**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

-----o0o----

Ảnh có chứa văn bản, thiết bị

Mô tả được tạo tự động

**[HCMUS][Introduction to AI][Lab1]**

**CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ**

**20120468 – Nguyễn Văn Hải**



Thành phố Hồ Chí Minh – 5/10/2022

1. **Vấn đề**

Sử dụng ngôn ngữ Python phiên bản 3.9.12 với các thư viện đinh kèm để giải quyết các bài toán tìm kiếm liên quan đến đồ thị.

Nêu các thành phần của một bài toán tìm kiếm và cách giải của phương pháp đó nói chung. Phân loại ra bài toán tìm kiếm (Uniformed Search và Informed Search).

Trình bày chi tiết về 4 thuật toán Breadth First Search (BFS), Depth First Search (DFS), Uniform-cost search (UCS) và A\* Search (A\*).

Cài đặt 4 thuật toán trên, đánh giá độ phức tạp về không gian và thời gian đồng thời kiểm tra sự hiệu quả, tính giới hạn của thuật toán.

1. **Giới thiệu chung.**
2. *Các thành phần của một bài toán tìm kiếm.*

Một bài toán tìm kiếm có thể được định nghĩa bằng 5 thành phần sau:

1. Trạng thái bắt đầu (Initial state).

Là điểm khởi đầu của một bài toán và yêu cầu lời giải của bài toán đó.

Trạng thái bắt đầu có thể được mô tả như sau In(Arad).

1. Mô tả các hành động (action).

Là các hành động có thể thực hiện được từ trạng thái bắt đầu qua các thời kì.

Mô tả: Action(In(Arad)) = {Go(S), Go(T), Go(Z)}.

1. Mô tả di chuyển (transition modle).

Mô tả kết quả của các hành động:

* Thuật ngữ **successor** tương ứng với các trạng thái có thể di chuyển được với một hành động duy nhất.
* Trạng thái bắt đầu, hành động và mô hình di chuyển định nghĩa **không gian trạng thái (state space)** của bài toán.
* Không gian trạng thái hình thành nên một **đồ thị** có hướng với đỉnh là các trạng thái và cạnh là các hành động.
* Một **đường đi** trong không gian trạng thái là một chuỗi các trạng thái được kết nối bằng một chuỗi hành động.

1. Kiểm tra đích (goal test).

Xác định trạng thái hiện tại xem có phải là đích hay không để đưa đến kết luận thuật toán kết thúc hay không.

1. Hàm chi phí đường đi (path cost)

Gán chi phí với trị số cho mỗi đường đi.

Chi phí đường đi khi thực hiện hành động *a* từ trạng thái *s* để trạng thái *s’* ký hiệu *c(s,a,s’)*

1. *Cách giải một bài toán tìm kiếm nói chung.*
2. Định nghĩa chung

Một lời giải (solution) là một chuỗi hành động di chuyển từ trạng thái bắt đầu cho đến trạng thái kết thúc.

Để giải một bài toán tìm kiếm ta thực hiện qua các bước sau:

* Bước 1: Xác định trạng thái bắt đầu **(Initial state)** của bài toán và yêu cầu kết thúc cuảt bài toán.
* Bước 2: Xác định thuật toán tìm kiếm càn áp dụng cho bài toán để tìm lời giải.
* Bước 2: Thực hiện lần lượt các hành động **(action)** để tạo nên mô hình di chuyển **(transition model)**, trong quá trình này cần chú đến chi phí đường **đi (path cost)**
* Bước 3: Kiểm tra đích **(goal test)** nếu sai thì tiếp tục thực hiện hành động, nếu đúng thì trả về kết quả => tạo ra một lời giải.

1. Các vấn đề giải quyết bài toán

***Giải quyết bài toán bằng phương pháp tìm kiếm:***

* Định nghĩa: quy trình tìm kiếm chuỗi hành động để tới được trạng thái đích.
* Một thuật toán tìm kiếm nhận một bài toán là input trả về kết quả là lời giải dưới dạng chuỗi hành động.
* Thực thi: khi đã tìm thấy lời giải, thực hiện các hành động:
* Khi thực hiện lời giải, không nhận tín hiệu trả về từ các hành động.
* Hệ thống open-loop

***Cây tìm kiếm:***

Một lời giải là một chuỗi hành động, các thuật toán tìm kiếm thực hiện chiến lược tìm kiếm bằng cách xem xét nhiều chuỗi hành động khả thi.

