TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC

BÁO CÁO ĐỒ ÁN

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Sinh viên thực hiện:	Trịnh Minh Anh	MSSV: 21280005
	Nguyễn Lưu Phương Ngọc Lam	MSSV: 21280096
Lớp:	21KDL1	
Giảng viên lý thuyết:	Bùi Tiến Lên	
Giảng viên thực hành:	Nguyễn Bảo Long	

Thành phố Hồ Chí Minh, 12-2022

Mục lục

LỜI CẢM ƠN

Trước tiên, chúng em xin đươc gửi một lời cảm ơn sâu sắc và lòng biết ơn chân thành nhất dành đến cho thầy Nguyễn Bảo Long vì đã chuẩn bị và đầu tư những bài giảng tâm huyết, nhằm truyền đạt những kiến thức lập trình bổ ích mà không kém phần thú vị trong suốt thời gian thưc học môn **Thực hành Cấu trúc dữ liệu và Giải thuật** vừa qua.

Trong quá trình nghiên cứu về các thuật toán tìm kiếm và sắp xếp, vì lí do thời gian và trình độ còn nhiều hạn chế, nhóm chúng em sẽ không thể tránh khỏi những thiếu sót. Như Socrates từng nói *"Tôi chỉ biết một điều là tôi không biết gì cả"* . Trên tinh thần cầu thị và sẵn sàng tiếp thu, ghi nhận mọi đóng góp để kịp thời sửa đổi, nhóm chúng em kính mong nhận đươc sư chỉ bảo và ý kiến đến từ quý thầy cô để có thể hoàn thiên sản phẩm của mình hơn.

Chúng em xin chân thành cảm ơn!

TỔNG QUAN

Tên đồ án: Tìm hiểu và cài đặt các thuật toán tìm kiếm trên đồ thị

Giảng viên hướng dẫn: Nguyễn Bảo Long

Các thành viên trong nhóm:

1. Trịnh Minh Anh (MSSV: 21280005)
2. Nguyễn Lưu Phương Ngọc Lam (MSSV: 21280096)

Mục tiêu đồ án:

Nghiên cứu, cài đặt và trình bày các thuật toán tìm kiếm trên đồ thị.
Minh hoạ thuật toán thông qua các kỹ thuật đồ hoạ bằng ngôn ngữ Python.

CHƯƠNG 1. GIỚI THIỆU ĐỒ ÁN

Cấu trúc dữ liệu đồ thị đã và đang được ứng dụng rộng rãi trong lĩnh vực máy tính. Các thuật toán tìm kiếm trên đồ thị có ý nghĩa đặc biệt quan trọng trong việc cài đặt, xử lý và ứng dụng đồ thị trong từng lĩnh vực chuyên biệt trong ngành khoa học máy tính. Ngoài ra, kiến thức về đồ thị cũng như các thuật toán tìm kiểm rất hữu ích trong các các ứng dụng tìm đường đi tối ưu phổ biến như Google Maps, BusMap...

Vì lẽ đó, đề tài này đem đến cho độc giả những thuật toán tìm kiếm thường được sử dụng trên đồ thị một cách tổng quát, chuyên sâu và dễ nắm bắt nhất.

CHƯƠNG 2. CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

2.1 Thuật toán tìm kiếm theo chiều rộng - Breadth First Search

BFS là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị theo chiều rộng, thuật toán đi qua tất cả các nodes/đỉnh có cùng level (theo thứ tự từ trái sang phải) trước khi chuyển sang level kế tiếp.

Thuật toán có thể được cài đặt bằng giải thuật đệ quy hoặc sử dụng cấu trúc dữ liệu là Queue (hàng đợi)

Ý tưởng thuật toán: Thuật toán có chiến lược tìm kiếm đơn giản bằng việc xây dựng cây tìm kiếm theo chiều rộng, node gốc sẽ được duyệt đầu tiên, sau đó tất cả những node con của node gốc sẽ được lưu trữ trong queue và sau đó sẽ được duyệt lần lượt theo queue. Cứ tiếp tục như vậy, các node con của những node con đó sẽ được duyệt.

Tổng quát: Tất cả các node ở độ sâu d trong cây tìm kiếm sẽ được duyệt trước những node ở độ sâu $d+1$

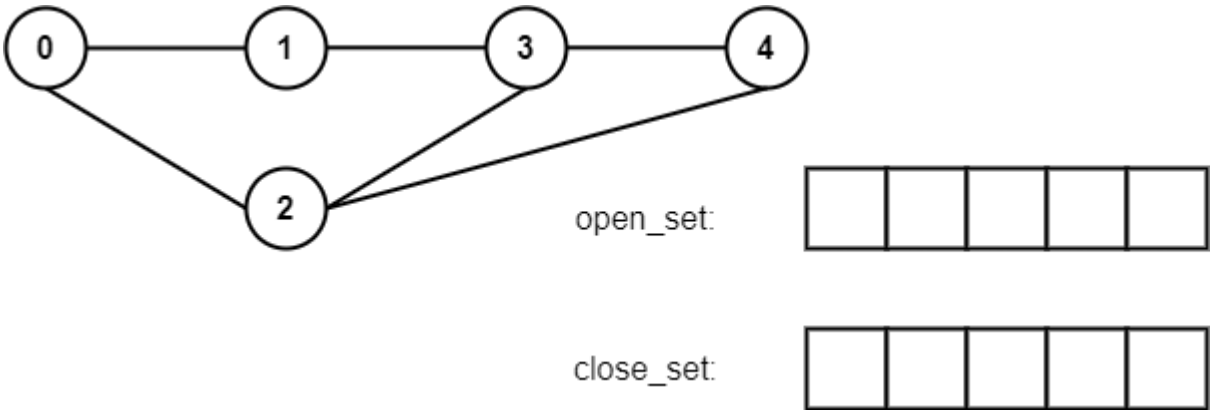
Mã giả:

```
    BFS
input: graph, start_node, end_node
output: the path from start_node to end_node

#create 2 empty arrays

open_set = [] # contains all the neighbours node
close_set = [] # contains all the visited node

draw_path = []

#input the start_node into open_set
open_set.add(start_node)

while true do
   if open_set is empty
      display "No path found"
      break
   end if

   current_node = remove the first element from open_set
   assign the current_node with yellow color
```

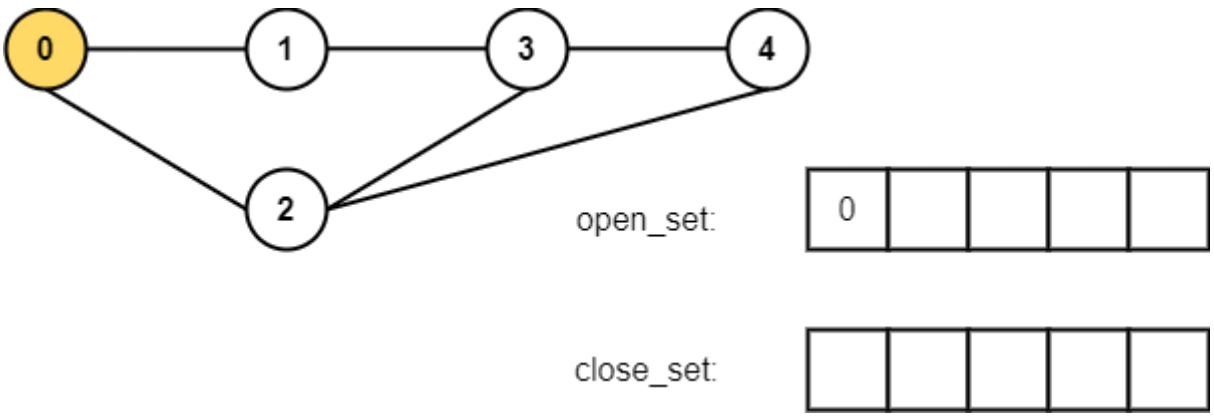
```
#input the current_node into close_set
close_set.add(current_node)

for each node in neighbours of current_node
   if node not in close_set
      marked the node with red color
      input the node into open_set
   end if
end for

marked the last element of close_set with blue color and save it in
    the draw_path

if current_node == end_node
   return draw_path
```

Minh họa thuật toán:

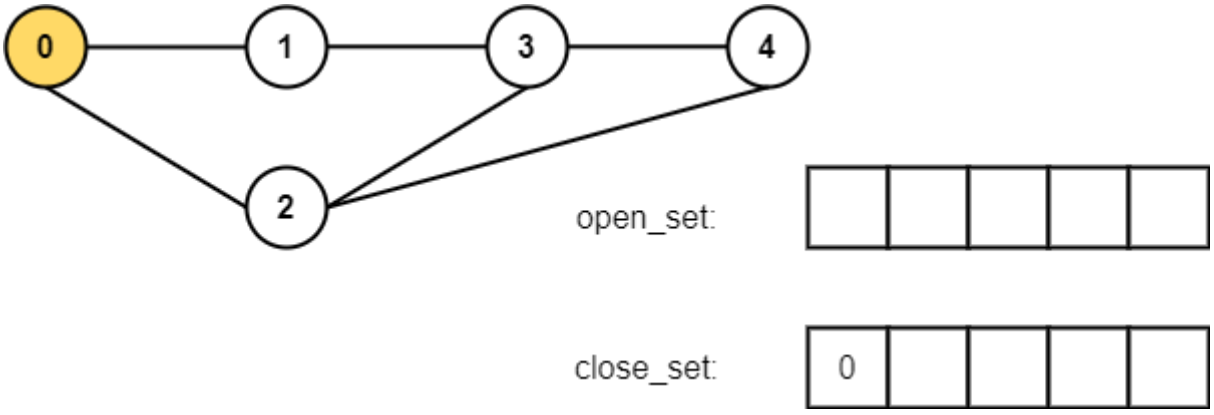
Chúng ta có đồ thị vô hướng như trên và hai mảng rỗng là $open-set$ và $close-set$



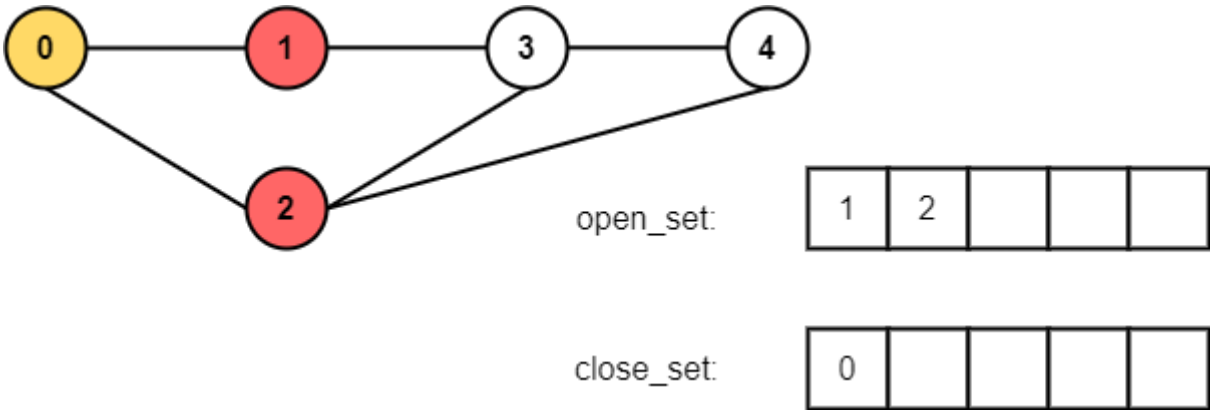
Đầu tiên, ta chọn start-node ở đây sẽ là node 0, sau đó ta input node 0 vào mảng $open-set$



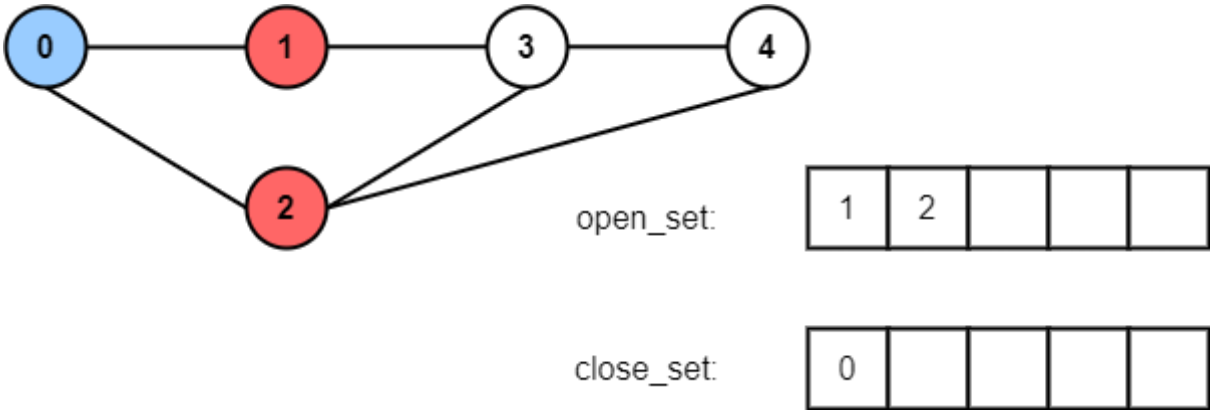
Tạo một biến current-node gán với giá trị đầu tiên của mảng $open - set$. đánh dấu current-node là màu vàng. Tiếp đến ta sẽ input vào $close - set$ giá trị của current-node như hình dưới.



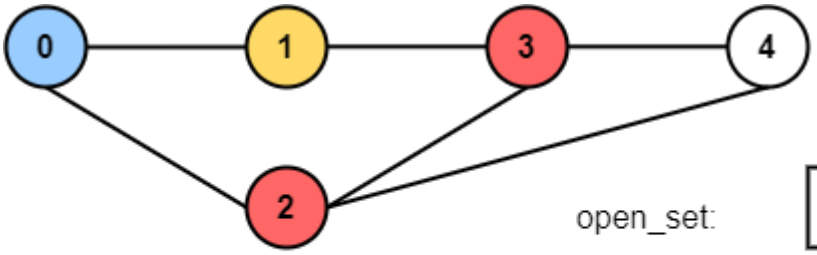
Xác định lần lượt các neighbours-node của current-node. Nếu neighbours-node chưa nằm trong $close - set$ thì ta sẽ thêm vào $open - set$. Và đánh dấu các neighbours-node là màu đỏ.



Sau đó ta đánh dấu phần từ cuối cùng của $close - set$ bằng màu xanh.

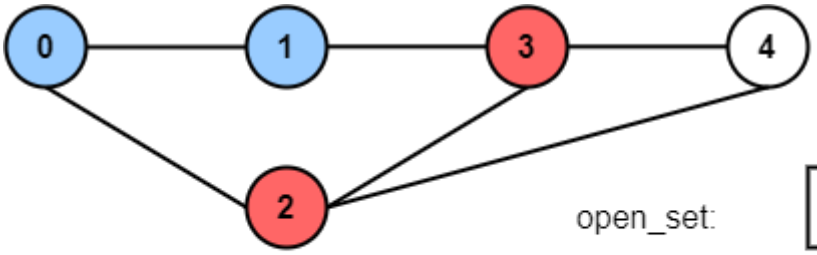


Tiếp đến, ta sẽ thực hiện lại các bước xác định current-node như trên.



