ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO
TÌM KIẾM TRONG ĐỒ THỊ

Nguyễn Đình Thành - 21120334

Môn: Cơ sở trí tuệ nhân tạo
Lớp: 21_21
Giảng viên: Thầy Lê Hoài Bắc
Giáo viên thực hành: Thầy Nguyễn Bảo Long
Hồ Chí Minh, tháng 10 năm 2023

Mục lục

1 Mở đầu

Trong trí tuệ nhân tạo, các thuật toán tìm kiếm đóng một vai trò rất quan trọng trong việc giải quyết các vấn đề thông qua việc tìm kiếm trong không gian trạng thái. Các thuật toán tìm kiếm được chia ra thành hai loại: tìm kiếm không có thông tin (tìm kiếm mù) và tìm kiếm có thông tin. Trong báo cáo này, chúng ta sẽ tìm hiểu về các thuật toán tìm kiếm cơ bản trong trí tuệ nhân tạo, bao gồm DFS, BFS, UCS và A*.

2 Nội dung chi tiết

2.1 Bài toán tìm kiếm

2.1.1 Định nghĩa bài toán tìm kiếm

Một bài toán tìm kiếm bao gồm các thành phần sau:

- Trạng thái bắt đầu: Là nơi mà tác nhân thông minh bắt đầu tìm kiếm.
- Hành động: Những hành động mà tác nhân thông minh có thể thực hiện.
- Trạng thái kế tiếp: Các trạng thái mà một hành động cụ thể có thể dẫn tới.
- Chi phí: xác định chi phí của đường đi từ trạng thái bắt đầu đến một trạng thái khác.
- Kiểm tra về đích: kiểm tra trạng thái hiện tại có phải là đích không.

2.1.2 Mã giả của bài toán tìm kiếm

Mã giả tổng quát để giải quyết bài toán tìm kiếm được trình bày như sau:

Algorithm 1 Solving problem by searching

```
procedure SEARCHING-ALGORITHM(PROBLEM)
initialize(frontier with problem.initialState)
initialize(exploredSet)
while frontier is not empty:
    state ← frontier.pop()
    if problem.isGoal(state):
        return path_to_node
    add state to exploredSet
    for action in problem.getActions(state):
        child ← problem.getActions(state
        if child not in exploredSet and child not in frontier:
            push child into frontier
return failed
```

2.1.3 Phân loại bài toán tím kiếm

Bài toán tìm kiếm được phân ra thành hai loại: tìm kiếm không có thông tin và tìm kiếm có thông tin.

- Tìm kiếm không có thông tin thuật toán tìm kiếm dựa trên thông tin cơ bản về các trạng thái và các hành động có thể thực hiện. Phương pháp tìm kiếm không thông tin không sử dụng thông tin đặc biệt về mục tiêu hoặc không gian trạng thái để hướng dẫn quá trình tìm kiếm.

- Tìm kiếm có thông tin là thuật toán tìm kiếm ngoài việc sử dụng các thông tin cơ bản về trạng thái và các hành động có thể thực hiện còn sử dụng thêm các thông tin đặc biệt về mục tiêu hoặc không gian trạng thái để hướng dẫn quá trình tìm kiếm. Tìm kiếm có thông tin thường cho kết quả tốt hơn và giảm số lượng trạng thái phải duyệt qua.

2.2 Các thuật toán tìm kiếm

2.2.1 Depth-First Search (DFS)

Depth-First Search là chiến lược tìm kiếm luôn chọn trạng thái sâu nhất tính từ trạng thái bắt đầu để mở.

