TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo

Description automatically generated

**BÀI TẬP GIỮA KỲ**

**MÔN NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

*Người thực hiện:* **NGUYỄN QUỐC CƯỜNG – 518H0003**

*Lớp*: **18H50203**

*Khoá*: **22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo

Description automatically generated

**BÀI TẬP GIỮA KỲ**

**MÔN NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

*Người thực hiện:* **NGUYỄN QUỐC CƯỜNG – 518H0003**

*Lớp*: **18H50203**

*Khoá*: **22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2021**

PHẦN ĐÁNH GIÁ CỦA GIẢNG VIÊN

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**MỤC LỤC**

[I. SỐ ĐẢO 5](#_Toc85296138)

[1. Mô tả cấu trúc dữ liệu: 5](#_Toc85296139)

[2. Giải thuật BFS – Breadth First Search: 6](#_Toc85296140)

[1. Sơ đồ giải thuật: 6](#_Toc85296141)

[2. Hiện thực: 7](#_Toc85296142)

[3. Kết quả và thảo luận: 8](#_Toc85296143)

[3. Giải thuật UCS – Uniform Cost Search: 9](#_Toc85296144)

[1. Sơ đồ giải thuật: 9](#_Toc85296145)

[2. Hiện thực: 10](#_Toc85296146)

[3. Kết quả và thảo luận: 11](#_Toc85296147)

[II. DU LỊCH Ở ROMANIA 12](#_Toc85296148)

[1. Mô tả cấu trúc dữ liệu: 12](#_Toc85296149)

[2. Giải thuật Greedy Best First Search (Heuristic): 13](#_Toc85296150)

[1. Sơ đồ giải thuật: 13](#_Toc85296151)

[2. Hiện thực: 14](#_Toc85296152)

[3. Kết quả và thảo luận: 15](#_Toc85296153)

[3. Giải thuật A\*: 16](#_Toc85296154)

[1. Sơ đồ giải thuật: 16](#_Toc85296155)

[2. Hiện thực: 17](#_Toc85296156)

[3. Kết quả và thảo luận: 18](#_Toc85296157)

[III. TỰ ĐÁNH GIÁ 19](#_Toc85296158)

[IV. TÀI LIỆU THAM KHẢO 20](#_Toc85296159)

# SỐ ĐẢO

## Mô tả cấu trúc dữ liệu:

* A picture containing kitchenware, grater

  Description automatically generatedA picture containing shoji, crossword puzzle, building

  Description automatically generatedDữ liệu đầu vào: Dữ liệu đầu vào được lưu dưới dạng ma trận gồm 10 dòng và 10 cột (số dòng và cột có thể thay đổi). Ở mỗi ô của ma trận, số 1 sẽ tượng trưng cho đất liền và số 0 sẽ tượng trung cho biển. Dựa vào những điều trên dữ liệu của chúng ta được chuyển đổi như sau.
* A picture containing text, electronics, screenshot

  Description automatically generatedQueue: Queue là một kiểu cấu trúc dữ liệu phổ biến trong ngôn ngữ lập trình. Nó tuân thủ nguyên tắc First In First Out (FIFO) – vào trước ra trước.

Ở trong bài này, queue được triển khai bằng mãng và ta chỉ sử dụng 3 chức năng chính của queue:

* + append(): thêm một phần tử vào cuối hàng chờ.
  + pop(0): lấy đi phần tử đầu tiên trong hàng chờ.
  + is Queue: dùng để kiểm tra xem hàng chờ có phần tử nào hay không.
* PriorityQueue: PriorityQueue là một loại hàng chờ đặc biệt, mỗi phần tử trong đó được gắn với một giá trị nhất định để xét độ ưu tiên. Cũng tương tự như Queue ở phần thêm dữ liệu và kiểm tra hàng chờ có rỗng hay không, nhưng ở phần lấy giá trị thì PriorityQueue sẽ lấy giá trị có độ ưu tiên cao nhất. Trong bài này đó chính là hướng đi ngắn nhất.