* **Cây tìm kiếm**: chuỗi hành động xuất phát từ đỉnh bắt đầu.
  + **Nhánh** là các hành động và các nút tương ứng với các trạng thái trong không gian tìm kiếm.
  + **Nút gốc** tương ứng với trạng thái bắt đầu.
  + Thực hiện hành động bằng cách mở trạng thái hiện tại (**nút cha**), sinh tập các trạng thái mới (**các nút con**)  
    **Biên (Frontier):** tập các nút lá có thể mở rộng tại một thời điểm.
* **Gốc** là trạng thái bắt đầu.
* **Một nút trên cây** không chỉ là một trạng thái mà còn là **một kế hoạch.**
* **Lưu ý:** một trạng thái có thể xuất hiện nhiều lần trong cây

***Đường đi dư thừa:***

* **Đường đi dư thừa** là không thể tránh khỏi, tồn tại khi nhiều hơn một đường đi giữa các trạng thái.

Chart, radar chart

Description automatically generated

* Di chuyển trên các đường đi dư thừa sẽ làm quy trình tìm kiếm mất kiểm soát.
  + Điều này vẫn đúng ngay cả với các thuật toán có khả năng tránh các vòng lặp vô tận.

***Kiến trúc dữ liệu:***

Thuật toán tìm kiếm cần phải có một kiến trúc dữ liệu để lưu vết quá trình xây dựng cây tìm kiếm. Với mỗi nút n trên cây tìm kiếm, ta cần phải lưu giữ:

* Trạng thái (state): trạng thái tương ứng với nút n.
* Cha (parent): nút cha của nút n.
* Hành động (action): hành động thực hiện với parent(n) để di chuyển đến n.
* Chi phí (path-cost): chi phí di chuyển từ trạng thái bắt đầu đến n, thường được ký hiệu g(n).

***Đánh giá thuật toán***

Độ phức tạp về mặt thời gian và không gian có thể đánh giá thông qua:

**b**: số nhánh tối đa của cây.  
**d**: chiều sâu thấp nhất của lời

Chart

Description automatically generated

giải.  
**m**: chiều sâu tối đa của không

gian tìm kiếm (có khả năng ∞)

Một lời giải tối ưu có chi phí đường đi thấp nhất trong tất cả các lời giải

1. *Phân loại bài toán tìm kiếm.*

Bài toán tìm kiếm được chia thành 2 loại cơ bản sau:

1. Uniformed search (tìm kiếm mù)

Uniformed search (tìm kiếm mù) là chiến lược tìm kiếm mà chúng ta không có hiểu biết nhiều về các đối tượng để có thể hướng dẫn tìm kiếm mà chỉ đơn thuần xem xét các đối tượng theo một hệ thống nào đó để phát hiện ra đối tượng cần tìm kiếm.

Đại diện cho Uniformed search là các thuật toán sau:

* Breadth-first Search (BFS) là thuật toán tìm kiếm theo kiểu mở rộng.
* Deapth-first Search (DFS) là thuật toán tìm kiếm dựa vào chiều sâu.
* Uniform-cost Search (UCS) là thuật toán tìm kiếm dựa vào chi phí di chuyển.

1. Informed search (tìm kiếm có định hướng)

Informed Search (tìm kiếm có định hướng) là chiến lược tìm kiếm mà lúc này chúng ta có thông tin vè đối tượng, từ đó xây dựng các hàm đánh giá (hay còn gọi là heuristic). Thuật toán Informed search phổ biến nhất hiện nay là A\*.

Các điểm nổi bật của tìm kiếm có định hướng:

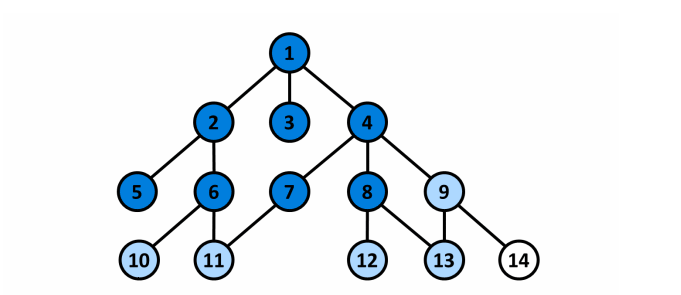
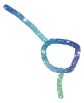
* Bên cạnh định nghĩa còn sử dụng tri thức cụ thể vào bài toán.
* Có khả năng tìm lời giải hiệu quả hơn so với các chiến lược tìm kiếm mù.