Mã giả của thuật toán DFS được trình bày như sau:

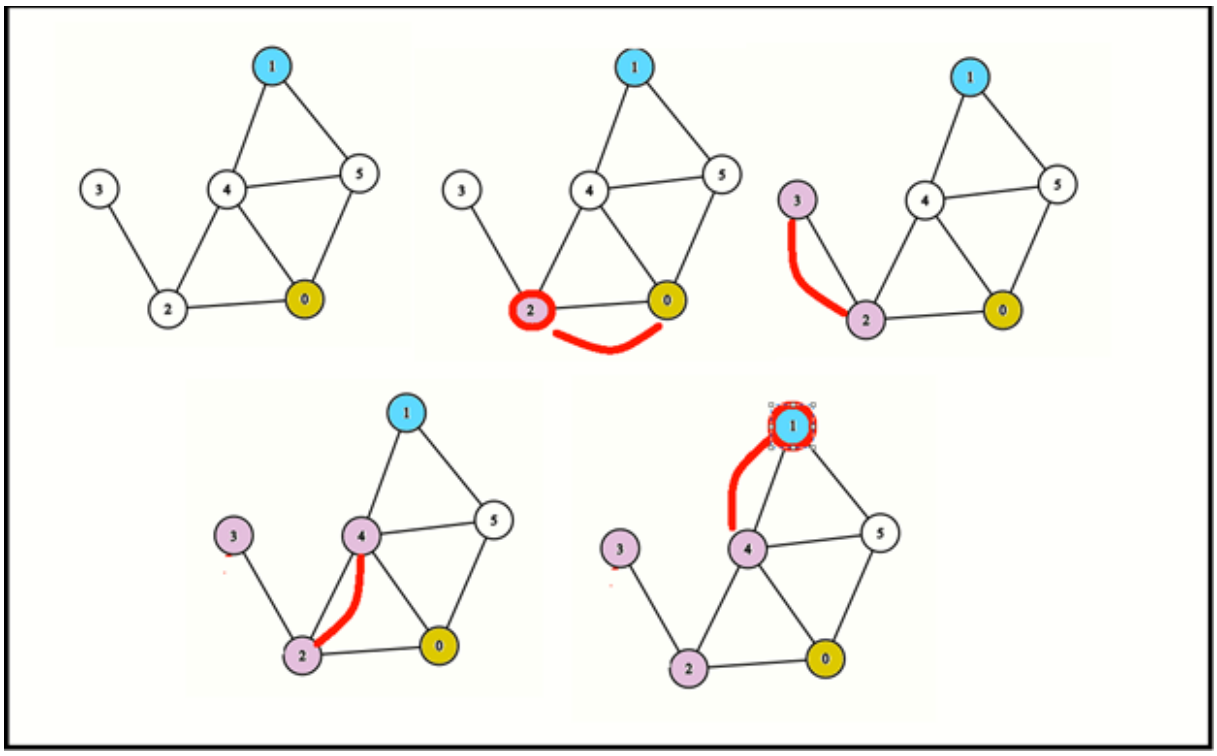
Algorithm 2 Depth First Search

```
procedure DFS(PROBLEM)
initialize(stack with initial state)
initialize(exploredSet)
while stack is not empty:
    state ← pop from stack
    if problem.isGoal(state):
        return solution
    if state not in exploredSet
        add state to exploredSet
        for action in problem.getActions(state):
            child ← problem.getResult(state, action)
            if child not in exploredSet:
                push child onnto stack
return failed
```

Phân tích thuật toán:

- Tính hoàn thiện: DFS đảm bảo tính hoàn thiện vì với không gian trạng thái có hữu hạn các trạng thái thì thuật toán này sẽ duyệt qua tất cả các trạng thái có thể duyệt đến trong không gian trạng thái.
- Tính tối ưu: thuật toán DFS không đảm bảo tính tối ưu do thuật toán này không sử dụng đến chi phí của đường đi trong quá trình tìm kiếm.
- Độ phức tạp: Độ phức tạp của BFS phụ thuộc vào kích thước của không gian tìm kiếm và cấu trúc của cây tìm kiếm. Tuy nhiên, trong trường hợp xấu nhất, độ phức tạp không gian của thuật toán sẽ là O(bm) và độ phức tạp thời gian sẽ là O(b^m) với b là hệ số phân nhánh và m là độ sâu lớn nhất của cây tìm kiếm.