## Giải thuật BFS – Breadth First Search:

### Diagram Description automatically generatedSơ đồ giải thuật:

### Hiện thực:

* Bread First Search (BFS): Là giải thuật tìm kiếm theo chiều rộng của đồ thị mà không cần quan tâm đến trọng số, hướng đi, và các giá trị được duyệt. BFS sẽ bắt đầu từ một điểm cho trước (0, 0) sau đó nó sẽ mở rộng ra các hướng liền kề với đỉnh đã cho. Và quá trình này được lập lại cho đến khi duyêt qua hết đồ thị.
* Bước thực hiện:
  + Đầu tiên chúng ta sẽ kiểm tra xem dữ liệu đầu vào có hay không.
  + Kế đến ta cần phải đếm số dòng và sốt cột của ma trận.
  + Xác định các hướng mà BFS sẽ duyệt. Bao gồm 8 hướng: chéo trên bên trái, bên trên, chéo trên bên phải, bên phải, chép dưới bên phải, bên dưới, chéo dưới bên trái và bên trái.
  + Khởi tạo biến đếm và queue.
  + Lần lượt duyệt qua hết ma trận, vị trí bắt đầu là (0, 0). Sau đó sẽ kiểm tra xem ô được duyệt có bằng 1 hay không (đảo). Nếu có thì sẽ tăng biến đếm lên, thêm toạ độ của ô đó vào queue và gán giá trị của ô đó bằng 0 để tránh bị duyệt trùng.
    - Tiếp theo, chúng ta sẽ kiểm tra queue có rỗng hay không nếu không thì sẽ lấy giá trị đầu tiên của queue. Ta sẽ duyệt lần lượt theo các hướng đã xác định bên trên, nếu ô được duyệt là đảo thì sẽ thêm vào hàng chờ và gán giá trị bằng không.
    - Khi hàng chờ rỗng sẽ tiếp tục duyệt ma trận theo vòng lặp trên.
* A close-up of a document

  Description automatically generated with medium confidenceCode:

### Kết quả và thảo luận:

* Text

  Description automatically generated with medium confidenceKết quả khi chạy thử chương trình với giá trị đầu vào như trên:
* Độ phức tạp theo thời gian của BFS: O (bd)
  + Trong đó b là ô dang được duyêt và d là các hướng từ ô đó có thể di chuyển

## Giải thuật UCS – Uniform Cost Search:

### Diagram Description automatically generatedSơ đồ giải thuật:

### Hiện thực:

* Uniform Cost Search (UCS): Là giải thuật tìm kiếm với chi phí cực tiểu. UCS sẽ bắt đầu từ một điểm cho trước (0, 0) sau đó nó sẽ mở rộng ra các hướng liền kề với đỉnh đã cho. Trong quá trình trên, thuật toán sẽ chọn ra những hướng có khoảng cách ngắn nhất để đi trước, khi các hướng đều có chi phí như nhau thì thuật toán sẽ áp dụng như BFS. Và quá trình này được lập lại cho đến khi duyêt qua hết đồ thị.
* Bước thực hiện:
  + Đầu tiên chúng ta sẽ kiểm tra xem dữ liệu đầu vào có hay không.
  + Kế đến ta cần phải đếm số dòng và sốt cột của ma trận.
  + Khởi tạo biến đếm và queue.
  + Lần lượt duyệt qua hết ma trận, vị trí bắt đầu là (0, 0). Sau đó sẽ kiểm tra xem ô được duyệt có bằng 1 hay không (đảo). Nếu có thì sẽ tăng biến đếm lên, thêm toạ độ của ô đó vào queue và gán giá trị của ô đó bằng 0 để tránh bị duyệt trùng.
    - Tiếp theo, chúng ta sẽ kiểm tra queue có rỗng hay không nếu không thì sẽ lấy giá trị đầu tiên của queue.
    - Xác định các hướng mà UCS sẽ duyệt. Bao gồm 8 hướng: chéo trên bên trái, bên trên, chéo trên bên phải, bên phải, chép dưới bên phải, bên dưới, chéo dưới bên trái và bên trái. Nhưng lần này sẽ khác với BFS là sẽ có thêm khoảng cách, khoảng cách của các đường chéo là và đường thẳng là 1.
    - Ta sẽ duyệt lần lượt theo các hướng đã xác định bên trên theo thứ tự khoảng cách từ thấp tơi cao, nếu ô được duyệt là đảo thì sẽ thêm vào hàng chờ và gán giá trị bằng không.
    - Khi hàng chờ rỗng sẽ tiếp tục duyệt ma trận theo vòng lặp trên.
* Text

  Description automatically generatedCode:

### Kết quả và thảo luận:

* Text, letter

  Description automatically generatedKết quả khi chạy thử chương trình với giá trị đầu vào như trên:
* Độ phức tạp theo thời gian của UCS: O (b1 + [C\*/ε])
  + Trong đó b là ô đang được duyệt, C\* là chi phí giải pháp tối ưu, và ε là từng bước để đến gần với ô cuối.

# DU LỊCH Ở ROMANIA

## Mô tả cấu trúc dữ liệu:

* Diagram, radar chart

  Description automatically generatedDữ liệu đầu vào: Dữ liệu đầu vào bao gồm 2 phần được lưu dưới dạng dictionary là mối liên hệ giữa các nút và heuristic. Kiểu dictionary sẽ lưu các phần tử dưới dạng key value.