1. **Tìm hiểu và ý tưởng chung của các thuật toán**
2. *Breadth First Search (BFS)*
   1. Giới thiệu

Thuật toán **duyệt đồ thị ưu tiên chiều rộng** (Breadth-first search - BFS) là một trong những thuật toán tìm kiếm mù cơ bản và thiết yếu trên đồ thị. Mà trong đó, những đỉnh nào gần đỉnh xuất phát hơn sẽ được duyệt trước.

Tất cả các nút ở một độ sâu nhất định phải được mở trước khi mở  
các nút ở độ sâu tiếp theo.

* 1. Quy trình





Start = 1

Goal = 8

Start là 1 thì ta mở biên là tập hơp điểm {2,3,4} => sau khi mở lần lượt biên từ 2 đến 4 ta có biên mới là tập hơp {5,6,7,8,9} lúc này thuật toán BFS sẽ mở lần lượt 5 dến 8 => tìm thấy giá trị và kết thúc thuật toán.

* 1. Đánh giá thuật toán

1. Ưu điểm

* Nếu có nhiều hơn một giải pháp cho một vấn đề nhất định, thì BFS sẽ cung cấp giải pháp tối thiểu yêu cầu số bước ít nhất.

1. Nhược điểm

* Nó yêu cầu nhiều bộ nhớ vì mỗi cấp độ của cây phải được lưu vào bộ nhớ để mở rộng cấp độ tiếp theo.
* BFS cần nhiều thời gian nếu giải pháp ở xa nút gốc.

1. Độ phức tạp

Đồ thị đang xét có: d, m là chiều sâu của đồ thị và điểm cần tìm, b là nhân tố nhánh.

Độ phức tạp về thời gian: O(bd).

Độ phức tạp về không gian: O(bd).

1. Tính hoàn thành, tính tối ưu và tính đầy đủ

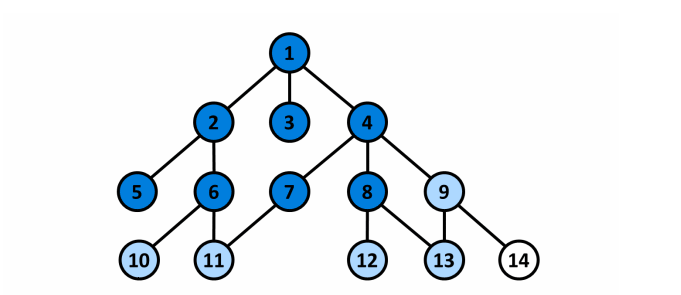
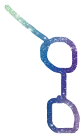
Tính hoàn thành: bài toán hoàn thành nếu ta có b hữu hạn.

Tính tối ưu: chi phí di chuyển như nhau.

1. *Depth First Search (DFS)*
   1. Giới thiệu

Thuật toán duyệt dồ thị ưu tiên chiều sâu (Depth First Search – DFS) là một trong những thuật toán tìm kiếm mù cơ bản của đồ thị. Thuật toán sẽ duyệt từ các nút sâu nhất trước.

* 1. Quy trình





Start = 1

Goal = 8

Mở các biên {2,3,4} => mở biên {5} lúc này không duyệt được nữa bỏ qua mở biên {6} rồi mở biên {11} => quay lại biên {3} => quay lại {4} duyệt đến {7} => duyệt qua {8} => dừng lại mà tìm thấy kết quả.

* 1. Đánh giá

1. Ưu điểm

* DFS yêu cầu rất ít bộ nhớ vì nó chỉ cần lưu trữ một chồng các nút trên đường dẫn từ nút gốc đến nút hiện tại.
* Mất ít thời gian hơn để đến được nút mục tiêu so với thuật toán BFS (nếu nó đi đúng đường).

1. Nhược điểm

Có khả năng nhiều trạng thái tiếp tục tái diễn và không có gì đảm bảo cho việc tìm ra giải pháp. Thuật toán DFS dùng để tìm kiếm sâu hơn và đôi khi nó có thể đi đến vòng lặp vô hạn.