Hình dưới đây mô tả từng bước chạy của thuật toán tìm kiếm theo chiều sâu trong việc tìm kiếm đường đi từ đỉnh 0 tới đỉnh 1:



2.2.2 Breadth-First Search (BFS)

Breath-First Search là chiến lược tìm kiếm luôn luôn chọn mở trạng thái tiếp theo là trạng thái nông nhất trong tập mở tính từ trạng thái bắt đầu.

Mã giả của thuật toán BFS được trình bày như sau:

Algorithm 3 Breath First Search

```
procedure BFS(PROBLEM)
initialize(queue with initial state)
initialize(exploredSet)
while queue is not empty:
        state ← dequeue from queue
        if problem.isGoal(state):
                return solution
        if state not in exploredSet
                add state to exploredSet
                for action in problem.getActions(state):
                        child <- problem.getResult(state, action)
                        if child not in exploredSet:
                                enqueue child into queue
return failed
```

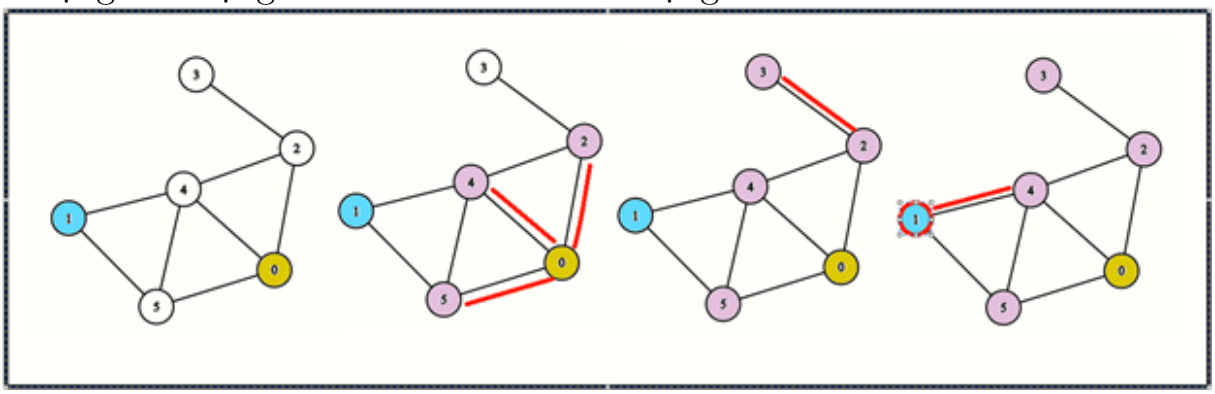
Phân tích thuật toán:

- Tính hoàn thiện: thuật toán BFS đảm bảo tính hoàn thiện vì với không gian trạng thái có hữu hạn các trạng thái, thuật toán này sẽ lần lượt duyệt qua hết các trạng thái có thể đến được theo từng bậc và có thể đến được trạng thái đích nếu tồn tại được đi từ trạng thái bắt đầu đến trạng thái đó.

- Tính tối ưu: thuật toán BFS không đảm bảo tính tối ưu do thuật toán này không sử dụng đến chi phí của đường đi trong quá trình tìm kiếm.

- Độ phức tạp: Độ phức tạp của BFS phụ thuộc vào kích thước của không gian tìm kiếm và cấu trúc của cây tìm kiếm. Trong trường hợp tệ nhất, thuật toán sẽ có độ phức tạp thời gian là $O(b^d)$ và độ phức tạp không gian cũng là $O(b^d)$, trong đó b là số lượng nhánh tối đa từ mỗi nút và d là độ sâu của lời giải ở độ sâu bé nhất.

Hình dưới đây mô tả từng bước chạy của thuật toán tìm kiếm theo chiều rộng với trạng thái ban đầu là 0 và trạng thái đích là 1:



2.2.3 Uniform Cost Search (UCS)

Uniform Cost Search là chiến lược tìm kiếm luôn chọn mở trạng thái có chi phí thấp nhất tính từ trạng thái bắt đầu.