Như hình trên, dữ liệu sẽ được lưu trữ như sau, ta chọn Arad là key và các nút kết nối với Arad sẽ là các value. Trong value sẽ bao gồm tên của nút kết nối với key và khoảng cách từ key đến value.

Dict = {'Arad': [('Sibiu', 140), ('Zerind', 75), ('Timisoara', 118)]}

Tương tự với heuristic, ta có key là tên thành phố và value là khoảng cách từ thành phố đó đến Bucharest.

* Queue: Ở phần II, cấu trúc Queue ta sử dụng sẽ giống tương tự như phần I.
* MyPriorityQueue: Về cơ bản, MyPriorityQueue cũng tương tự như PriorityQueue của phần I, nhưng đã được thêm một vài chức năng để thuận tiện hơn trong quá trình áp dụng với giải thuật.

Các chức năng chính:

* + isEmpty(): Kiểm tra xem hàng chờ có rỗng hay không, kết quả sẽ trả về True hoặc False.
  + add(): Thêm phần tử vào hàng chờ.
  + pop(): Lấy phần tử có trọng số thấp nhất ra khỏi hàng chờ.
  + is\_in(): Kiểm tra phần tử có tồn tại trong hàng chờ hay không, nếu có sẽ trả về giá trị tồn tại trong hàng chờ và ngược lại sẽ trả về 0.
  + replace(): Dùng để thay thế giá trị đã tồn tại trong hàng chờ bằng cách xoá bỏ giá trị cũ và thêm vào giá trị mới.
  + delete(): Xoá bỏ giá trị ở hàng chờ theo tham số truyền vào.

## Giải thuật Greedy Best First Search (Heuristic):

### Sơ đồ giải thuật:

A picture containing diagram

Description automatically generated

### Hiện thực:

* Greedy Best First Search (GBFS): Greedy Best First Search là sử kết hợp giữa BFS và DFS, thuật toán sử dụng hàm đám giá (heuristic) để luôn chọn hướng đi tối ưu nhất ở thời điểm đó. Ở trong bài này, GBFS sẽ bắt đầu từ một điểm cho trước và tìm đường đi ngắn nhất để đến đích dựa vào bảng khoảng cách đường chim bay từ các điểm tới điểm đích. Ví dụ ta chọn Arad là điểm bắt đầu và Bucharast là điểm đích.

Diagram

Description automatically generatedTa sẽ chọn được điểm kế tiếp là Sibiu do khoảng cách từ Sibiu đến đích là ngắn nhất. Và tương tự với các điểm sau, ta sẽ đến được đích.

* Các bước thực hiện:
  + Đầu tiên ta sẽ kiểm tra xem có tồn tại dữ liệu đầu vào hay không, nếu không sẽ trả về 0.
  + Kế đến sẽ khởi tạo các mảng path, queue, và thêm giá trị bắt đầu vào queue.
  + Sử dụng while để kiểm tra xem queue có giá trị hay không.
    - Gán current bằng giá trị đầu tiên trong hàng chờ queue. Và thêm vào path.
    - Kiểm tra nếu current bằng với điểm đích thì sẽ trả về path.
    - Nếu không, khởi tạo MyPriorityQueue, và lần lượt thêm vào các điểm con của current cùng với khoảng cách từ điểm con đến đích được cho sẵn.
    - Sau cùng ta sẽ thêm vào queue phần từ có trọng số nhỏ nhất trong hàng chờ ưu tiên. Và lập lại vòng lập này cho đến khi đến đích.
* Text

  Description automatically generated with medium confidenceCode:

### Kết quả và thảo luận:

* Text

  Description automatically generated with medium confidenceKết quả khi chạy thử chương trình khi chọn điểm bắt đầu là Arad:
* Độ phức tạp theo thời gian: O(bm)
  + Trong đó b là điểm đang được duyệt và m là số lượng điểm con của b.
* Điểm yếu: GBFS có thể sẽ bị vướng vào các vòng lập vô tận như sau:

Isai – Neamt – Isai – Neamt ...