1. Độ phức tạp

Độ phức tạp về thời gian: O(bm ).

Độ phức tạp về không gian: O(bm).

1. Tính hoàn thành, tính tối ưu và tính đầy đủ

Tính hoàn thành (complete): có nếu không gian hữu hạn.

Tính tối ưu (optimal): Tối ưu nhất tại lời giải “phải nhất”

1. *Uniform-cost search (UCS)*
   1. Giới thiệu

Uniform-cost search (USC) là thuật toán tìm kiếm mù cổ điển trên đồ thị mà nó quan tâm tới chi phí di chuyển.

Thuật toán ưu tiền tìm đường đi ngắn nhất từ điểm bắt đầu đến điểm kết thúc.

Hàm đánh giá f(n) = g(n) với g(n) là chi phí di chuyển.

* 1. Quy trình

Diagram

Description automatically generated

Start = v1 và Goal = v7

F1 = {v2 = 5; v3 = 31; v4 = 40}

F2 = {v3 = 31; v3(v2) = 32; v4 = 40; v5(v2) = 78}

F3 = {v4(v3) = 39; v4 = 40; v6( v3) = 56; v2(v3) = 58; v7(v3) = 69; v5(v3) = 80}

F4 = {v6(v4) = 55; v6(v3) = 56; v7(v3) = 69; v5(v3) = 80}

F5 = v7(v6) = 67; v7(v3) = 69; v5(v3) = 80}

Đường đi dựa trên thuật toán UCS là: v1 => v3 => v4 => v6 => v7

* 1. Đánh giá

1. Ưu điểm

* Tối ưu ở mọi trạng thái, con đường có chi phí thấp nhất được chọn.
* Thuật toán luôn tìm thấy kết quả.

1. 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.

1. Độ phức tạp

Với chi phí di chuyển thấp nhất là €, C\* là chi phí lời giải tối ưu.

Độ phức tạp theo không gian: O(b1+[ C\* / €] )

Độ phức tạp theo thời gian: O(b1+[ C\* / €] )

1. Tính hoàn thành, tính tối ưu và tính đầy đủ.

Tính hoàn thành (complete): luôn tìm thấy giá trị.

Tính tối ưu (optimail) : có.

Tính đầy đủ: có.

1. *A\* Search (A\*)*
   1. Giới thiệu

Thuật toán tìm kiếm bắt đầu tại nút gốc và duyệt tiếp tục bằng cách đi tiếp các nút có mức ước lượng chi phí thấp nhất.

A\* search là dạng tìm kiếm Best-first Search phổ biến nhất.

Tích hợp heuristic vào quá trình tìm kiếm, tránh các đường đi có chi phí lớn.

* 1. Quy trình

Thuật toán A\* tìm kiếm dựa trên hàm đánh giá f(n) = g(n) + h(n) trong đó:

* G(n) là chi phí đường đi tới đích.
* H(n) là ước tính tới đích.
* Và F(n) là ước tính chi phí.

Diagram, shape

Description automatically generated

Start: A

Goal: E



F1 = {C: 5 + 10; B: 10 + 21}

F2 = {D(C) = 12 + 5; B: 10 + 21}

F3 = {B: 10 + 21; E(D): 26}

F4 = {E(B): 25; E(D): 36}

Đường đi dựa vào thuật toán A\*: A => B => E.

* 1. Đánh giá

1. Ưu điểm

* Thuật toán tìm kiếm A \* là thuật toán tối ưu và hoàn chỉnh. Tốt nhất so với các thuật toán tìm kiếm khác và có thể giải quyết các vấn đề rất phức tạp.
* A\* mở theo các hướng chính tới kết quả => tiết kiệm được không gian.

1. Nhược điểm

* Nó không phải lúc nào cũng tạo ra đường đi ngắn nhất vì nó chủ yếu dựa trên heuristics và tính gần đúng.
* Thuật toán tìm kiếm A \* có một số vấn đề phức tạp.
* Hạn chế chính của A \* là yêu cầu bộ nhớ vì nó giữ tất cả các nút được tạo trong bộ nhớ, vì vậy nó không thực tế cho các vấn đề quy mô lớn khác nhau.

