khoảng cách Euclide là khoảng cách giữa ô mục tiêu và ô hiện tại sử dụng công thức khoảng cách:

$$h = \sqrt{(currCell.x - goal.x)^2 + (currCell.y - goal.y)^2}$$

   Chúng tôi sử dụng phương pháp heuristic này khi chúng tôi được phép di chuyển theo bất kỳ hướng nào mà chúng tôi lựa chọn.

3.5.5 Đánh giá

1. **Tính đầy đủ:** Cũng như tìm kiếm theo chiều rộng (breadth-first search), A* là thuật toán đầy đủ (complete) theo nghĩa rằng nó sẽ luôn luôn tìm thấy một lời giải nếu bài toán có lời giải. Vì A* linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic.

2. **Tính tối ưu:** A* còn có tính chất hiệu quả một cách tối ưu (optimally efficient) với mọi hàm heuristic h, có nghĩa là không có thuật toán nào cũng sử dụng hàm heuristic đó mà chỉ phải mở rộng ít nút hơn A*, trừ khi có một số lời giải chưa đầy đủ mà tại đó h dự đoán chính xác chi phí của đường đi tối ưu.

   Nếu hàm heuristic h có tính chất thu nạp được (admissible), nghĩa là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A* có tính chất thu nạp được (hay tối ưu) nếu sử dụng một tập đóng. Nếu không sử dụng tập đóng thì hàm h phải có tính chất đơn điệu (hay nhất quán) thì A* mới có tính chất tối ưu. Nghĩa là nó không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực. Phát biểu một cách hình thức, với mọi nút

x, y trong đó y là nút tiếp theo của x:

$$h(x) <= g(y) - g(x) + h(y)$$

3. **Độ phức tạp thời gian:** Độ phức tạp về thời gian của A* phụ thuộc vào phương pháp heuristic. Trong trường hợp xấu nhất của không gian tìm kiếm không giới hạn, số lượng nút được mở rộng là theo cấp số nhân ở độ sâu của giải pháp (đường dẫn ngắn nhất) d: $O(b^d)$, trong đó b là hệ số phân nhánh (số lượng người kế nhiệm trung bình trên mỗi trạng thái). Điều này giả định rằng một trạng thái mục tiêu tồn tại ở tất cả, và có thể đạt được từ trạng thái bắt đầu; nếu không, không gian trạng thái là vô hạn, thuật toán sẽ không kết thúc.

   Chức năng heuristic có ảnh hưởng lớn đến hiệu suất thực tế của tìm kiếm A *, vì heuristic tốt cho phép A* cắt bớt nhiều nút bd mà một tìm kiếm không hiểu biết sẽ mở rộng. Chất lượng của nó có thể được thể hiện dưới dạng yếu tố phân nhánh hiệu quả b*, có thể được xác định theo kinh nghiệm cho một trường hợp vấn đề bằng cách đo số lượng nút được tạo ra bởi sự mở rộng, N, và độ sâu của giải pháp, sau đó giải quyết

$$N + 1 = 1 + b* + b*^2 + ... + b*^d$$

   Heuristics tốt là những người có hệ số phân nhánh hiệu quả thấp (tối ưu là b* = 1).

   Độ phức tạp thời gian là đa thức khi không gian tìm kiếm là một cây, có một trạng thái mục tiêu duy nhất và hàm heuristic h đáp ứng điều kiện sau:

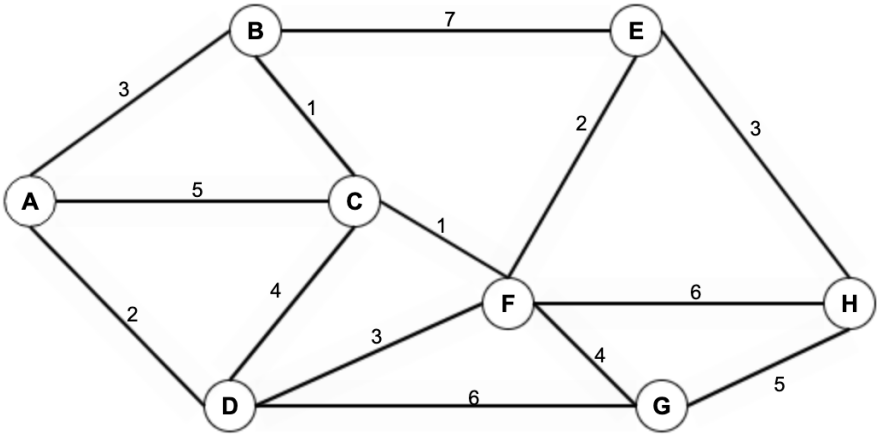
$$|h(x) - h*(x)| \leq O(logh*(x))$$

   trong đó h* là phương pháp heuristic tối ưu, chi phí chính xác để đi từxđến mục tiêu. Nói cách khác, sai số củah sẽ không phát triển nhanh hơn logarit của "heuristic hoàn hảo"h* trả về khoảng cách thực từ x đến mục tiêu.

4. **Độ phức tạp không gian:** gần giống như của tất cả các thuật toán tìm kiếm đồ thị khác, vì nó giữ tất cả các nút được tạo trong bộ nhớ. 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. Trong thực tế, đây hóa ra là nhược điểm lớn nhất của tìm kiếm A*, dẫn đến sự phát triển của các tìm kiếm 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*).

3.5.6 Ví dụ

Cho đồ thị như hình vẽ:



Thuật toán A* tìm đường đi ngắn nhất từ điểm đầu A tới điểm đích H.
Trong ví dụ minh hoạ này, để xác định h, chúng tôi chọn heuristics xấp xỉ theo khoảng cách Manhattan. Công thức được tóm tắt dưới đây

$$h = |currCell.x - goal.x| + |currCell.y - goal.y|$$

Điểm	Toạ độ	Heuristics Cost
A	(0; 5.5)	14
B	(3.5; 8)	13
C	(5; 5.5)	9
D	(3; 2.5)	11
E	(9; 8)	7.5
F	(7; 4)	5.5
G	(9.5; 2.5)	4.5
H	(12.5; 4)	0

BẢNG 3.9: Ước lượng khoảng cách Heuristics từ một điểm đến đích

Vậy đường đi từ A đến H là $A \to D \to F \to E \to H$ với tổng chi phí là 10.

Ví dụ:

- $A \to B : f(A \to B) = g(A \to B) + h(B \to H) = 3 + 13 = 16$
- $A \to D \to F : f(A \to D \to F) = g(A \to D \to F) + h(F \to H)$
  $= g(A \to D) + g(D \to F) + h(F \to H) = 2 + 3 + 5.5 = 10.5$

Current Node	Open Set	Close Set
	A(0, A)	∅
A	B(16, A), C(14, A), D(13, A)	A
D	C(14, A), F(10.5, D), G(12.5, D)	D
F	C(14, A), E(14.5, F), G(12.5, D), H(11, F)	F

H	∅	H

BẢNG 3.10: Mô tả cách chạy của thuật toán A Star

Chương 4

So sánh các thuật toán

4.1 BFS và DFS

Thuật toán / Cơ sở so sánh	BFS	DFS
Phương pháp tìm kiếm	Tìm kiếm theo chiều rộng. Với mỗi đỉnh thì BFS lại duyệt hết các đỉnh kề với đỉnh đó, sau đó lại tiếp tục duyệt toàn bộ các đỉnh kề với đỉnh tiếp theo.	Tìm kiếm theo chiều sâu. Ở mỗi nhánh của đồ thị, nó sẽ duyệt từng cạnh kề với đỉnh gốc và đi sâu xuống tới khi không còn cạnh kề nữa.
Căn bản	Thuật toán dựa trên đỉnh	Thuật toán dựa trên cạnh
Cấu trúc cây được xây dựng	Rộng và ngắn	Hẹp và dài
Trường hợp tốt nhất	Vét cạn toàn bộ	- Phương án chọn hướng đi tuyệt đối chính xác. - Phương án chọn hướng đi tuyệt đối chính xác.
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ
Tính hiệu quả	- Hiệu quả khi lời giải nằm gần gốc của cây tìm kiếm. - Hiệu quả của chiến lược phụ thuộc vào độ sâu của lời giải. - Lời giải thích càng xa gốc thì hiệu quả của chiến lược càng giảm. - Lời giải thích càng xa gốc thì hiệu quả của chiến lược càng giảm.	-Hiệu quả khi lời giải nằm sâu trong cây tìm kiếm và có một phương án chọn hướng đi chính xác. - Hiệu quả của chiến lược phụ thuộc vào phương án chọn hướng đi - Phương án càng kém hiệu quả thì hiệu quả của chiến lược càng giảm. - Thuận lợi khi chỉ muốn tìm một lời giải thích.
Độ phức tạp	O(|E| + |V|)	O(|E| + |V|)
Thời gian đi qua	Các đỉnh không mong muốn cũ nhất được khám phá lúc đầu	Các đỉnh dọc theo các cạnh được khám phá ngay từ đầu

