|  |  |
| --- | --- |
|  | **TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**  **KHOA CÔNG NGHỆ THÔNG TIN** |

**BÁO CÁO LAB 01: SEARCH**

**MÔN CÁC THUẬT TOÁN THÔNG MINH NHÂN TẠO VÀ ỨNG DỤNG**

1. **THÔNG TIN CHUNG**

**Yêu cầu:** Tìm hiểu và thực hiện 5 thuật toán tìm kiếm

* Breadth-first search
* Uniform-cost search
* Iterative deepening search
* Greedy-best first search
* Graph-search A\*

**Học viên thực hiện:**

**21C11036: Nguyễn Thị Ngọc Trâm**

1. **NỘI DUNG BÁO CÁO**
   1. **Giới thiệu các thuật toán**

**TÌM KIẾM MÙ**

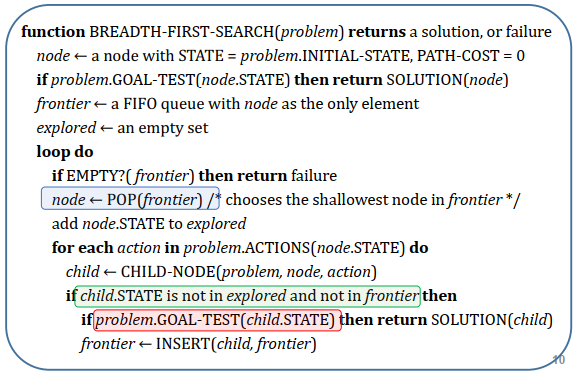
* + 1. **Breadth-first search (BFS)**

Thuật toán tìm kiếm theo chiều rộng là thuật toán tìm kiếm có hướng hoặc vô hướng, không trọng số, nếu có trọng số thì trọng số giống nhau. BFS giải quyết bài toán:

* Tìm kiếm đường đi từ một đỉnh bất kỳ đến các đỉnh khác trong đồ thị.
* Luôn tìm được đường đi ngắn nhất nếu có tồn tại đường đi.
* Độ phức tạp về bộ nhớ thì lớn hơn vấn đề thời gian chạy.

Độ phức tạp thời gian: O(V+E) với V là số đỉnh, E là số cạnh

Độ phức tạp không gian: O(bd) với b là số node, d là chiều sâu của cây mở rộng.



*Hình 1: Thuật toán BFS*

Ví dụ minh họa:

0

3

1

4

5

2

**Source**

**Goal**

**Bước 1:** **Chuẩn bị dữ liệu**

Chuyển danh sách đỉnh kề vào **graph**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Đỉnh kề | 1, 3 | 0, 2, 3, 5 | 1, 5 | 0, 1, 4, 5 | 3 | 1, 2, 3 |

Mảng đánh dấu các đỉnh đã xét **visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Trạng thái | False | False | False | False | False | False |

Mảng lưu vết đường đi **path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | -1 | -1 | -1 | -1 | -1 |

Tạo hàng đợi lưu đỉnh đang xét **queue**

|  |
| --- |
|  |

**Bước 2: Bỏ đỉnh 0**.

Bỏ đỉnh 0 vào hàng đợi **queue**

|  |
| --- |
| **0** |

Đánh dấu đã xét đỉnh 0 ở mảng **visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | **0** | 1 | 2 | 3 | 4 | 5 |
| Trạng thái | **True** | False | False | False | False | False |

**Bước 3:** **Lấy đỉnh đầu queue ra xét:**

Lấy đỉnh 0 ra xét và tìm các đỉnh có kết nối với đỉnh 0 mà chưa được xét bỏ vào queue. Cập nhật:

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | **0** | 1 | 2 | 3 | 4 | 5 |
| Đỉnh kề | **1, 3** | 0, 2, 3, 5 | 1, 5 | 0, 1, 4, 5 | 3 | 1, 2, 3 |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | **3** | 4 | 5 |
| Trạng thái | True | **True** | False | **True** | False | False |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | **3** | 4 | 5 |
| Lưu vết | -1 | **0** | -1 | **0** | -1 | -1 |

**queue**

|  |  |
| --- | --- |
| 1 | 3 |

* **Chạy lại bước 3 lần 2:**

**Lấy đỉnh 1 ra khỏi queue, xét đỉnh 1:**

**queue:**

|  |
| --- |
| **3** |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | 3 | 4 | 5 |
| Đỉnh kề | 1, 3 | **0, 2, 3, 5** | 1, 5 | 0, 1, 4, 5 | 3 | 1, 2, 3 |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | **2** | 3 | 4 | **5** |
| Trạng thái | True | True | **True** | True | False | **True** |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | **2** | 3 | 4 | **5** |
| Lưu vết | -1 | 0 | **1** | 0 | -1 | **1** |

**queue**

|  |  |  |
| --- | --- | --- |
| 3 | 2 | 5 |

* **Chạy lại bước 3 lần 3:**

**Lấy đỉnh 3 ra khỏi queue, xét đỉnh 3:**

**queue:**

|  |  |  |
| --- | --- | --- |
| **3** | 2 | 5 |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | **3** | 4 | 5 |
| Đỉnh kề | 1, 3 | 0, 2, 3, 5 | 1, 5 | **0, 1, 4, 5** | 3 | 1, 2, 3 |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | **4** | 5 |
| Trạng thái | True | True | True | True | **True** | True |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | **4** | 5 |
| Lưu vết | -1 | 0 | 1 | 0 | **3** | 1 |