Mã giả của thuật toán UCS như sau:

Phân tích thuật toán:

- Tính hoàn thiện: thuật toán UCS luôn đảm bảo tính hoàn thiện nêu mọi chi phí đều không âm.

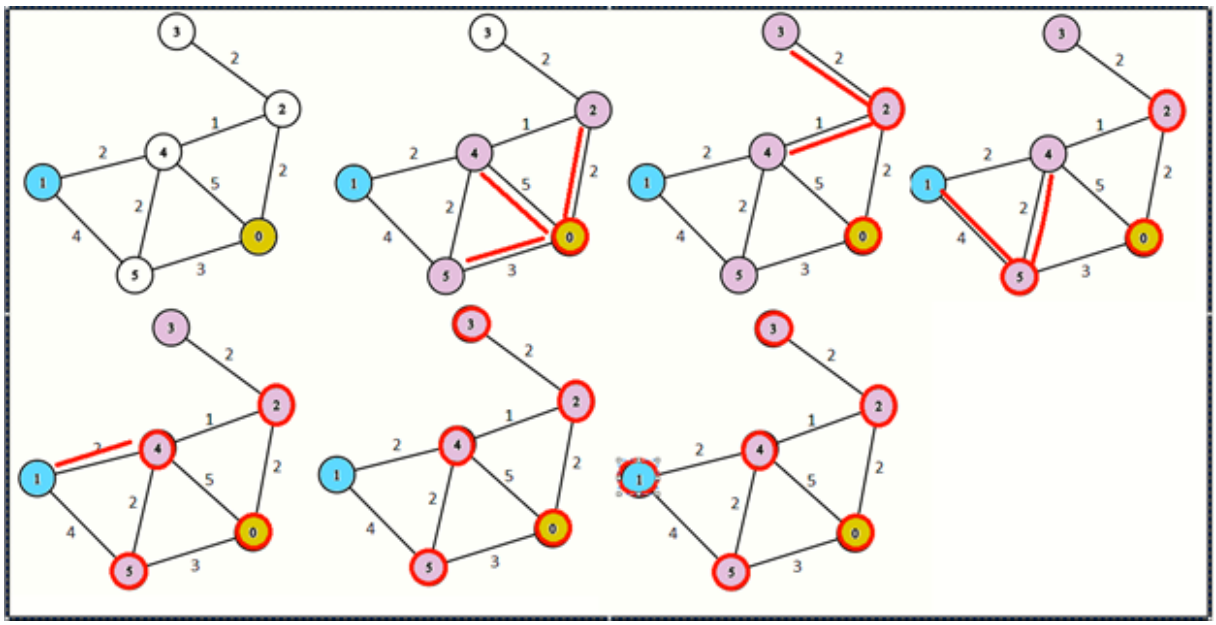
Algorithm 4 Uninform Cost Search

```
procedure UCS(PROBLEM)
initialize(priority queue)
initialize(exploredSet)
push initial state with cost 0 into priority queue
while priority queue is not empty:
    current_Cost ,state ← dequeue from priority queue
    if problem.isGoal(state):
        return solution
    add state to exploredSet
    for action in problem.getActions(state):
        child ← problem.getResult(state, action)
        newCost ← current_cost + cost(state,child)
        if child not in exploredSet and child not in priority queue:
            enqueue child with newCost into queue
        else if child is in priority queue and newCost < cost of this child:
            update priority queue with the newCost of this child
return failed
```

- Tính tối ưu: UCS cũng đảm bảo tìm được giải pháp tối ưu cho vấn đề tìm kiếm với chi phi nhỏ nhất.

- Độ phức tạp: Với hệ số phân nhánh là b và chi phí của giải pháp tối ưu là C. Ta có độ phức tạp thời gian của thuật toán này là O($b^{\frac{C*}{\epsilon}}$) và độ phức tạp không gian cũng là O($b^{\frac{C*}{\epsilon}}$).