## Giải thuật A\*:

### Diagram Description automatically generatedSơ đồ giải thuật:

### Hiện thực:

* Diagram

  Description automatically generatedA\*: Giải thuật A\* là giải thuật phổ biến nhất của Best First Search, và nó sử dụng hàm đánh giá heuristic h(n) và chi phí (khoảng cách) từ điểm bắt đầu đến điểm hiện tại g(n). A\* là sự kết hợp của GBFS và UCS nên nó được tối ưu hơn so với các giải thuật khác. Khi mà chúng ta kết hợp giữa heuristic và chi phí, sẽ được f(n) = h(n) + g(n), từ đó A\* sẽ quyết định điểm kế tiếp dựa trên f(n).
* Các bước thực hiện:
  + Đầu tiên ta sẽ kiểm tra dữ liệu đầu vào và khởi tạo 2 biến open và close dưới dạng MyPriorityQueue, open dùng để chứa các điểm đang được xét duyệt và close để chứa các điểm đã được xét.
  + Đồng thời ta cũng tạo 2 dictionary là g để lưu trữ khoảng các từ điểm bắt đầu cho đến hiện tại và prev để lưu điểm cha của điểm đó. Ta sẽ gán điểm bắt đầu bằng với 0 và thêm điểm bắt đầu cùng g(n) vào open.
  + Kế đến, sẽ kiểm tra xem open có rỗng hay không. Nếu không ta gán p là giá trị đầu tiên của open. Và thêm p vào close. Nếu p là giá trị đích thì sẽ trả về đường đi.
  + Cho q lần lượt là các điểm con của p.
    - Gán g(q) = g[p] + khoảng cách từ q đến p, prev[q] = p và f(q) = g(q) + h(q).
    - Nếu q không nằm trong cả open và close. Thêm q vào open.
    - Nếu q nằm trong open, ta sẽ kiểm tra tiếp xem g(q) có lớn hơn g(p) + khoảng cách p-q không. Nếu có ta sẽ thay thế giá trị mới của q với giá trị cũ.
    - Cuối cùng là nếu q tồn tại trong close mà g(q) lớn hơn g(p) + khoảng cách p-q. Ta sẽ xoá q khỏi close và thêm q vào lại open. Lập lại thao tác kiểm tra open cho tới khi đến đích.
* Text, letter

  Description automatically generatedA picture containing text, receipt, screenshot

  Description automatically generatedCode:

### Kết quả và thảo luận:

* Text

  Description automatically generatedKết quả khi chạy thử chương trình với điểm bắt đầu là Arad:
* Độ phức tạp theo thời gian: O(bd)
* Trong đó b là nút đang được duyệt và d là độ sâu của hướng giải quyết.

# TỰ ĐÁNH GIÁ

|  |  |  |  |
| --- | --- | --- | --- |
| **Bài tập** | **Nội dung** | **Điểm** | **Ghi chú** |
| **Bài 1**  **(5 điểm)** | **Mô tả cấu trúc dữ liệu (1 đ)** | 1 đ |  |
| **Giải thuật BFS (2 đ)** | 1.75 đ |  |
| Sơ đồ giải thuật (0.5 đ) | 0.5 đ |  |
| Hiện thực (1 đ) | 1 đ |  |
| Kết quả và thảo luận (0.5 đ) | 0.25 đ |  |
| **Giải thuật UCS (2 đ)** | 1.75 đ |  |
| Sơ đồ giải thuật (0.5 đ) | 0.5 đ |  |
| Hiện thực (1 đ) | 1 đ |  |
| Kết quả và thảo luận (0.5 đ) | 0.25 đ |  |
| **Bài 2**  **(5 điểm)** | **Mô tả cấu trúc dữ liệu (1 đ)** | 1 đ |  |
| **Giải thuật Greedy Best First Search (Heuristic) (2 đ)** | 1.75 đ |  |
| Sơ đồ giải thuật (0.5 đ) | 0.5 đ |  |
| Hiện thực (1 đ) | 1 đ |  |
| Kết quả và thảo luận (0.5 đ) | 0.25 đ |  |
| **Giải thuật A\* (2 đ)** | 1.75 đ |  |
| Sơ đồ giải thuật (0.5 đ) | 0.5 đ |  |
| Hiện thực (1 đ) | 1 đ |  |
| Kết quả và thảo luận (0.5 đ) | 0.25 đ |  |

# TÀI LIỆU THAM KHẢO

1. <https://www.javatpoint.com/ai-informed-search-algorithms>
2. <https://www.javatpoint.com/ai-uninformed-search-algorithms>
3. <https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/>
4. <https://en.wikipedia.org/wiki/Breadth-first_search>
5. <https://www.geeksforgeeks.org/find-number-of-islands/>
6. <https://www.geeksforgeeks.org/best-first-search-informed-search/>
7. <https://www.geeksforgeeks.org/a-search-algorithm/>
8. Slide bài giảng:
   1. Lesson 2 - Solving problems by searching
   2. Lesson 3 - Informed Search