Dung lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Không hiệu quả. Vì BFS phải lưu toàn bộ các trạng thái.	Hiệu quả. Vì DFS không chiếm dụng thời gian tuyến tính nên chúng tôi chỉ lưu lại các trạng thái chưa được kiểm tra.
Tối ưu	Tối ưu cho việc tìm kiếm khoảng cách ngắn nhất, không phải tốn nhiều chi phí	Mang lại giải pháp sâu hơn, nhưng không tối ưu, nhưng nó hoạt động tốt khi giải pháp dày đặc
Ứng dụng	Kiểm tra biểu đồ lưỡng cực, thành phần được kết nối và đường dẫn ngắn nhất có trong biểu đồ	Kiểm tra đồ thị được kết nối 2 Cạnh, đồ thị được kết nối mạnh, đồ thị theo chu kỳ và thứ tự tôpô

BẢNG 4.1: Bảng so sánh sự khác nhau giữ thuật toán BFS và DFS

4.2 UCS và Dijsktra

Thuật toán / **Cơ sở so sánh**	**UCS**	**Dijkstra**
Phương pháp tìm kiếm	Tìm đường đi có chi phí thấp nhất từ đỉnh đến đích	Ttìm đường đi có chi phí thấp nhất từ đỉnh đến tất cả các đỉnh khác trên đồ thị
Cơ chế	Hàng đợi ưu tiên (priority queue)	Hàng đợi ưu tiên (priority queue)
Cấu trúc thường được xây dựng	UCS có thể áp dụng cho cả đồ thị tường minh và đồ thị ẩn (nơi các trạng thái/nút được tạo ra).	Djikstra chỉ được áp dụng trong các đồ thị tường minh khi biết tất cả các đỉnh và các cạnh.
Nhược điểm	Nó không quan tâm đến số bước liên quan đến việc tìm kiếm và chỉ quan tâm đến chi phí đường dẫn. Do đó thuật toán này có thể bị mắc kẹt trong một vòng lặp vô hạn.	Không áp dụng cho đồ thị có trọng số âm
Trường hợp xấu nhất	Vét cạn toàn bộ	Vét cạn toàn bộ
Độ phức tạp	O(|E| + |V|)	O((|E| + |V|)log|V|)
Thời gian chạy	Nhanh do yêu cầu về bộ nhớ thấp	Chậm do yêu cầu về bộ nhớ nhiều
Dung lượng bộ nhớ sử dụng để lưu trữ các trạng thái	Hiệu quả. Vì UCS chỉ lưu từ đỉnh bắt đầu đến khi tìm được đỉnh kết thúc	Không hiệu quả. Vì Dijkstra phải lưu toàn bộ đỉnh của đồ thị.

Tối ưu	Tối ưu cho việc tìm kiếm khoảng cách ngắn nhất, không phải tốn nhiều chi phí	Tuỳ từng trường hợp. Thuật toán Dijkstra là tối ưu với đồ thị dày ($E = O(V^2)$) nhưng với đồ thị thưa thì thuật toán này khá chậm. Tuy nhiên, ý tưởng của Dijkstra sau này đã được cải tiến - sử dụng Fibonaci Heap và có thời gian chạy O(E+VlogV)

BẢNG 4.2: Bảng so sánh sự khác nhau giữ thuật toán UCS và Dijkstra

Chương 5

Thực nghiệm

5.1 Mô tả cài đặt

Sử dụng ngôn ngữ Python, phiên bản 3.9.12 với các thư viện được đính kèm trong requirements.txt

Cài đặt 4 Thuật toán BFS, DFS, UCS, ASTAR có đỉnh bắt đầu là 71 và đỉnh kết thúc là 318.

Trong thực nghiệm này, chi phí đi từ node này đến node khác là khoảng cách Euclid giữa 2 node. Do đó, chi phí đi đến các node up, down, left, right là như nhau và chi phí đi đến các node up_left, up_right, down_left, down_right là như nhau.