**queue**

|  |  |  |
| --- | --- | --- |
| 2 | 5 | 4 |

* **Chạy lại bước 3 lần 4:**

**Lấy đỉnh 2 ra khỏi queue, xét đỉnh 2:**

**queue:**

|  |  |  |
| --- | --- | --- |
| **2** | 5 | 4 |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | **2** | 3 | 4 | 5 |
| Đỉnh kề | 1, 3 | 0, 2, 3, 5 | **1, 5** | 0, 1, 4, 5 | 3 | 1, 2, 3 |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Trạng thái | True | True | True | True | True | True |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | 0 | 1 | 0 | 3 | 1 |

**queue**

|  |  |
| --- | --- |
| 5 | 4 |

* **Chạy lại bước 3 lần 5:**

**Lấy đỉnh 5 ra khỏi queue, xét đỉnh 5:**

**queue:**

|  |  |
| --- | --- |
| **5** | 4 |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | **5** |
| Đỉnh kề | 1, 3 | 0, 2, 3, 5 | 1, 5 | 0, 1, 4, 5 | 3 | **1, 2, 3** |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Trạng thái | True | True | True | True | True | True |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | 0 | 1 | 0 | 3 | 1 |

**queue**

|  |
| --- |
| 4 |

* **Chạy lại bước 3 lần 6:**

**Lấy đỉnh 5 ra khỏi queue, xét đỉnh 5:**

**queue:**

|  |
| --- |
| **4** |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | **4** | 5 |
| Đỉnh kề | 1, 3 | 0, 2, 3, 5 | 1, 5 | 0, 1, 4, 5 | **3** | 1, 2, 3 |

**visited**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Trạng thái | True | True | True | True | True | True |

**path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | 0 | 1 | 0 | 3 | 1 |

**queue**

Empty

**= > path:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| **Lưu vết** | **-1** | **0** | **1** | **0** | **3** | **1** |

BFS(source, goal) = > **0 🡪 1 🡪 5**

Thứ tự duyệt BFS: 0, 1, 3, 2, 5, 4

***Kết luận:*** Thuật toán BFS chỉ tìm được đường đi có số bước biến đổi ít nhất. Nó không thể tìm thấy đường đi có chi phí nhỏ nhất. Đối với những bài toán tìm kiếm mà các bước biến đổi có chi phí (đồ thị có trọng số) thì BFS cho ra lời giải không phù hợp.

Thuật toán tìm kiếm theo chiều rộng sẽ tốt trong trường hợp các bước biến đổi có chi phí như nhau, vì nó luôn mở rộng nút gần nhất.

* + 1. **Uniform-cost search (UCS)**

UCS (Thuật toán tìm kiếm chi phí đồng nhất): Thay vì mở rộng đến nút gần nhất như BFS thì UCS sẽ mở rộng nút n có chi phí đường đi là thấp nhất => Thuật toán UCS thì tổng quát hơn BFS.

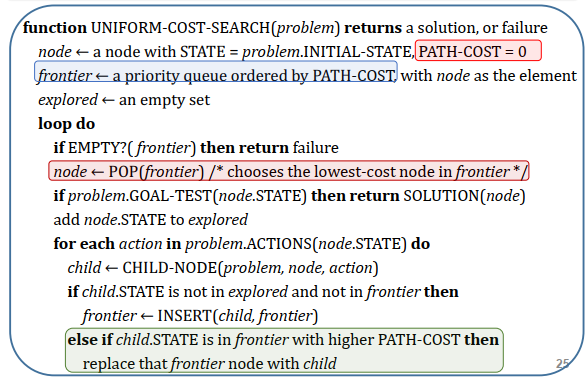
Thuật toán UCS tương đương BFS khi các bước biến dổi có chi phí như nhau.

Tương đương thuật toán Dijkstra.

Độ phức tạp thuật toán bình thường sẽ là O(V2) với V là số đỉnh.

Tuy nhiên có thể giảm nếu sử dụng kết hợp các cấu trúc dữ liệu khác:

* Dùng Heap (priority queue) độ phức tạp là: O(Elogv).
* Dùng kết hợp với cấu trúc dữ liệu cây nhị phân tìm kiếm: O(ElogV)



*Hình 2: Thuật toán UCS*

Ví dụ minh họa:

**1**

**2**

**5**

**0**

**3**

**4**

**7**

**5**

**1**

**1**

**2**

**2**

**1**

**4**

**3**

**1**

**Bước 1: Chuẩn bị dữ liệu**

Danh sách đỉnh kề và trọng số vào **graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| (Đỉnh kề,  Trọng số) | (1, 1) | (2, 5)  (3, 2)  (5, 7) | (5, 1) | (0, 2)  (2, 1)  (4, 4) | (3, 3) | (4, 1) |

Mảng chứa chi phí đường đi **dist**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Chi phí | INF | INF | INF | INF | INF | INF |

Mảng lưu vết đường đi **path**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | -1 | -1 | -1 | -1 | -1 |