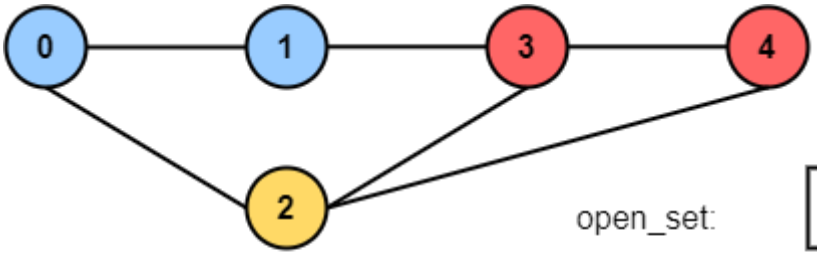
open_set: | 2 | 3 | | | |

close_set: | 0 | 1 | | | |



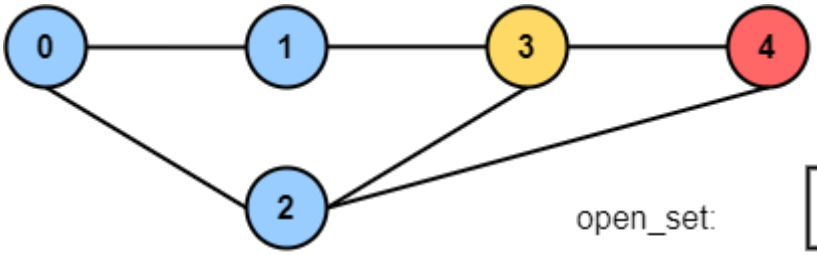
open_set: | 2 | 3 | | | |

close_set: | 0 | 1 | | | |



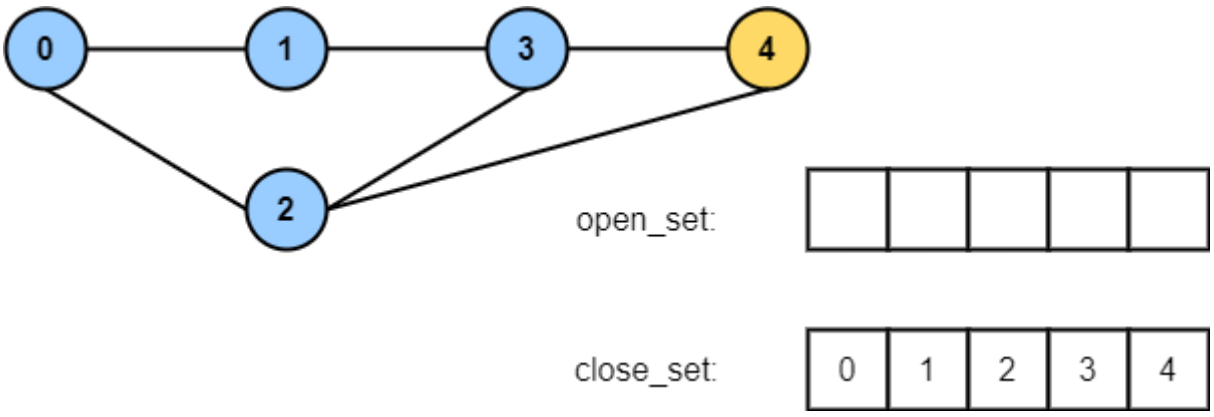
open_set: | 3 | 4 | | | |

close_set: | 0 | 1 | 2 | | |

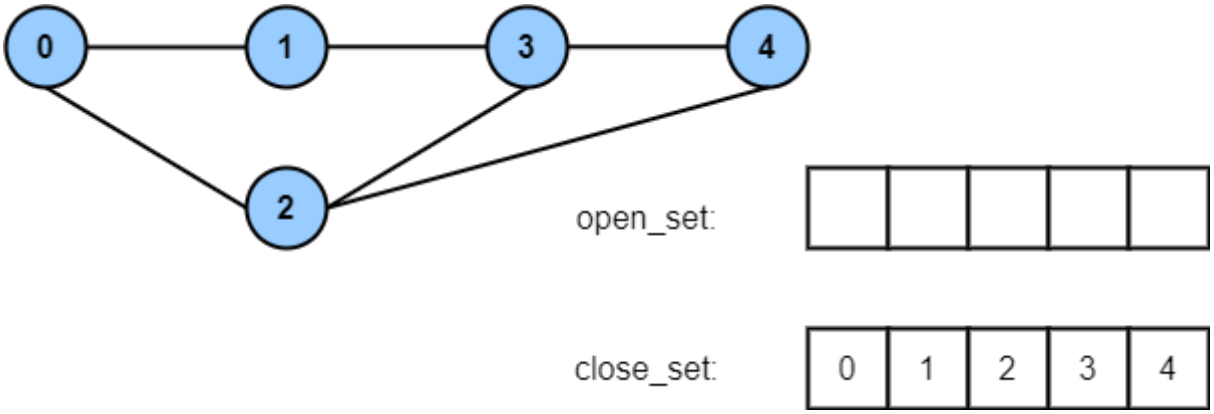


open_set: | 4 | | | | |

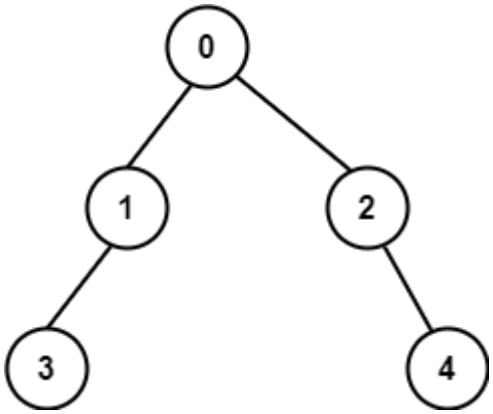
close_set: | 0 | 1 | 2 | 3 | |



Sau khi đã duyệt hết đồ thị (khi $open - set$ trở thành mảng rỗng), ta sẽ có giá trị thứ tự duyệt trong $close - set$ là 0, 1, 2, 3, 4



Chúng ta có minh họa dưới dạng cây của BFS như sau:



Đánh giá thuật toán: Để đánh giá thuật toán nào là thích hợp nhất, chúng ta có thể xem xét các yếu tố sau:

Tính đầy đủ: BFS có tính đầy đủ. Nếu node ban đầu nông nhất ở độ sâu d, sau khi duyệt tất cả các node nông hơn d, nó cũng sẽ tìm thấy nó.

Tính tối ưu: Tối ưu cho đồ thị không có trọng số, nhưng trường hợp tổng quát thì không tối ưu.

Độ phức tạp: Với b là số node con của những node không phải đích đến, d là độ sâu khi tìm thấy đích.

1. Về không gian: Tất cả các nút cho đến $d-1$ được ghi lại trong tập đóng để tránh trường hợp xét các đường đi lặp lại gây lãng phí tài nguyên.

   Ta có: $b^1+b^2+b^3+...+b^d = O(b^d)$ ($O(b^{d-1})$ cho những đỉnh đã được duyệt, và $O(b^d)$ cho hàng đợi)

2. Về thời gian: Ta có: $1+b+b^2+b^3+...+b^d = O(b^d)$

2.2 Thuật toán tìm kiếm theo chiều sâu - Depth First Search

DFS là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị theo chiều sâu: Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Thuật toán sẽ sử dụng cấu trúc dữ liệu là Stack

Ý tưởng thuật toán: DFS sẽ bắt đầu ở node gốc và mở rộng, duyệt càng xa càng tốt dọc theo mỗi nhánh trước khi thực hiện backtracking.

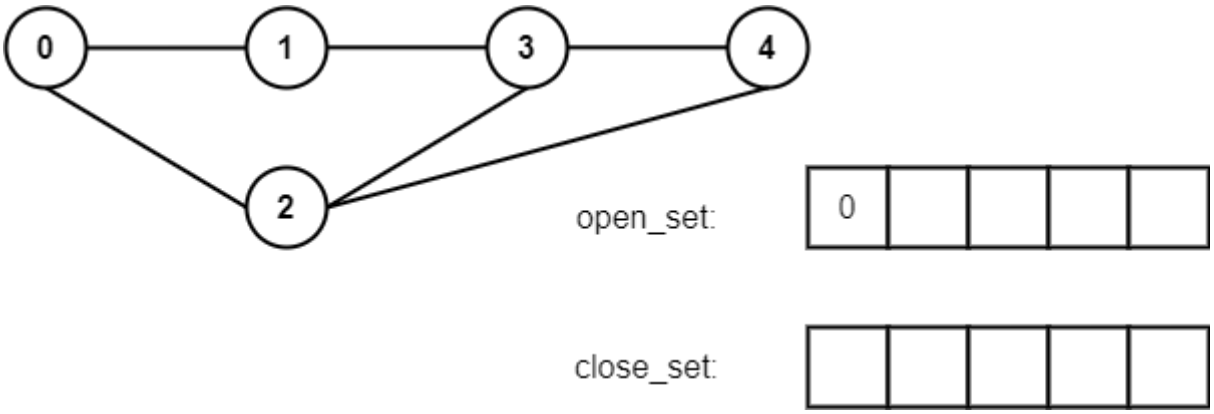
Mã giả:

```
    DFS
input: graph, start_node, end_node
output: the path from start_node to end_node

#create 2 empty arrays

open_set = [] # contains all the neighbours node
close_set = [] # contains all the visited node

draw_path = []

#input the start_node into open_set
open_set.add(start_node)

while true do
   if open_set is empty
      display "No path found"
      break
   end if

   current_node = remove the last element from open_set
   assign the current_node with yellow color
```