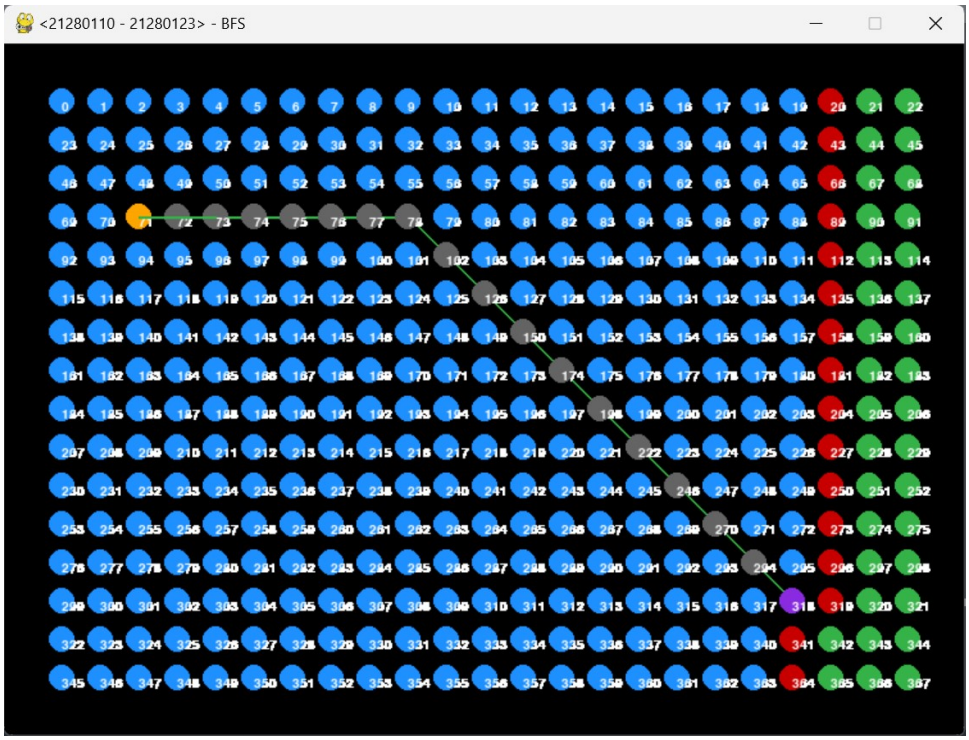
Heuristic của một node là khoảng cách Euclid từ node đó đến đích.

Giải thích code

- File Space.py định nghĩa một không gian tìm kiếm. Không gian này là một đồ thị có dạng như hình vẽ và chứa class Node , class Graph :
  - class Node : Định nghĩa đối tượng node và các hàm bổ trợ.
  - class Graph : Định nghĩa đối tượng graph và các hàm bổ trợ.
- File SearchAlgorithm.py : Nơi cài đặt các thuật toán tìm kiếm.
  - open_set : Tập mở, chứa các node được mở rộng đến.
  - closed_set : Tập đóng, chứa các mode đã được xét đến.
  - father : Từ node x mở rộng được node y thì father[y] = x .
  - cost : Chi phí đi từ trạng thái đầu đến node x là y thì cost[x] = y .
- File Constants.py : Chứa các hằng số.
- File main.py : Các bạn gọi file này để thực thi chương trình.