Hàng đợi ưu tiên **priority queue,** lưu cặp giá trị (đỉnh, chi phí)

|  |
| --- |
|  |

**Bước 2: Lấy đỉnh bắt đầu đi** (đỉnh 0). Gán chi phí cho đỉnh 0 là 0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Chi phí | 0 | INF | INF | INF | INF | INF |

**Priority queue**

|  |
| --- |
| **(0, 0)** |

**Bước 3: Chạy thuật toán lần 1**

**Lấy đỉnh 0, và chi phí 0 ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 0**

**Priority queue**

|  |
| --- |
| **(0, 0)** |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | **0** | 1 | 2 | 3 | 4 | 5 |
| (Đỉnh kề,  Trọng số) | **(1, 1)** | (2, 5)  (3, 2)  (5, 7) | (5, 1) | (0, 2)  (2, 1)  (4, 4) | (3, 3) | (4, 1) |

**dist**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | 3 | 4 | 5 |
| Chi phí | 0 | **1** | INF | INF | INF | INF |

**path**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | **0** | -1 | -1 | -1 | -1 |

Hàng đợi ưu tiên **priority queue,** lưu cặp giá trị (đỉnh, chi phí)

|  |
| --- |
| **(1, 1)** |

**Chạy lại bước 3 lần 2:**

**Lấy đỉnh 1, và chi phí 1 ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 1**

**Priority queue**

|  |
| --- |
| **(1, 1)** |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | 2 | 3 | 4 | 5 |
| (Đỉnh kề,  Trọng số) | (1, 1) | **(2, 5)**  **(3, 2)**  **(5, 7)** | (5, 1) | (0, 2)  (2, 1)  (4, 4) | (3, 3) | (4, 1) |

**dist**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | **2** | **3** | 4 | **5** |
| Chi phí | 0 | **1** | **5+1 = 6** | **2+1 = 3** | INF | **7+1 = 8** |

**path**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | **2** | **3** | 4 | **5** |
| Lưu vết | -1 | 0 | **1** | **1** | -1 | **1** |

Hàng đợi ưu tiên **priority queue,** lưu cặp giá trị (đỉnh, chi phí)

|  |  |  |
| --- | --- | --- |
| **(3, 3)** | **(2, 6)** | **(5, 8)** |

**Chạy lại bước 3 lần 3:**

**Lấy đỉnh 3, và chi phí 3 ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 3**

**Priority queue**

|  |  |  |
| --- | --- | --- |
| **(3, 3)** | **(2, 6)** | **(5, 8)** |

**graph:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | **3** | 4 | 5 |
| (Đỉnh kề,  Trọng số) | (1, 1) | (2, 5)  (3, 2)  (5, 7) | (5, 1) | **(0, 2)**  **(2, 1)**  **(4, 4)** | (3, 3) | (4, 1) |

**dist**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | **1** | **2** | 3 | **4** | 5 |
| Chi phí | 0 | **1** | **6 🡪 4** | 3 | **3+4 = 7** | 8 |

**path**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | **2** | 3 | **4** | 5 |
| Lưu vết | -1 | 0 | **3** | 1 | **3** | 1 |

Hàng đợi ưu tiên **priority queue,** lưu cặp giá trị (đỉnh, chi phí)

|  |  |  |  |
| --- | --- | --- | --- |
| **(2, 4)** | **(2, 6)** | **(4, 7)** | **(5, 8)** |

**Tương tự chạy thuật toán đến khi priority queue rỗng, mảng path không thay đổi 🡪 dừng thuật toán**

**Kết quả:**

**path**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | 4 | 5 |
| Lưu vết | -1 | 0 | 3 | 1 | 5 | 2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 | **4** | 5 |
| Chi phí | 0 | 1 | 4 | 3 | **6** | 5 |

Đường đi ngắn nhất từ 0 đến 4:

**0 🡪 1 🡪 3 🡪 2 🡪 5 🡪 4**

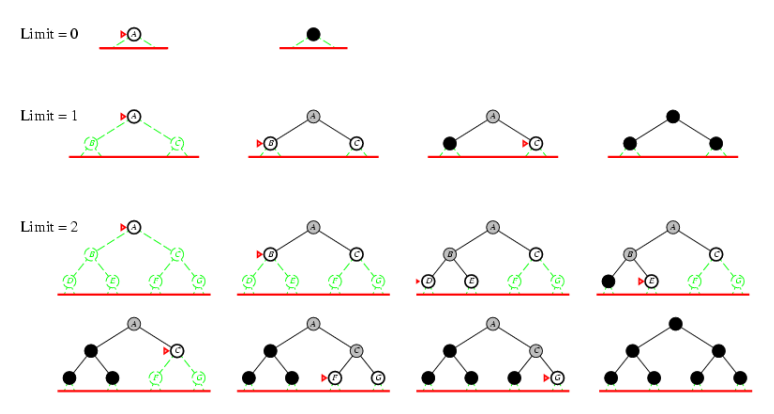
**Cost: 6**

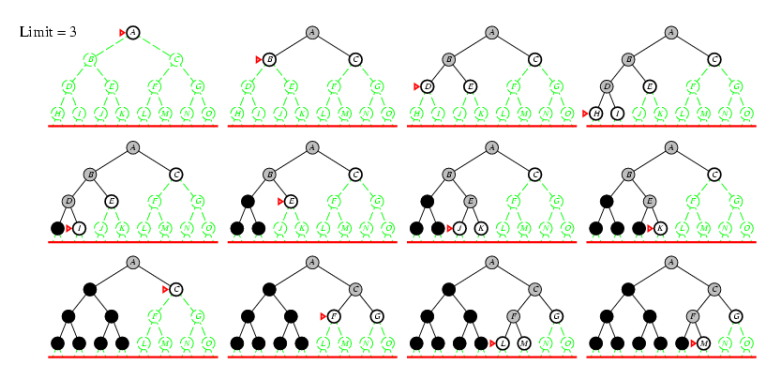
***Kết luận*:** Thuật toán tìm kiếm chi phí đồng nhất không quan tâm đến số bước của đường đi, mà chỉ quan tâm đến chi phí tổng cộng. Do đó, thuật toán có thể lặp vô hạn, nếu gặp một nút có chi phí chuyển đổi tới cùng trạng thái đó bằng 0.

* + 1. **Iterative deepening search (IDS)**

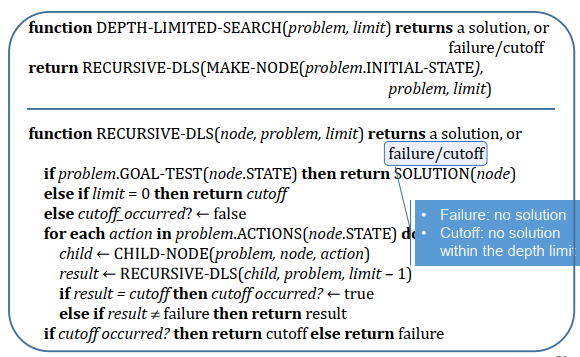
Tìm kiếm lặp sâu dần (Tìm kiếm với độ sâu tăng dần): là phương pháp cải tiến vấn đề chọn độ sâu nào tốt nhất bằng cách lần lượt thử hết các độ sâu có thể (0, 1, 2, …).

IDS kết hợp những ưu điểm của BFS và DFS: vừa tối ưu (khi chi phí không giảm theo độ sâu), vừa đầy đủ (khi số nhánh con tối đa b hữu hạn) giống như BFS như lại có sự tối ưu về bộ nhớ O(bd) của tìm kiếm theo chiều sâu.

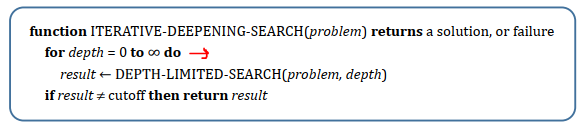




*Hình 3: Bốn bước lặp của thuật toán lặp sâu dần trên cây nhị phân*



*Hình 4: Thuật toán Depth-limited search (DLS)*

****

*Hình 5: Iterative deepening search (IDS)*

***Kết luận***: Độ phức tạp về mặt thời gian của IDS là O(bd) và về mặt không gian là O(bxd). Tóm lại, IDS được sử dụng nhiều khi không gian tìm kiếm lớn và độ sâu của lời giải không xác định.

IDS nhanh hơn BFS dù trạng thái phát sinh lặp nhiều lần.

**TÌM KIẾM CÓ THÔNG TIN**

* + 1. **Greedy-best fist search (GBFS)**

Tìm kiếm tối ưu kiểu tham lam (GBFS) mở rộng nút gần đích nhất với hi vọng cách làm này dẫn đến lời giải một cách nhanh nhất. Do đó thuật toán này đánh giá chi phí của các nút chỉ dựa trên hàm heuristic.

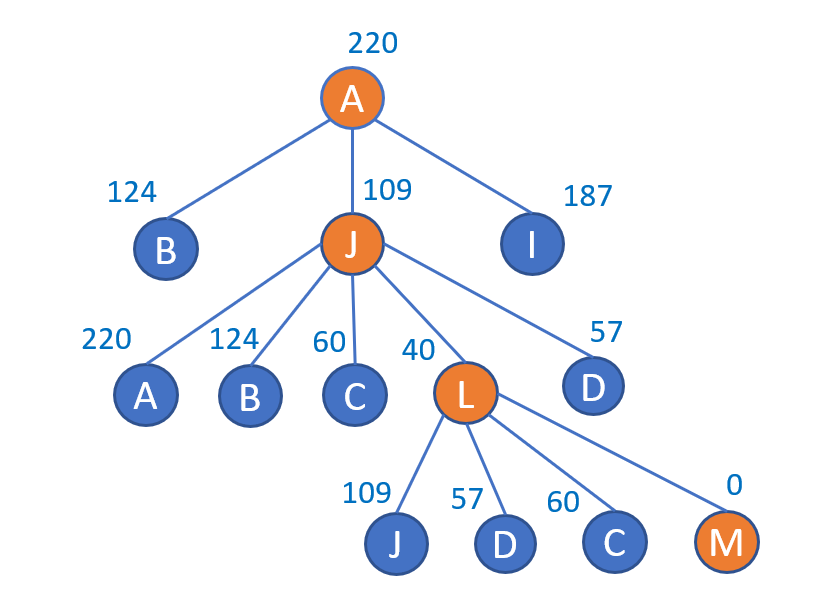
f(n) = h(n)

Ví dụ minh họa:

Cho heuristic là đường chim bay.

|  |  |  |  |
| --- | --- | --- | --- |
| A (Source) | 220 | I | 187 |
| B | 124 | J | 109 |
| C | 60 | K | 130 |
| D | 57 | L | 40 |
| E | 46 | M (Goal) | 0 |
| F | 27 | N | 48 |
| G | 51 | O | 141 |
| H | 55 | P | 83 |

*Bảng 1: Khoảng cách đường chim bay đến Goal.*



*Hình 6: Minh họa thuật toán Greedy Best first search (chọn node có khoảng cách đến goal nhỏ nhất)*

***Kết quả***: Greedy-best first search giống DFS ở cách tìm kiếm chỉ đi theo một con đường để lần đến đích, còn nếu gặp ngõ cụt sẽ quay lại đi sang hướng khác.