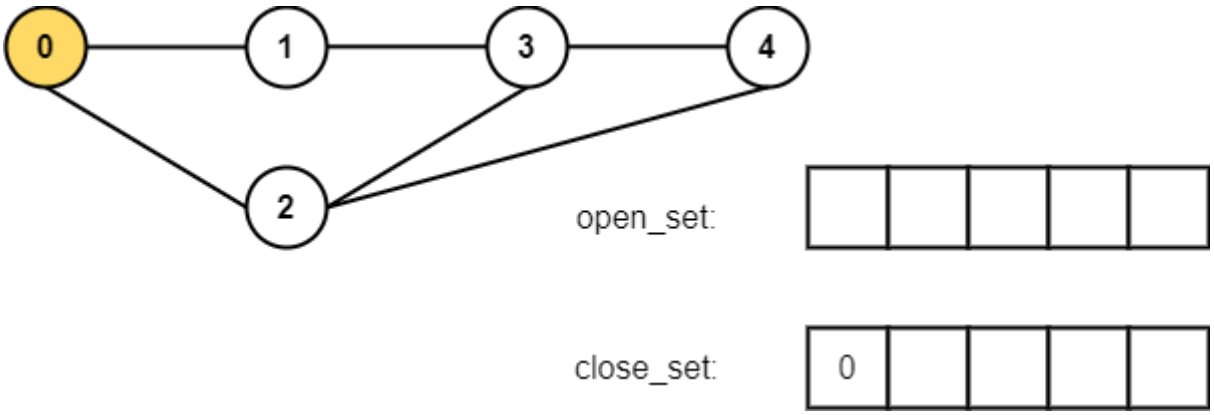
```
#input the current_node into close_set
close_set.add(current_node)

for each node in current_node neighbours
   if node not in close_set
      marked the node with red color
      input the node into open_set
   end if
end for

marked the last element of close_set with blue color and save it into
    the draw_path

if current_node == end_node
   return draw_path
```

Minh họa thuật toán:

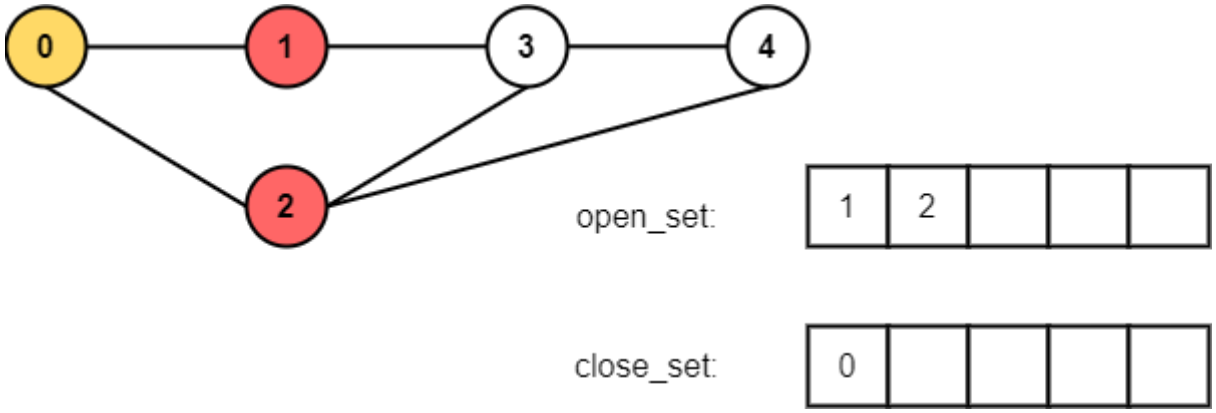
Chúng ta co đồ thị như hình và hai mảng rỗng là: *open – set* và *close – set*



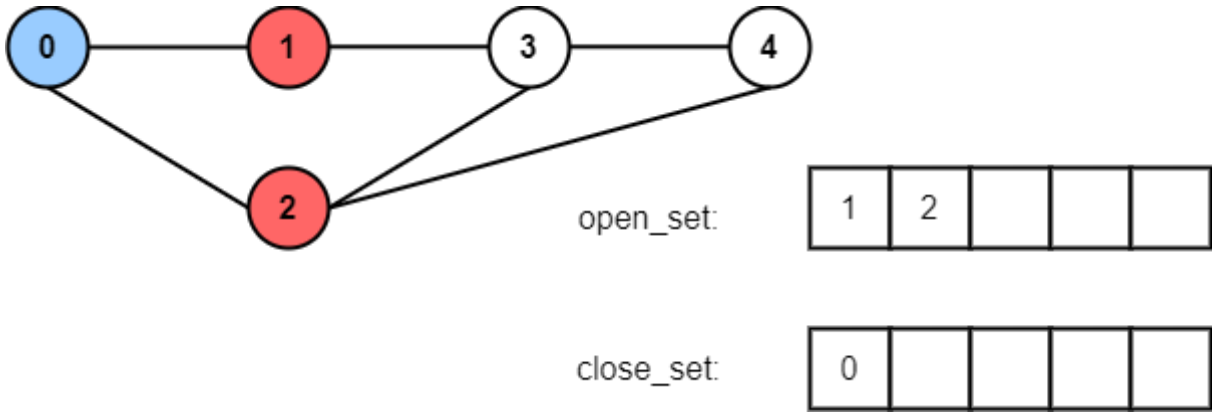
Đầu tiên, ta chọn start-node ở đây sẽ là node 0, sau đó ta input node 0 vào mảng *open – set*



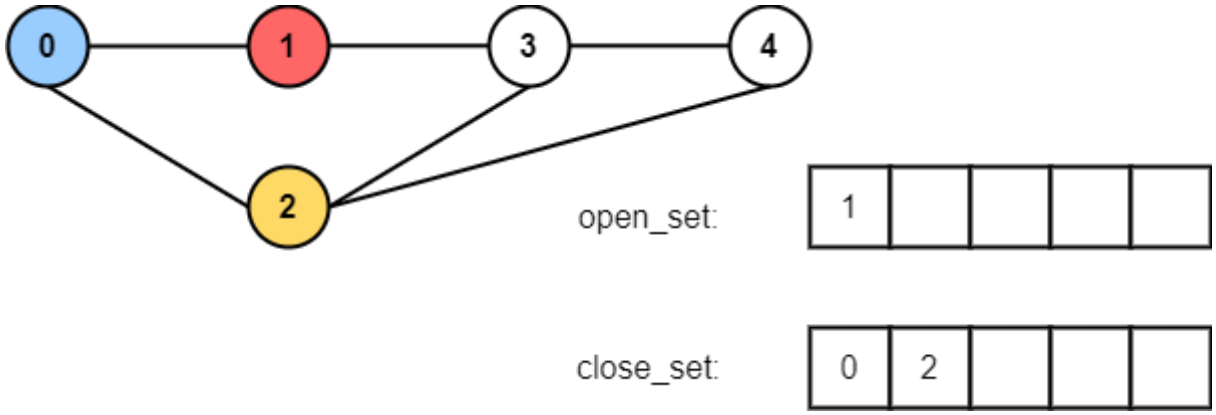
Tạo một biến current-node gán với giá trị cuối cùng của mảng $open - set$. đánh dấu current-node là màu vàng. Tiếp đến ta sẽ input vào $close - set$ giá trị của current-node. Đồng thời, xác định các neighbours-node của current-node như hình dưới. Ta thấy node 1 và node 2 không nằm trong $close - set$ nên ta sẽ đánh dấu màu đỏ và input chúng vào $open - set$