5.2 Kết quả, nhận xét

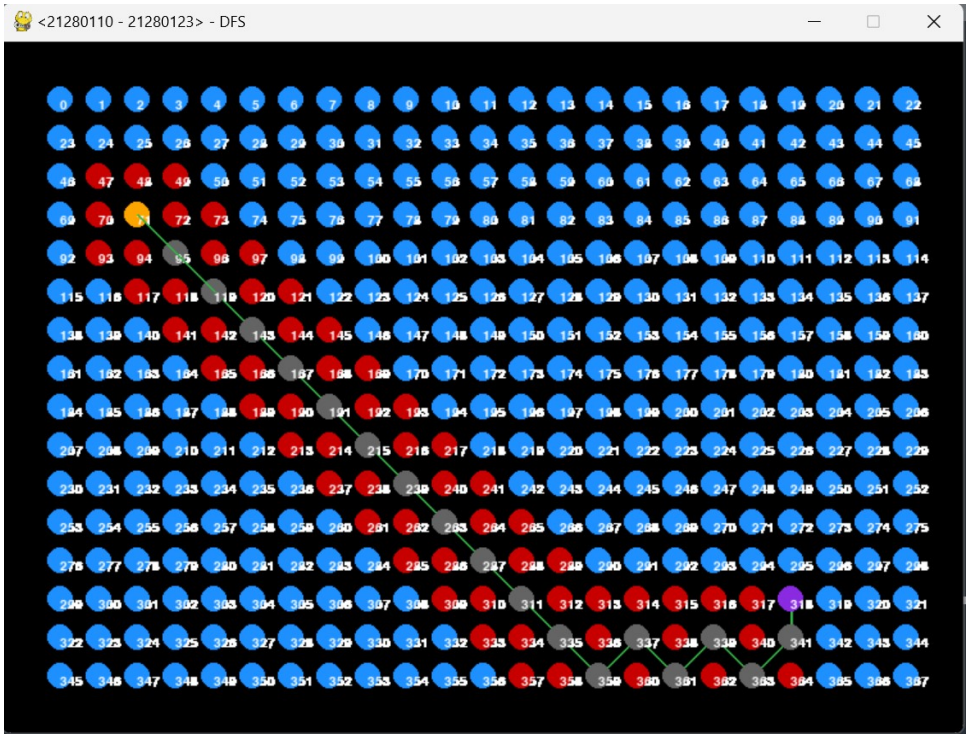
5.2.1 BFS



Thuật toán BFS sẽ ưu tiên tìm kiếm theo chiều rộng, duyệt đến các node có cùng độ sâu trước. Từ hình trên, ta thấy rằng vùng tìm kiếm của BFS có dạng hình vuông (hình chữ nhật)

Thuật toán BFS có thời gian tìm kiếm khá dài vì nó gần như phải duyệt qua toàn bộ ma trận để tìm được đường đi tới ô đích do xét hết tất cả các nút lân cận với nó trước. Tuy nhiên, nó đưa ra được đường đi ngắn nhất đến node đích.

5.2.2 DFS



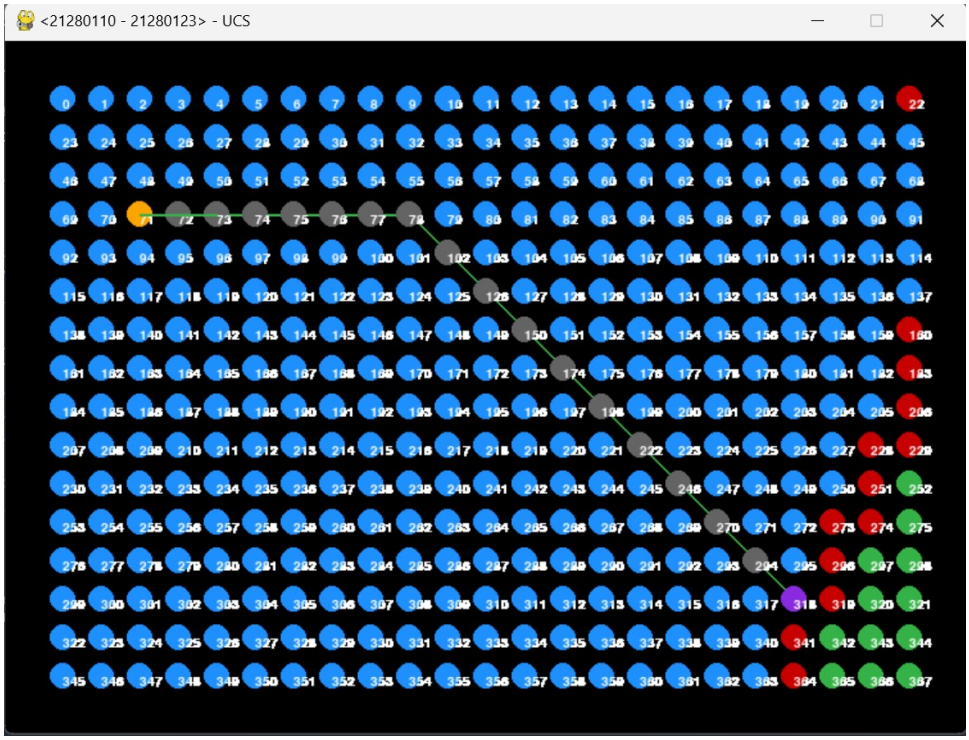
Thuật toán DFS sẽ ưu tiên tìm kiếm theo chiều sâu, hướng sâu xuống các node con của node vừa duyệt.