Dưới đây là hình ảnh minh hoạ từng bước thực hiện thuật toán UCS để tìm đường đi từ đỉnh 0 tới đỉnh 1:



2.2.4 A Star (A*)

A Star là chiến lược tìm kiếm luôn luôn chọn trạng thái tiếp theo là trạng thái có tổng chi phí ước tính bé nhất, với tổng chi phí là toàn bộ chi phí từ trạng thái bắt đầu đến trạng thái kết thúc.

Mã giả của thuật toán A Star được trình bày như sau:

Algorithm 5 A Star

```
procedure ASTAR(PROBLEM)
initialize(priority queue)
initialize(exploredSet)
push initial state with estimated cost into priority queue
while priority queue is not empty:
    current_Cost ,state ← dequeue from priority queue
    if problem.isGoal(state):
        return solution
    add state to exploredSet
    for action in problem.getActions(state):
        child ← problem.getResult(state, action)
        f(child) ← g(child) + h(child)
        if child not in exploredSet and child not in priority queue:
            enqueue child with estimated total cost f(child) into queue
        else if child is in priority queue and f(child) < cost of this child:
            update priority queue with the f(child) of this child
return failed
```

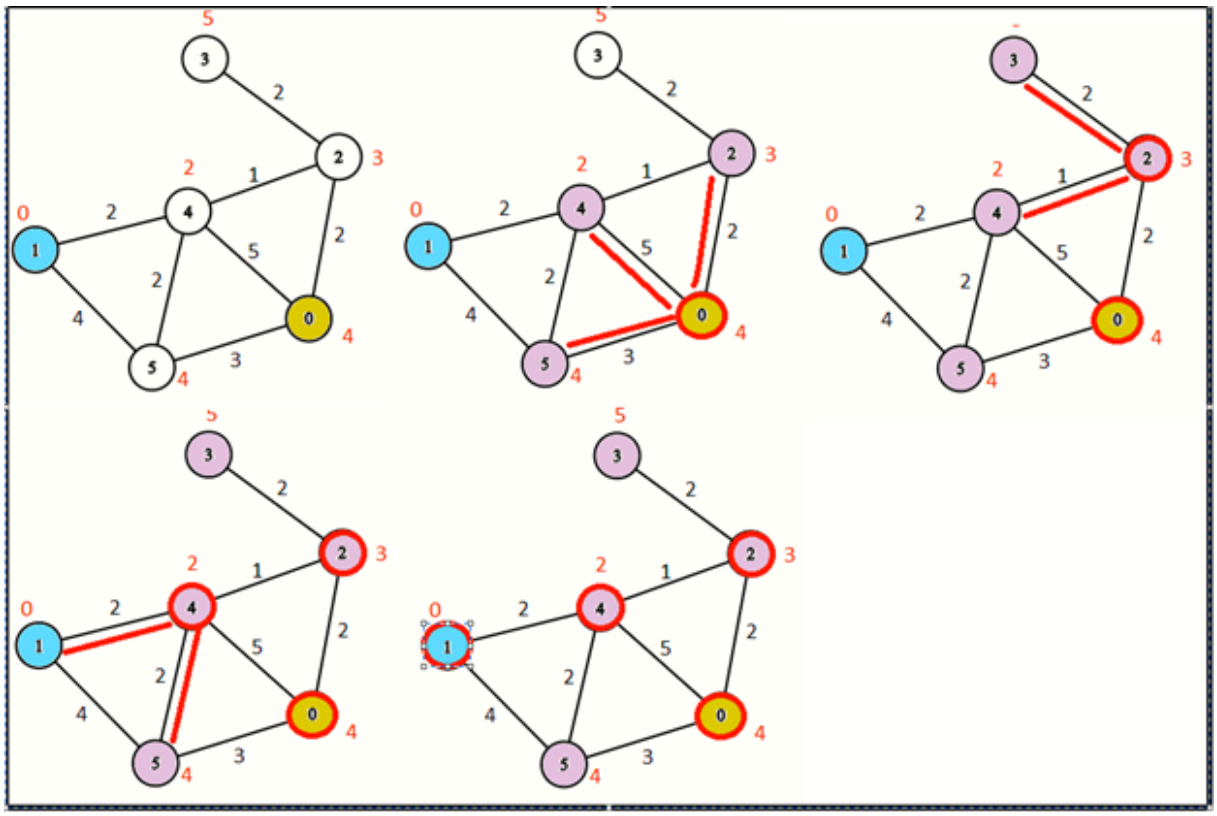
Phân tích thuật toán:

- Tính hoàn thiện: Nếu sử dụng một hàm heuristic đủ tốt, thuật toán A Star hoàn toàn có thể đảm bảo tính hoàn thiện.
- Tính tối ưu: Tương tự như tính hoàn thiện, khi được cho một hàm heuristic đủ tốt, thuật toán A Star cũng có thể đảm bảo tính tối ưu.
- Độ phức tạp: Độ phức tạp của thuật toán A Star hoàn toàn phụ thuộc vào hàm heuristic nhưng trong trường hợp tệ nhất, thuật toán này sẽ có độ phức tạp là $O(b^d)$ với b là hệ số phân nhánh và d là độ sâu của lời giải.

Một heuristic thường được đánh giá dựa trên hai tính chất sau:

- Tính chấp nhận được (Admissible): Một heuristic được gọi là admissible nếu nó không bao giờ đánh giá cao hơn chi phí thực tế để đạt được trạng thái đích. Nói cách khác, heuristic không được đánh giá quá cao. Một heuristic admissible đảm bảo rằng thuật toán tìm kiếm sẽ tìm được lời giải tối ưu hoặc lời giải không tệ hơn.
- Tính liên tục (Consistent): Một heuristic được gọi là consistent (hoặc monotonic) nếu giá trị heuristic của trạng thái kế tiếp không vượt quá giá trị heuristic của trạng thái hiện tại cộng với chi phí di chuyển từ trạng thái hiện tại đến trạng thái kế tiếp. Xác định tính chất consistent giúp đảm bảo rằng thuật toán tìm kiếm A Star sẽ tìm được lời giải tối ưu.

Dưới đây là hình ảnh mô tả từng bước thực hiện thuật toán A Star để tìm đường đi từ đỉnh 0 tới đỉnh 1:



2.3 So sánh các thuật toán

2.3.1 So sánh UCS, Greedy và A Star

Việc so sánh ba thuật toán UCS, Greedy và A Star có thể được tóm tắt trong bảng sau:

Thuật toán	Uninform cost search	Greedy	A Star
Loại thuật toán tìm kiếm	Không thông tin	Có thông tin	Có thông tin
Cách chọn mở node	Dựa trên chi phí đến trạng thái	Dựa vào Heuristic (Chi phí ước lượng đến đích)	Dựa trên tổng chi phí đến trạng thái hiện tại và Heuristic
Tính hoàn thiện	Có tính hoàn thiện	Không đảm bảo tính hoàn thiện	Có tính hoàn thiện
Tính tối ưu	Có tính tối ưu	Không đảm bảo tính tối ưu	Có tính tối ưu nếu có heuristic tốt

Bảng 1: So sánh UCS, Greedy và A Star

2.3.2 So sánh UCS và Dijkstra

Việc so sánh hai thuật toán UCS và Dijkstra có thể được trình bày trong báng sau:

Thuật Toán	Uninform cost search	Dijkstra
Mục tiêu	Tìm đường đi từ một điểm đến một điểm khác với chi phí nhỏ nhất	Tìm đường đi từ một điểm đến tất cả các điểm khác với chi phí nhỏ nhất
Yêu cầu về không gian	Tốn ít không gian hơn do hàng đợi ưu tiên được thêm vào dần dần	Tốn nhiều không gian hơn do tất cả các trạng thái được thêm vào từ đầu với chi phí vô cực
Thông tin về đồ thị	Không yêu cầu biết trước thông tin về đồ thị	Yêu cầu biết trước thông tin về đồ thị

Bảng 2: So sánh UCS và Dijkstra

3 Tài liệu tham khảo

- Artificial Intelligence: A Modern Approach (Fourth Edition).
- Note1 trên trang môn học.