Khuyết điểm cũng giống DFS đó là: không tối ưu và không đầy đủ vì nó có thể đi vào con đường vô tận và không bao giờ quay lại được.

Độ phức tạp thời gian và không gian là O(bm) với m là độ sâu tối đa của không gian tìm kiếm. Tuy nhiên với hàm heuristic tốt thì độ phức tạp có thể giảm đáng kể.

* + 1. **Graph-search A\***

Thuật toán A\* đánh giá một nút dựa trên *chi phí đi từ nút gốc đến nút đó g(n)*, cộng với *chi phí của nút đó đến đích h(n)*

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

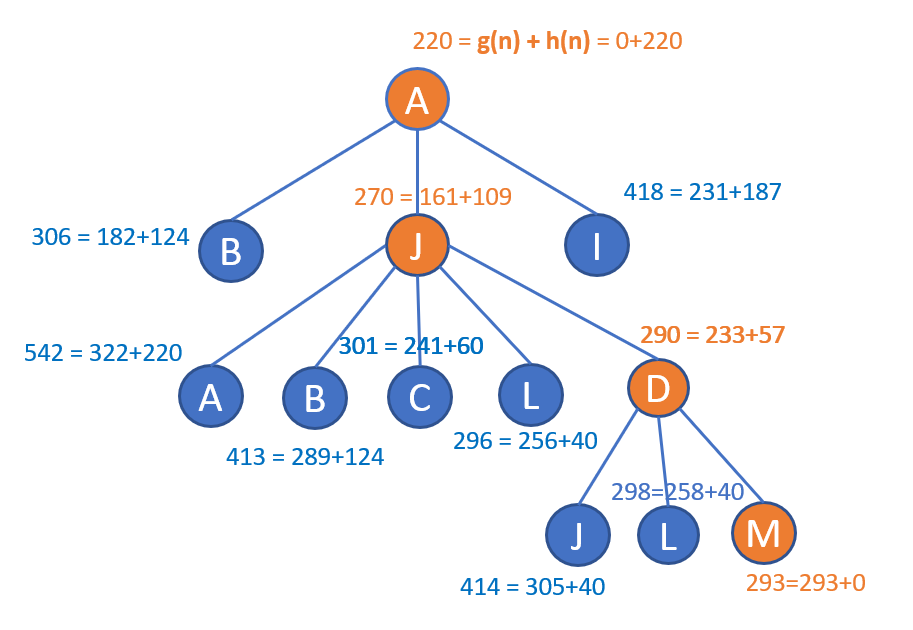
f(n) = ước lượng chi phí của lời giải “tốt nhất” qua n

Do đó, nếu tìm lời giải tốt nhất nghĩa là tìm g(n)+h(n) nhỏ nhất. Hơn nữa nếu heuristic thỏa một số điều kiện nhất định thì tìm kiếm A\* đầy đủ và tối ưu.

Để đảm bảo A\* tìm được lời giải tối ưu thì hàm h(n) phải là hàm heuristic chấp nhận được.

Tuy nhiên, độ phức tạp về mặt không gian của A\* vẫn rất cao.

Ví dụ minh họa:



*Hình 7: Minh họa thuật toán A\* (chọn node có khoảng cách đến goal nhỏ nhất) với g(n) là chi phí thực sự đến nút n*

* 1. **Demo - Minh họa các thuật toán**

Thư viện sử dụng: Tkinter

Giới thiệu các hàm chuẩn bị cho trước khi chạy thuật toán trong phần code demo:

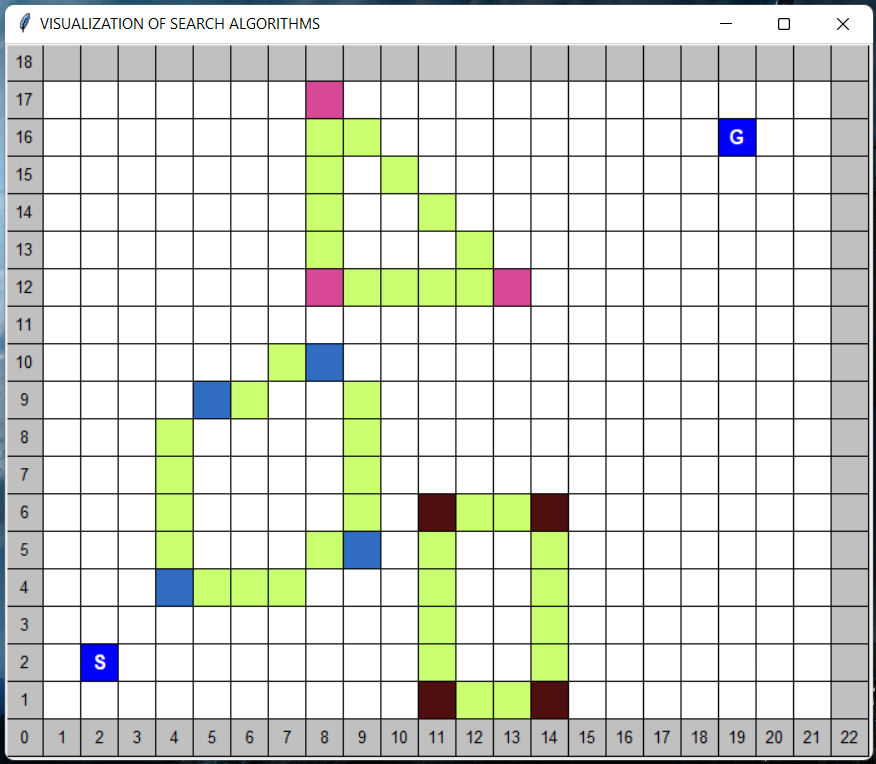
+ Hàm *getInput(path)*: đầu vào là đường dẫn của file input chứa các thông tin input, trả ra các giá trị mazeSize, SGPos, nObstacle, obstacleList tương ứng kích thước mê cung, vị trí của điểm nguồn và điểm đích, số lượng chướng ngại vật, danh sách vị trí của các chướng ngại vật tương ứng.

+ Hàm *createCell()*: input là tọa độ một điểm trong mê cung, màu sắc, … Output: tô màu ô muốn tô.

+ Hàm *createMaze()*: input là size của mê cung lấy từ hàm getInput(), output: mê cung với kích thước như file input yêu cầu.

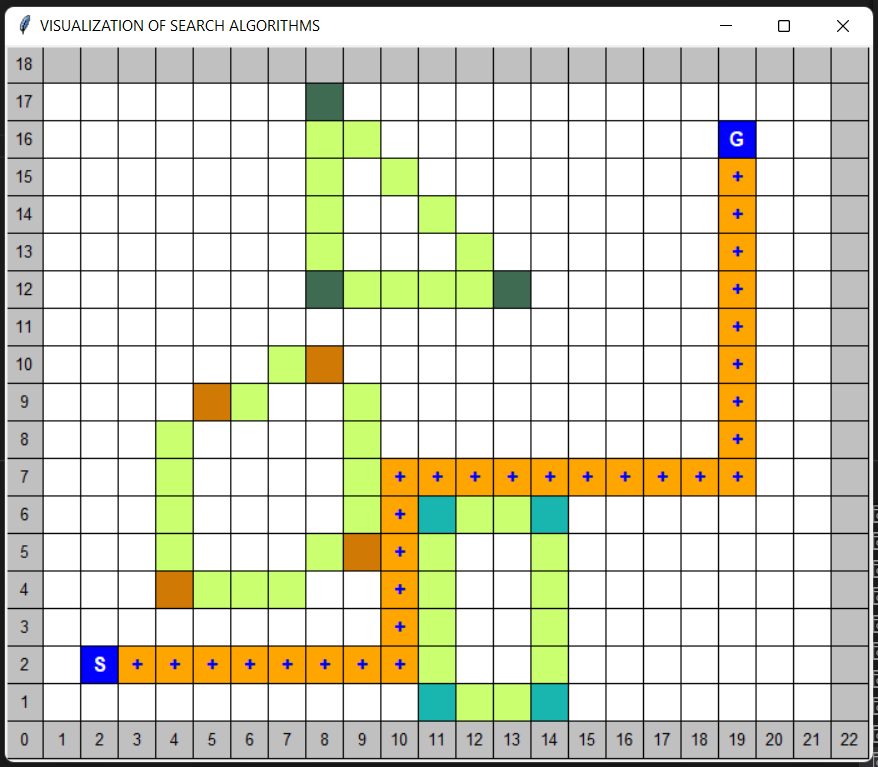
+ Class OBSTACLE: Tạo các obstacle dựa trên danh sách các điểm cho trước. Các điểm theo chiều kim đồng hồ. Ý tưởng: xét 2 điểm liền kề nếu các điểm này có cùng hoành, tung độ thì các điểm thảnh phần đi thẳng (ví dụ trường hợp obstacle là hình vuông hoặc chữ nhật); 2 điểm lệch 1 đơn vị thì cho đi thẳng và ô kế cuối lệch 1 đơn vị (các điểm của khối polygon đầu tiên trong file input); 2 điểm có hoành độ và tung độ lệch n đơn vị mỗi ô thành phần mới tạo ra sẽ lệch n đơn vị (áp dụng cho đường chéo của tam giác);

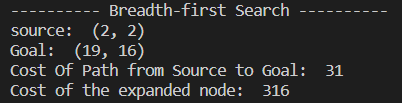
+ Hàm *get\_next\_node():* đầu vào là điểm đang xét và trả ra các điểm trên, dưới, trái, phải so với điểm đó nếu tồn tại vì có điểm nằm ngoài bìa mê cung.