DFS có thể tìm ra một đường đi tới đích nhanh chóng hơn BFS nhưng đường đi thường dài và không tối ưu tốt bằng bfs.

Thuật toán DFS có thời gian tìm kiếm khá dài nếu node cần tìm không nằm trong khoảng thuận lợi trên đường duyệt của DFS. Từ hình vẽ, ta thấy rằng node 318 không nằm trong khoảng thuận lợi tính từ node 71 nên DFS hầu như phải vét cạn toàn bộ các node trong đồ thị.

Hầu hết các trường hợp, DFS không cho ra được đường đi tối ưu.

5.2.3 UCS

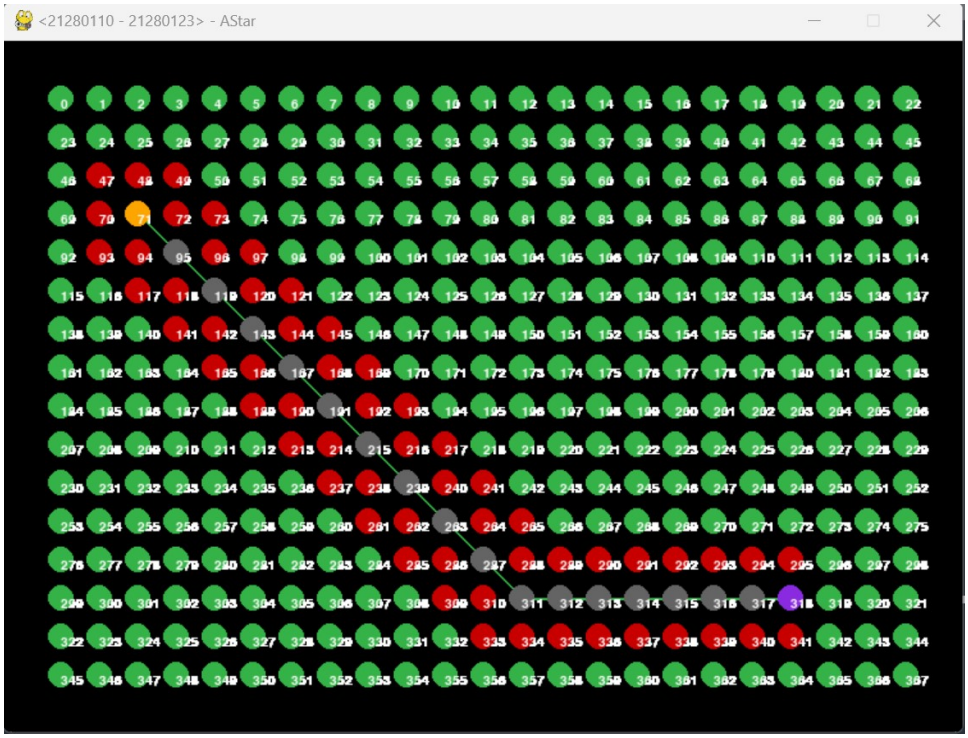


Thuật toán UCS sẽ ưu tiên tìm kiếm dựa theo chi phí, duyệt đến các node có chi phí nhỏ nhất từ node bắt đầu đến node đó. Từ hình trên, ta thấy rằng vùng tìm kiếm của UCS có dạng hình tròn.

Mặc dù UCS thích hợp làm việc với các bản đồ có trọng số, còn BFS thì không nhất thiết bản đồ phải có trọng số nhưng kết quả giống nhau là do UCS dựa vào trọng số để quyết định nước đi tiếp theo, tuy nhiên trọng số của 4 hướng di chuyển đều bằng 1, do đó quyết định nút mở theo cấu trúc lưu trữ là hàng đợi, nên cách lựa chọn đường đi có phần giống BFS.

Thuật toán UCS có thời gian tìm kiếm khá dài vì phải mở rộng dến nhiều node. Tuy nhiên, nó đưa ra được đường đi có chi phí nhỏ nhất từ node bắt đầu đến node đích.