1. Độ phức tạp

Với chi phí di chuyển thấp nhất là €, C\* là chi phí lời giải tối ưu.

Độ phức tạp thời gian: cấp số mũ.

Độ phức tạp về không gian: cấp số mũ.

1. Tính hoàn thành, tính tối ưu và tính đầy đủ

Tính hoàn thành (complete): hoàn thành nếu € > 0 và không gian trạng thái hữu hạn.

Tính tối ưu (optimal): Nếu có heuristic hợp lí hoặc nhất quán.

Một heuristic là hợp lí khi:

* Không ước lượng quá cao chi phí đến đích.
* H(n) ≤ h\* (n) chi phí thấp nhất từ n đến đỉnh.

Một heuristic là nhất quán khi:

* h(n) là một heuristic nhất quán nếu với mỗi succesor n′ của n, khoảng  
  cách ước tính đến đích từ n không lớn hơn khoảng cách đến đích ước  
  tính từ n′ cộng với chi phí di chuyển từ n đến n′: h(n) ≤ c(n, a, n′) +  
  h(n′).

Tính dầy đủ: có

1. **So sánh các thuật toán**
2. *Sự khác biệt giữa UCS, Greedy và A\**

|  |  |  |  |
| --- | --- | --- | --- |
| Tiêu chí | UCS | Greedy | A\* |
| Phân loại | Uniformed Search. | Informed Search. | Informed Search. |
| Chiến lược | Bài toán chỉ quan tâm đến chi phí di chuyển. | Là một thuật toán giải quyết dựa theo kiểu metaheuristic để tìm kiếm lựa chọn tối ưu ở địa phương ở mỗi bước hy vọng tìm được tối ưu toàn cục. | Bài toán quan tâm đến chi phí di chuyển và heuristic ước lượng tới đích của điểm đó. |
| Độ phức tạp thời gian | O(b1+[ C\* / €] ) |  | Cấp số mũ. |
| Độ phức tạp không gian | O(b1+[ C\* / €] ) |  | Cấp số mũ. |
| Tính tối ưu | Có | Đối với nhiều bài toán, hầu như không cho ra lời giải tối ưu. | Có nếu heuristic là hợp lí hoặc nhất quán. |
| Tính hoàn thành | Luôn tìm thấy. | Có. | Có nếu không gian là hữ hạn, và € > 0. |

1. *Sự khác biệt giữa UCS và Dijkstra*

aaaaaa

1. **Cài đặt thuật toán và mã giả**
2. *Breadth First Search (BFS)*
   1. Mă giả

g:graphs, open\_set:[g.start], close\_set:[]

def BFS\_algorithm(g, open\_set, close\_set):

    currentList <- close\_set

    while currentList != NULL:

        curNode <- currentList[0]

        if curNode == ggGoal:

            #set color path

            return

        close\_set.append(curNode)

        open\_set.remove(curNode)

        neighbors = curNode.get\_neighbors()

        while neighbors != NULL:

            if (neighbors[0] not in open\_set) and (neighbors[0] not in close\_set):

                open\_set.append(neighbors[0])

            neighbors.remove(neighbors[0])

    BFS\_algorithm(g, open\_set, close\_set)

Ta vào terminal gõ: python .\main.py --algo BFS --start 71 --goal 231

Kết quả trên màn hình:

Background pattern

Description automatically generated

Giải thích:

* Open\_set sẽ lấy giá trị start được truyền vào graphs.
* Vòng lặp đầu tiên currentList (tập các node đang được mở) sao chép toàn bộ giá trị của open\_set (lúc này là start)
* Kiểm tra nếu currentList khác null thì:
  + Xóa node đang xét khỏi open\_set và thêm vào closed\_set.
  + Duyệt lần lượt từng Node trong list đang xét. Nếu có Node nào bằng KQ => return.
  + Nếu không thì xét các biên của Node đó. Nếu các Nút biên không nằm trong open\_set và closed set thì thêm vào open set.
* Đệ quy lại Hàm BFS.

* 1. Ví dụ

1. *Depth First Search (DFS)*
   1. Mã giả
   2. Ví dụ
2. *Uniform-cost Search (UCS)*
   1. Mã giả
   2. Ví dụ
3. *A\* Search (A\*)*
   1. Mã giả
   2. Ví dụ