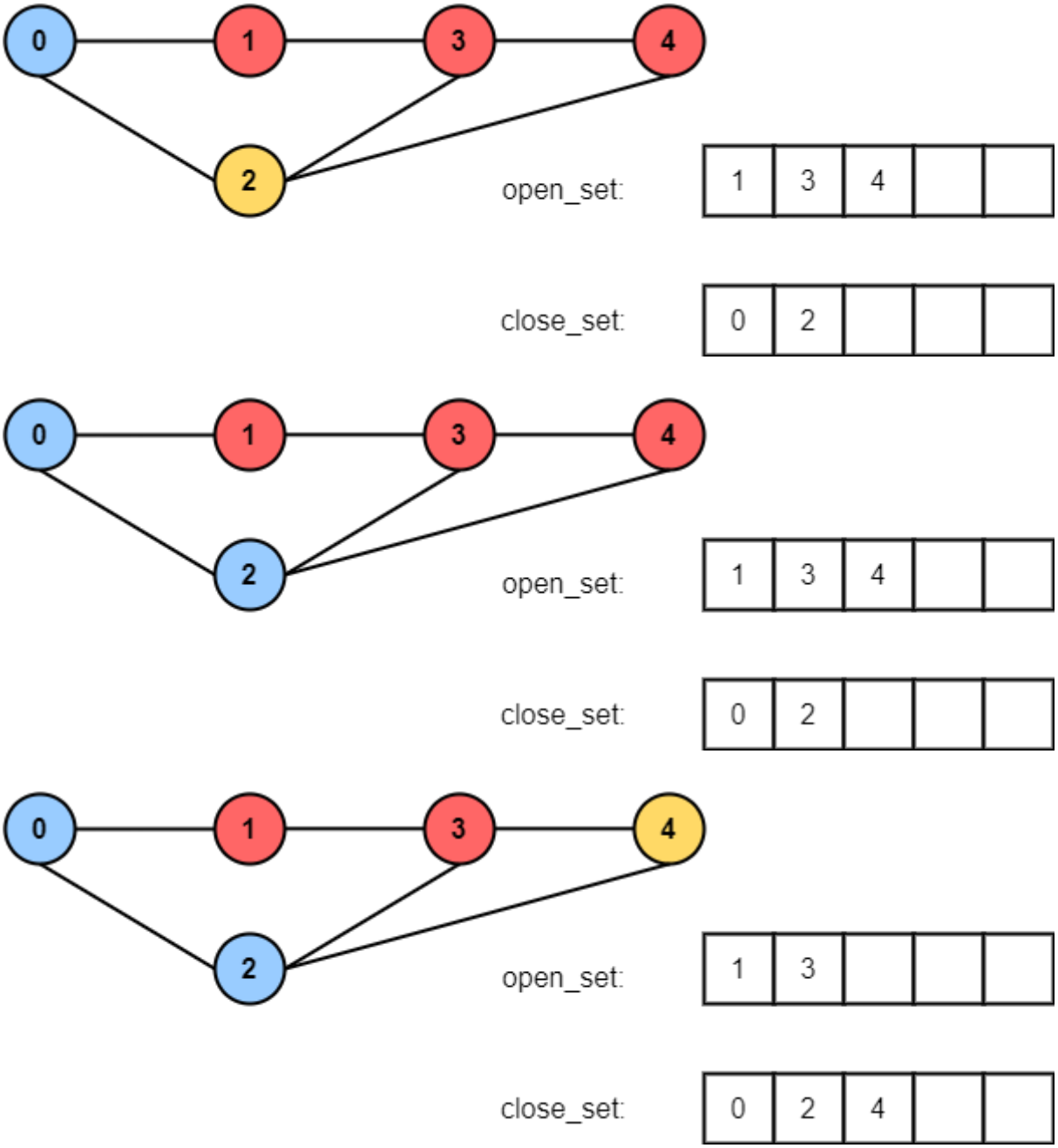


Tiếp đến, ta sẽ đánh dấu phần tử cuối cùng trong $close - set$ bằng màu xanh

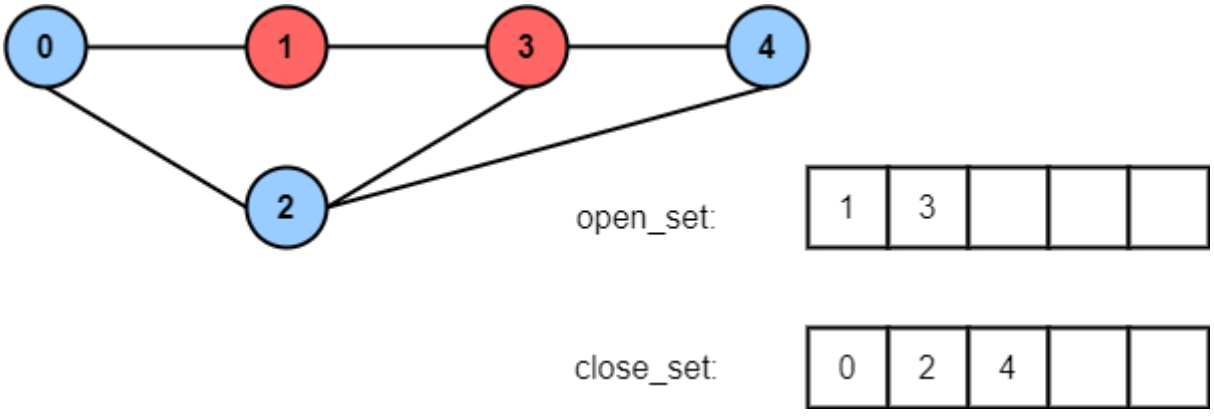


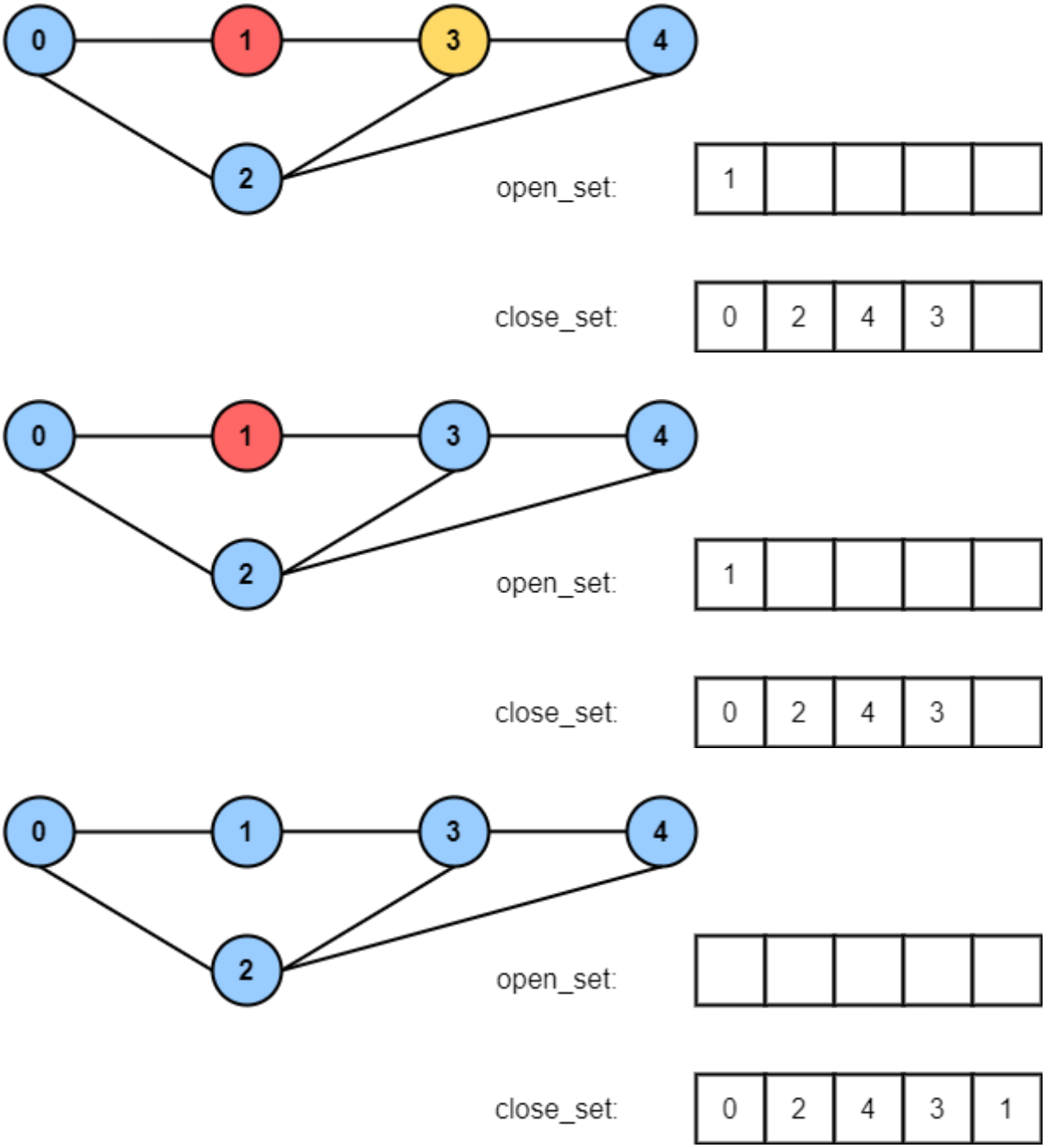
Tiếp tục thực hiện việc xác định current-node là giá trị cuối cùng của $open - set$ và tiếp tục các bước như trên.



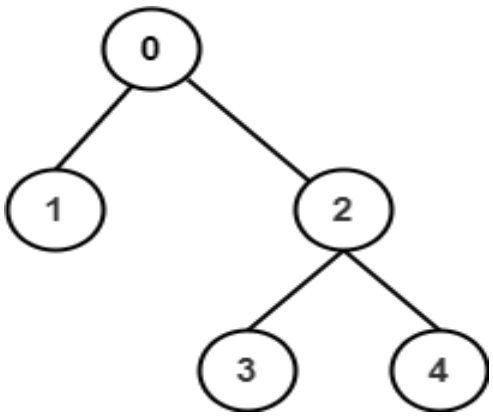


Ở đây, ta thấy neighbours-node của node 4 gồm node 3 và node 2, nhưng node 2 đã nằm trong $close - set$, nên ở bước này, ta sẽ không input lại node 2 vào trong $open - set$.