5.2.4 AStar



Từ hình trên, ta thấy rằng vùng tìm kiếm của A* mở rộng trực tiếp đến node cần tìm.

Do việc A* tìm đường đi tiếp theo dựa vào cả trọng số và heuristic nhưng không cộng dồn heuristic qua các nước đi mà chỉ xét heuristic của nước đi kế tiếp, do đó heuristic ở điểm đích nếu tính dựa vào tọa độ của 2 điểm trong không gian sẽ luôn là 0. Nên chi phí đường đi là bằng nhau với những giá trị trọng số không đổi.

Thuật toán A* có thời gian tìm kiếm ngắn và số node phải duyệt cũng ít hơn rất nhiều so với các thuật toán trước. Nó đưa ra được đường đi tối ưu về chi phí cũng như khoảng cách Euclid từ node đó đến đích.

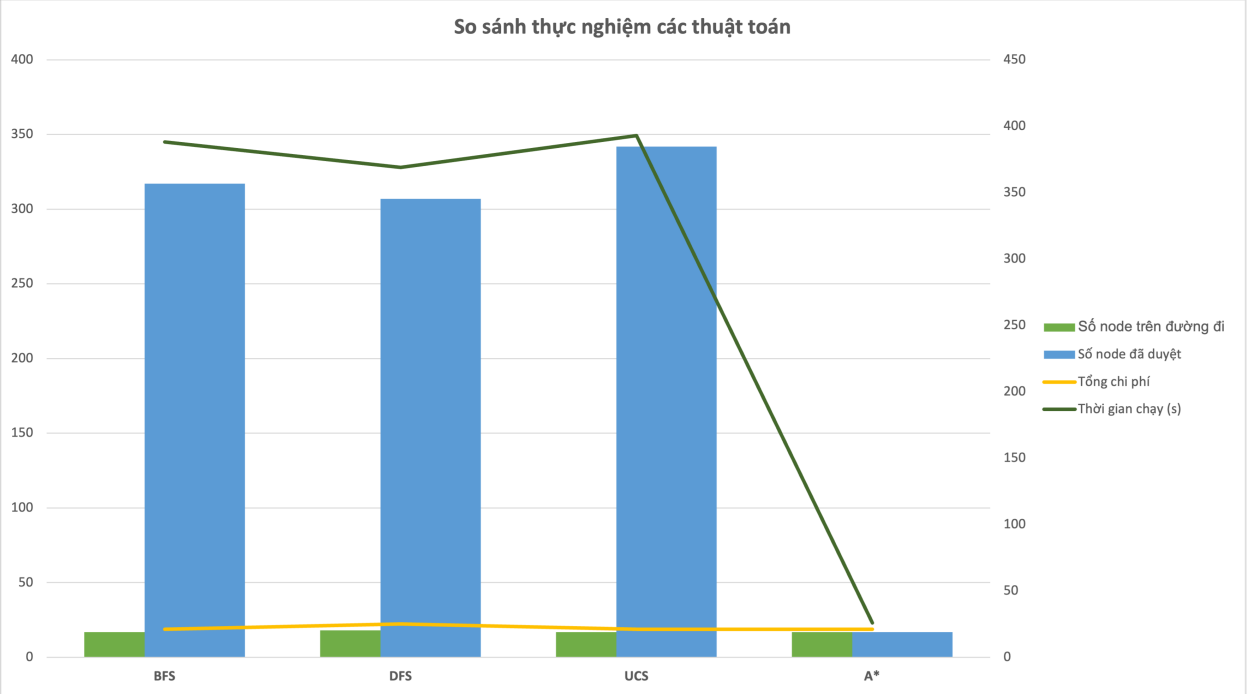
5.2.5 Nhận xét chung

Do phong cách lập trình của mỗi người và độ mạnh yếu của máy tính là khác nhau. Nên nhận xét dưới đây không mang tính tổng quát chung. Nhận xét dựa theo các kết quả của lần chạy thứ 10.