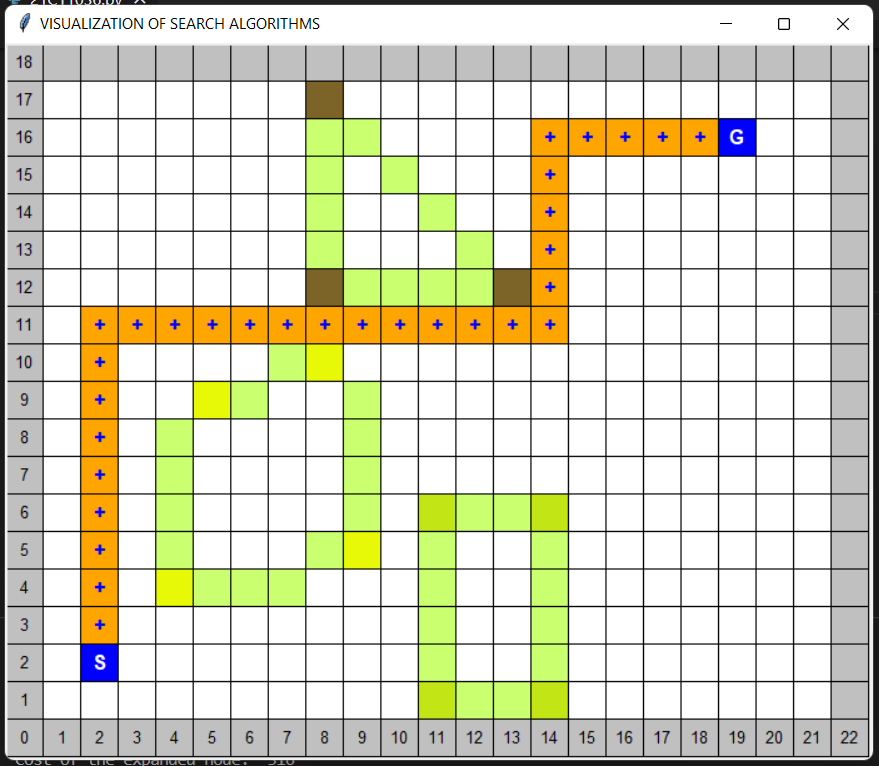
*Hình 8: Màn hình trước khi chạy thuật toán gồm có điểm nguồn (S), điểm đích (G) màu xanh và các khối chướng ngại vật màu xanh lá với các điểm gốc là bộ màu tạo random*

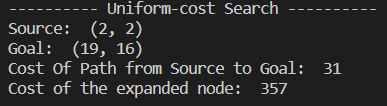
+ Hàm *drawPath()* lấy tọa độ điểm nguồn, đích và đường đi lưu vết sau khi chạy các thuật toán tương ứng và trả ra đường đi, chi phí đường đi, chi phí mở rộng các nút, …



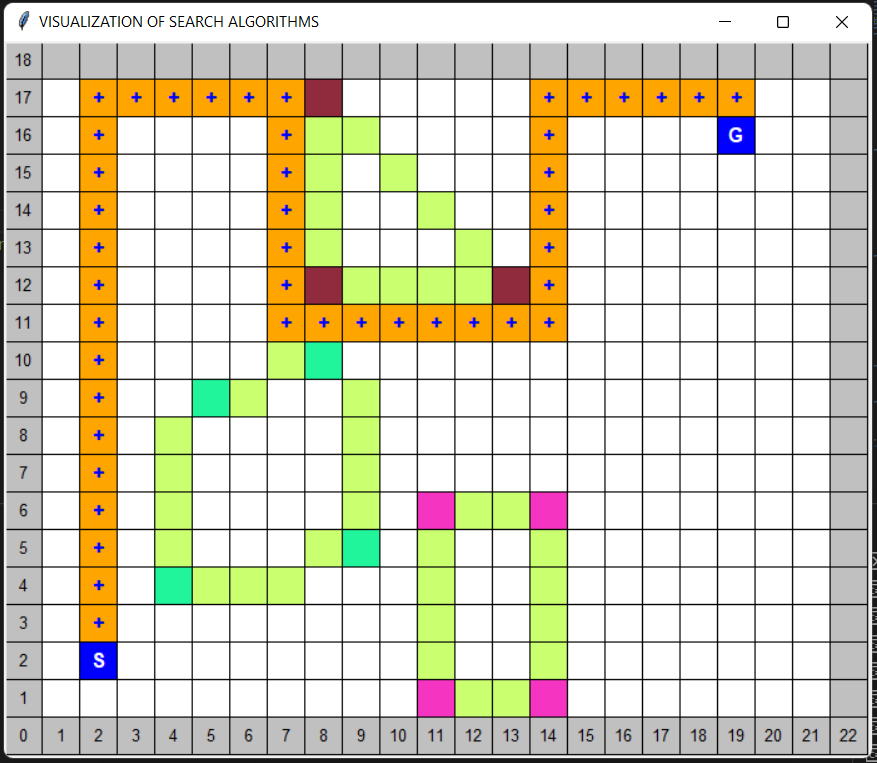


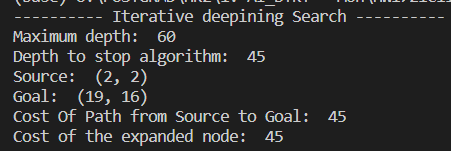
*Hình 9: Kết quả sau khi chạy thuật toán BFS*