Và cuối cùng ta sẽ có đồ thị cây duyệt theo DFS như hình.



Đánh giá thuật toán: Đầu vào DFS được giả định là một đồ thị hữu hạn, được biểu diễn một cách rõ ràng dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự.

Tính đầy đủ: DFS không đảm bảo tính đầy đủ nếu không gian tìm kiếm là vô hạn, khi không gian tìm kiếm hữu hạn thì ngược lại.

Tính tối ưu: DFS không đảm bảo tính tối ưu.

Độ phức tạp: Với b là số nút con của những nút không phải đích đến, d là độ sâu khi tìm thấy đích.

1. Về không gian: Gần như là tuyến tính $O(bm)$
2. Về thời gian: Tăng theo hàm mũ $O(b^m)$, trường hợp xấu là khi m lớn hơn nhiều lần so với độ sâu d.

2.3 So sánh giữa BFS và DFS

Đặc điểm	BFS	DFS
Định nghĩa	Là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị theo chiều rộng: Thuật toán đi qua tất cả các nodes/đỉnh có cùng level (theo thứ tự từ trái sang phải) trước khi chuyển sang level kế tiếp.	Là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị theo chiều sâu: Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.
Ý tưởng	Sử dụng để tìm đường đi ngắn nhất trong đồ thị không có trọng số vì BFS tiến tới một đỉnh với số lượng cạnh kề là nhỏ nhất (tính từ đỉnh bắt đầu).	DFS sẽ duyệt qua nhiều cạnh kề để tìm đến được đến đỉnh kết thúc (tính từ đỉnh bắt đầu).
Cấu trúc dữ liệu	Hàng đợi - FIFO	Ngăn xếp - LIFO
Ứng dụng	Tìm kiếm các đỉnh gần đỉnh nguồn đã cho.	Tìm kiếm đỉnh cách xa đỉnh nguồn đã cho.
Thứ tự duyệt	Duyệt những node chị em trước khi duyệt node con	Duyệt các node con trước khi duyệt các node chị em
Xóa node/đỉnh đã được duyệt/thăm	Những node/đỉnh đã được duyệt thì sẽ được xóa trong queue.	Những node/đỉnh đã được thăm thì sẽ được thêm vào stack, sau đó sẽ bị xóa đi khi không còn node/đỉnh để thăm nữa.
Kỹ thuật quay lui	Không có	Có
Ứng dụng	Được sử dụng trong các bài toán về đồ thị hai phía (Bipartite graph), tìm kiếm đường đi ngắn nhất, v..v	Được sử dụng trong các bài toán về đồ thị tuần hoàn và thứ tự cấu trúc liên kết,..
Độ phức tạp thời gian	$O(b^d)$	Tăng theo hàm mũ $O(b^m)$
Độ phức tạp không gian	$O(b^{d-1})$ cho những đỉnh đã được duyệt và $O(b^d)$ cho hàng đợi. => Tốn nhiều bộ nhớ hơn DFS	Gần như là tuyến tính O(bm). => Tốn ít bộ nhớ
Tốc độ	Chậm hơn so với DFS	Nhanh hơn BFS

Đặc điểm	BFS	DFS
Tính đầy đủ	Có	Nếu không gian tìm kiếm là vô hạn thì thuật toán không đảm bảo tìm được đường đi. Còn với không gian hữu hạn thì có.
Tính tối ưu	Tối ưu cho đồ thị không có trọng số, nhưng trường hợp tổng quát thì không tối ưu trong bài toán xác định sự tồn tại của đường đi	Không tối ưu trong việc tìm đường đi ngắn nhất.

2.4 Thuật toán tìm kiếm chi phí cực tiểu - Uniform Cost Search

Đây là một biến thể của BFS vì nó tìm đường đi ngắn nhất theo ý nghĩa số bước di chuyển ít nhất nhưng không tìm đường đi cho chi phí nhỏ nhất. Điều đó thể hiện rõ ở tính tối ưu còn hạn chế về loại đồ thị (yêu cầu là đồ thị không có trọng số).

Thuật toán sẽ sử dụng hàng đợi có độ ưu tiên.

Ý tưởng thuật toán: UCS là một thuật toán tìm kiếm tại nút bắt đầu và tiếp tục duyệt các nút tiếp theo với trọng số hay chi phí thấp nhất tính từ nút gốc.

Mã giả:

```
function UCS(graph, start, goal):

    open_set <- create an empty priorityQueue
    close_set <- create an empty array

    while !open_set.is_empty:
        #lay mot node ra khoi open, cost cua node nay bang 0
        selecte_node = remove from opened list, the node with the minimun
            distance value
         if selected_node == goal:
           calculate path
           return path
        #Them node da chon vao close
        add selected_node to closed list
        #Tao mot tap child_node chua cac node hang xom cua node da chon o
            tren
        child_node = get_neighbors(selected_node)
        if the selected node has children:
            for each child_node in children:
                calculate the distance value of child_node
            if child_node not in closed and opened lists:
                child.parent = selected_node
                add the child_node to opened list
            else if child_node in opened list:
                 if the distance value of child_node < the corresponding
                    node in opened list:
                   child.parent = selected_node
                   add the child_node to opened list
} #end function
```

Đánh giá thuật toán: Đầu vào UCS được giả định là một đồ thị hữu hạn **(có trọng số)**, được biểu diễn một cách rõ ràng dưới dạng danh sách kề, ma trận kề hoặc biểu diễn tương tự.

Tính đầy đủ: Có. nếu tồn tại hữu hạn phân nhánh trong đồ thị và trọng số bằng 0.

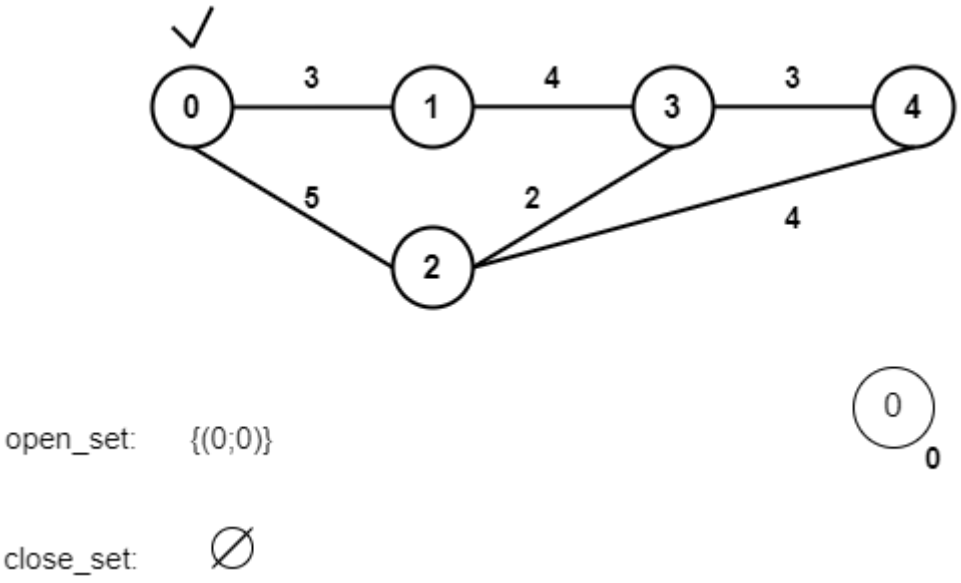
Tính tối ưu: luôn tối ưu vì nó chỉ chọn một đường dẫn có chi phí đường dẫn thấp nhất.