BẢNG 5.1: Bảng so sánh chi phí các thuật toán khi chạy

Tiêu chí / Thuật toán	Số node trên đường đi	Số node đã duyệt	Tổng chi phí	Thời gian chạy (s)
BFS	18	317	$7+10\sqrt{2}$	388
DFS	19	307	$1+17\sqrt{2}$	369
UCS	18	342	$7+10\sqrt{2}$	393
AStar	18	17	$7+10\sqrt{2}$	26

Biểu đồ dưới đây giúp mọi người hình dung rõ hơn sự khác nhau của các thuật toán



HÌNH 5.1: Biểu đồ so sánh kết quả thực nghiệm các thuật toán

Nhận xét:

Trong suốt quá trình chạy, các thuật toán BFS, DFS, UCS gần như duyệt hết đồ thị. Nên có số node đã duyệt là khá nhiều, trung bình 322 node - một con số khá lớn. Riêng với thuật toán AStar, số node đã duyệt là 17, ít gần 18 lần so với các thuật toán trên.

Số node trên đường đi của các thuật toán gần như là bằng nhau, bằng 17 node. Riêng thuật toán DFS có số node là 18

Tương tự, tổng chi phí của BFS, UCS, AStar là $7 + 10\sqrt{2}$, DFS là $1 + 17\sqrt{2}$. Lớn hơn ≈ 3.8995.

AStar có tốc độ tìm đường đi tới đích là nhanh nhất, khoảng 26s. Trong khi các thuật toán khác phải tốn khá nhiều thời gian. Thời gian chạy gần như gấp 44 lân so với AStar.

Dù DFS có tổng chi phí và số lượng node trên đường đi đều lớn hơn BFS và UCS. Nhưng tốc độ tìm ra đường đi đến đích của DFS luôn nhanh hơn, nó gần như tìm ra một đường đi hợp lệ tới đích sau vài lần thử đầu tiên. Tuy nhiên, đường đi thu được lại là một đường đi rất dài.

Chương 6

Kết luận

Lời kết luận

Không có thuật toán tìm kiếm nào là tốt nhất trong mọi trường hợp, lựa chọn nào cũng sẽ có sự đánh đổi, nếu chọn thời gian thì sẽ mất nhiều bộ nhớ và ngược lại. Đồng thời, kiểu dữ liệu, tình trạng, phân bố dữ liệu cũng là các tác nhân ảnh hưởng đến độ hiệu quả của thuật toán. Chính vì vậy, tùy thuộc vào từng bộ dữ liệu đầu vào, không gian bộ nhớ cho phép, tốc độ thực thi mong muốn mà đưa ra lựa chọn thuật toán cho phù hợp.

Việc tìm hiểu chi tiết, kỹ càng về các thuật toán sẽ giúp ta có những hình dung đúng đắn về cách hoạt động và có thể từ đó có những cải tiến, kết hợp để tạo ra những thuật toán phù hợp hơn với nhu cầu, bài toán riêng mà ta cần giải quyết.

Đồ án **"Tìm hiểu về các thuật toán tìm kiếm trên đồ thị"** là một đồ án nhằm tìm hiểu sâu hơn về ý tưởng, cách cài đặt, visualize của các thuật toán thuộc nhóm thuật toán tìm kiếm trên. Thông qua đồ án, các thành viên trong nhóm có được những kinh nghiệm, bài học nhất định về các thuật toán tìm kiếm, sắp xếp cũng như kỹ năng làm việc nhóm.

Một lần nữa, các thành viên trong nhóm xin cám ơn giảng viên đã cung cấp cho chúng em cơ hội thực hiện đồ án này, tạo cơ hội để chúng em thực hành làm việc nhóm nhiều hơn.

Trân trọng!

Các thành viên nhóm

Phụ lục A

Tài liệu tham khảo

Frank M. Carrano and Timothy Henry (2014), *Walls and Mirrors: Data Abstraction And Problem Solving with C++*, University of Rhode Island, pp. 603-629.

Geeksforgeeks (08/12/2022), *Graph Data Structure And Algorithms*, URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/?ref=ghm, (12/2022)

HelpEx (12/02/2019), *Từ Dijkstra đến A Star (A *), Phần 2: Thuật toán AStar (A *)*, https://helpex.vn/article/tu-dijkstra-den-a-star-a-phan-2-thuat-toan-a-star-a-5c6b287fae03f628d053c721, (11/2022)

Lê Minh Hoàng (1999-2002), *Giải thuật và lập trình*, Đại học Sư phạm Hà Nội, pp. 177-189.

Simplilearn (15/11/2022), *A* Algorithm in Artificial Intelligence You Must Know in 2023*, URL: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm, (12/2022)