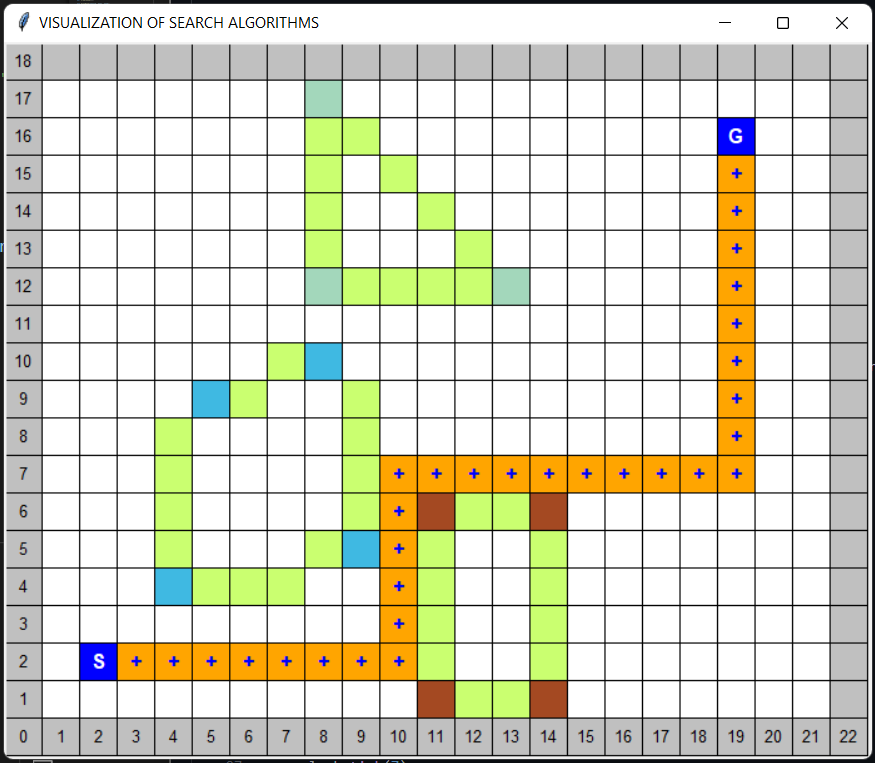


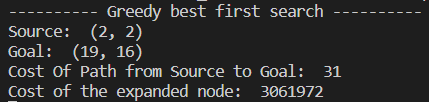
*Hình 10: Kết quả sau khi chạy thuật toán UCS*



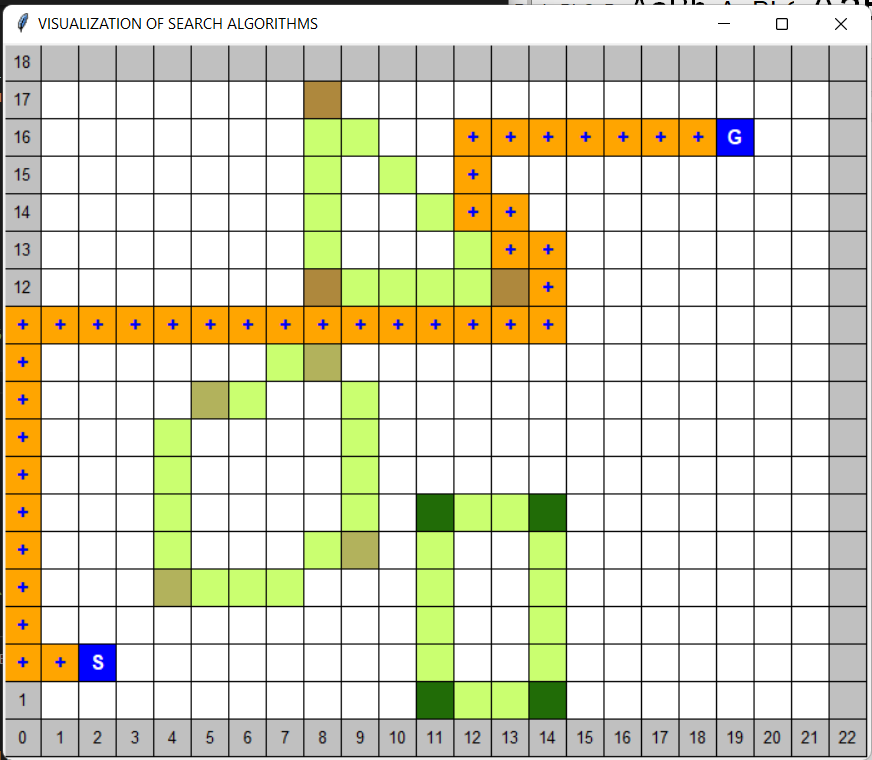


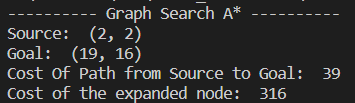
*Hình 10: Kết quả sau khi chạy thuật toán IDS với đầu vào chiều sâu tối đa cây là 60, thuật toán dừng ở độ sâu 45 = chi phí đường đi.*





*Hình 11: Kết quả khi chạy thuật toán Greedy Best first search*

**

**

*Hình 12: Kết quả trả ra khi chạy thuật toán Graph search A\**