Độ phức tạp: Với ε là chi phí thấp nhất và $C*$ là chi phí tối ưu để đi từ node bắt đầu đến node đích. Ta có:

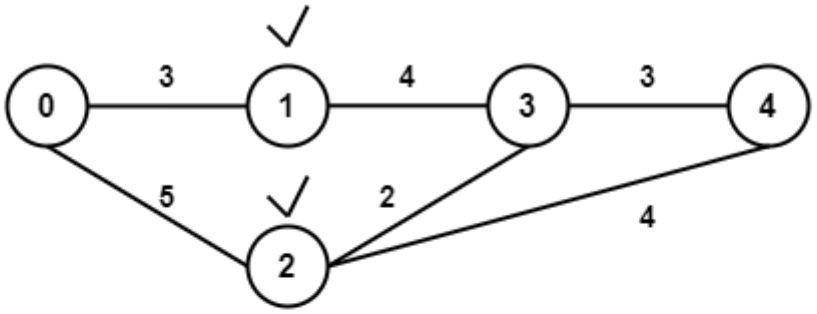
1. Về không gian: $O(b^{1+[\frac{C*}{\varepsilon}]})$
2. Về thời gian: $O(b^{1+[\frac{C*}{\varepsilon}]})$

Minh họa thuật toán:

Đưa start node (node 0) vào open set:

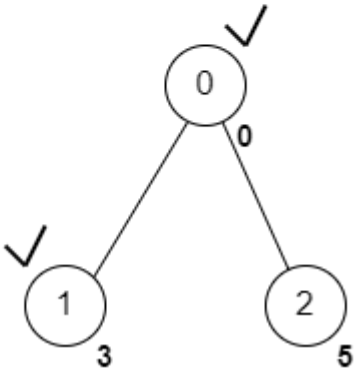


Node con của 0 là 1 và 2, với cost lần lượt là 3 và 5 được duyệt. Đưa node 1,2 vào open set.



open_set: {(1,3), (2,5)}

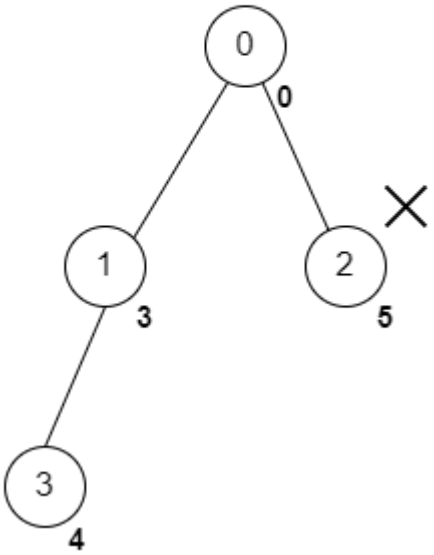
close_set: {(0,0)}



Vì cost[1] < cost[2] nên loại bỏ node 2 khỏi open set. Đưa node 1 vào close set.

open_set: {(3,7)}

close_set: {(0,0), (1,3)}

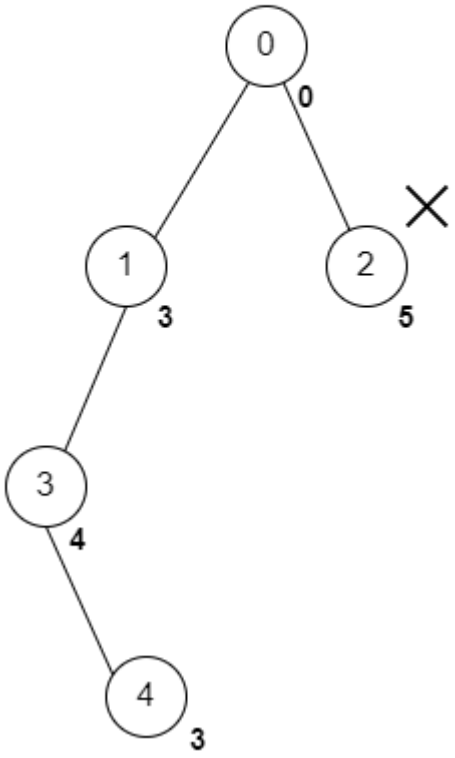


Node 1 có đường dẫn đến node 3 với cost[3] = 7. Ta lưu node 3 vào open set

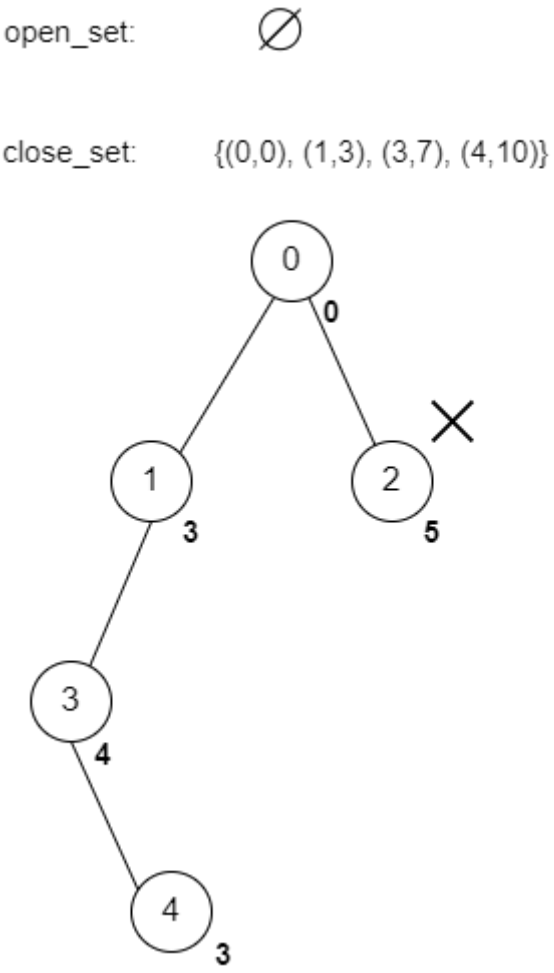
Tại node 3 có đường dẫn đến node 4 với cost[4] = 10. Loại bỏ node 3 khỏi open set và lưu vào close set.

open_set: {(4,10)}

close_set: {(0,0), (1,3), (3,7)}



Node 4 là đích đến nên đẩy node 4 ra khỏi open set vào close set, ta dừng thuật toán.



2.5 So sánh giữa UCS và Dijkstra

Đặc điểm	**UCS**	**Dijkstra**
Khởi tạo	Một tập hợp chỉ chứa duy nhất đỉnh nguồn của đồ thị	Một tập hợp chứa tất cả các đỉnh của đồ thị.
Bộ nhớ	Chỉ lưu trữ những đỉnh cần thiết sử dụng	Cần không gian lưu trữ nhiều hơn.
Cấu trúc dữ liệu	Hàng đợi ưu tiên	Ngăn xếp
Độ phức tạp thời gian	Nhanh hơn. $O(b^{1+[\frac{C*}{\varepsilon}]})$ Trong đó b là số node "hàng xóm" tối đa của một node gốc, C* là chi phí tối ưu và ε là chi phí thấp nhất để giải bài toán.	Chậm hơn. $O(V^2)$ hoặc $O(\|E\|+\|V\|*log\|V\|)$ Trong đó E là số cạnh và V là số đỉnh.
Độ phức tạp không gian	$O(b^{1+[\frac{C*}{\varepsilon}]})$	$O(\|V\|)$
Nhược điểm	Tối ưu cho đồ thị có trọng số, nhưng vì chỉ quan tâm đến chi phí đường đi nên thuật toán này có thể bị mắc kẹt trong một vòng lặp vô hạn.	Khi làm việc với các đồ thị có trọng số âm hoặc đồ thị phức tạp với $E \approx V^2$, thuật toán Dijkstra không tính toán chính xác các đường đi ngắn nhất.

2.6 Thuật toán tìm kiếm AStar - A* Search

Tìm kiếm A^* là một hình thức của tìm kiếm tối ưu thông qua việc áp dụng hàm heuristic.
Heuristic là phương pháp giải quyết vấn đề dựa trên trực giác đưa ra các ước lượng để tìm ra giải pháp gần như là tốt nhất và nhanh chóng.
Hàm Heuristic là hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.
A^* là một thuật giải đánh giá một nút dựa trên chi phí từ nút gốc đến nút đó - $g(n)$ cộng với một ước lượng chi phí từ nút đó đến đích - $h(n)$
Hàm số đánh giá:

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

Trong đó:
- $g(n)$ là chi phí từ nút gốc đến nút n hiện tại.
- $h(n)$ là ước lượng chi phí ngắn nhất để đi từ nút hiện tại n đến đích.
- $f(n)$ là ước lượng chi phí của lời giải "tốt nhất" qua n.

Trường hợp đặc biệt của A^*:

Trường hợp 1:

$$f(n) = g(n)$$

Đó là Uniform Cost Search - UCS đã được trình bày ở phần trên. $\Longrightarrow A^T$ Trường hợp 2:

$$f(n) = h(n)$$

Đó là tìm kiếm tham lam - Greedy Search với các thông tin đỉnh, cạnh, chi phí, thông tin cân bằng từ hàm $h(n)$. $\Longrightarrow A^{KT}$

Ý tưởng thuật toán:

Cân bằng lại trọng số cho đồ thị thông qua hàm Heuristic tuỳ biến và sau đó tìm kiếm bằng thuật toán Dijsktra hoặc Greedy và các thuật toán tìm kiếm khác trên đồ thị mới này.

Mã giả:

```
function Astar(g:graph, start, goal):
    input: graph, start_node, end_node
    output: the path from start_node to end_node
```

```
    open_set = {g.start}
    close_set = {}
    g(start) = 0
    f(start) = h(start)

    draw_path = []

    if open_set is empty then
        return failure

    selection_node = open_set.pop(node_lowest_cost)
    close_set.append(selection_node)

        if selection_node == g.goal then
            return solution
        else
            for each node in neighbours of selection_node
                if node not in close_set or node not in close_set then:
                    g(node) = g(selection_node) +
                        euclid_distance(selection_node,node)
                    f(node) = g(selection_node) + h(node)
                    open_set.add(node)
                end if
                if node in open_set or close_set then:
                    g(node) = min(g(node, g(selection_node) +
                        cost(selection_node, node) ))
                    f(node) = g(node) + h(node)
                end if
                if f(node) decrease and node in close_set then:
                    open_set.reverse(node)
                end if
            end for
        end if
end function
```

Đánh giá thuật toán:

Tính đầy đủ và tối ưu: Thuật giải A^* sẽ cho ra kết quả đầy đủ và tối ưu nếu chọn hàm heuristic $h(n)$ phù hợp nhất.

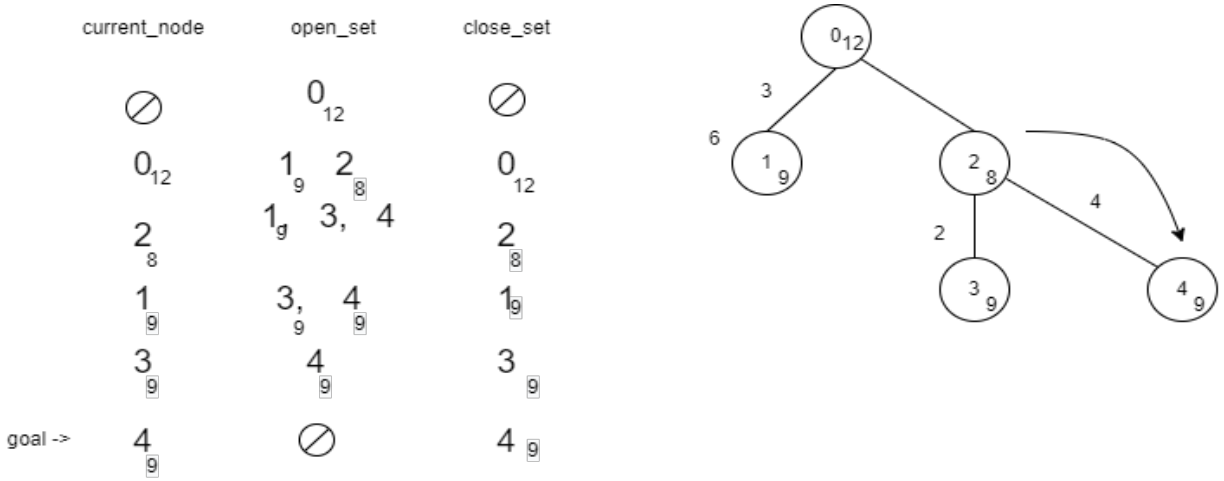
Với đồ thị hữu hạn, trọng số không âm, A^* đảm bảo được sự kết thúc và đầy đủ, tức

là nó sẽ cho ra được đường đi tới đích nếu tồn tại. Với đồ thị vô hạn, hệ số phân nhánh của mỗi nút hữu hạn và chi phí mỗi cạnh là một số dương, thuật giải vẫn có thể đưa ra được đường đi tới đích nếu tồn tại.

Độ phức tạp về thời gian: Độ phức tạp về thời gian của A* phụ thuộc vào cách chọn hàm Heuristic hay chính là h(n). Hàm Heuristic tốt và tối ưu thì thuật toán đạt hiệu năng cao.Trong trường hợp xấu nhất của không gian tìm kiếm không bị giới hạn, số lượng nút được mở rộng là cấp số nhân theo chiều sâu d của lời giải (đường đi ngắn nhất): $O(b^d)$, trong đó b là hệ số phân nhánh

Độ phức tạp về không gian: Độ phức tạp về không gian của A* gần giống như các thuật toán khác, vì tất cả mọi thứ như đường đi, chi phí cần được lưu trữ, và việc tính toán các hàm heuristic cũng phức tạp, nên sẽ rất tốn bộ nhớ.

Minh họa thuật toán:



Tài liệu

[1] Trần Hoài An, vnoi.info, *Các thuật toán về tìm đường đi ngắn nhất*, access date: 25/11/2022

URL: https://vnoi.info/wiki/algo/graph-theory/shortest-path.md

[2] GeeksforGeeks, 22 Nov, 2022 *Graph Data Structure And Algorithms*, access date: 28/11/2022

URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

[3] Milos Simic, Baeldung, *Bidirectional Search for Path Finding*, access date: 28/11/2022

URL: https://www.baeldung.com/cs/bidirectional-search

[4] Gang Wu, Baeldung, *Comparison Between Uniform-Cost Search and Dijkstra's Algorithm*, access date: 28/11/2022

URL: https://www.baeldung.com/cs/uniform-cost-search-vs-dijkstras

[5] Javatpoint, *Uninformed Search Algorithms*, access date: 30/11/2022

URL: https://www.javatpoint.com/ai-uninformed-search-algorithms

[6] Stuart J. Russell và Peter Norvig, 2010, Publisher Prentice Hall *Artificial Intelligence: A Modern Approach (Third Edition)*.

[7] Andreas Soularidis, February 21st, 2022, *Uniform Cost Search (UCS) Algorithm in Python*, access date: 08/12/2022

URL: https://plainenglish.io/blog/uniform-cost-search-ucs-algorithm-in-p

[8] Huỳnh Luật, *Lab01 - Lop co so tri tue nhan tao 20-22 HCMUS, Function drawpath* , access date: 04/12/2022

URL: https://www.youtube.com/watch?v=2_cWBZ3x5Sc&ab_channel=Lu%E1%BA%ADtHu%E1%BB%B3nh

[9] Shivali Bhadaniya, December 13th, 2020, *Breadth First Search in Python (with Code) | BFS Algorithm*, access date: 08/12/2022

URL: https://favtutor.com/blogs/breadth-first-search-python

[10] GeekforFeeks, June 22, 2022, *Python Program for Breadth First Search or BFS for a Graph*, access date: 08/12/2022

URL: https://www.geeksforgeeks.org/python-program-for-breadth-first-sear

[11] eductive, *How to implement a breadth-first search in Python*, access date: 09/12/2022

URL: https://www.educative.io/answers/how-to-implement-a-breadth-first-search-in-python

[12] Shivali Bhadaniya, December 21th, 2020, *Depth First Search in Python (with Code) | DFS Algorithm*, access date: 09/12/2022

URL: https://favtutor.com/blogs/depth-first-search-python

[13] Python Pool, *The Insider's Guide to A* Algorithm in Python*, access date: 10/12/2022

URL: https://www.pythonpool.com/a-star-algorithm-python/

[14] Red Blod Games,Jul 2014, then Feb 2016, Nov 2018, Oct 2020,*mplementation of A**, access date: 10/12/2022

URL: https://www.redblobgames.com/pathfinding/a-star/implementation.html

PHÂN CÔNG CÔNG VIỆC

Ngày bắt đầu	Công việc	Nội dung công việc	Hạn chót	Mức độ hoàn thành
23/11/2022	Lên kế hoạch làm việc	- Liệt kê những công việc cần thực hiện - Phân chia nhiệm vụ cho từng thành viên - Tạo Google Drive để tập hợp các tư liệu, source code. - Cài đặt và học Python cấp tốc	27/11/2022	100%
28/11/2022	Tìm hiểu & trình bày thuật toán	- BFS, DFS, UCS, Dijkstra & Astar - Yêu cầu: Ý tưởng chung, đánh giá và so sánh	29/11/2022	100%
30/11/2022	Cài đặt thuật toán	- Viết mã giả - Vẽ minh hoạ thuật toán - Cài đặt thuật toán trên Python + Minh Anh: BFS, DFS + Ngọc Lam: UCS, Astar	06/12/2022	90%
05/12/2022	Fix bug	- Kiểm tra tính đúng đắn của các mã giả, thuật toán, đối chiếu lẫn nhau: Minh Anh. - Tiến hành fix bug và hoàn thiện cài đặt: Ngọc Lam.	08/12/2022	100%
10/12/2022	Video demo	- Quay và edit video demo: Ngọc Lam	11/12/2022	100%
07/12/2022	Viết báo cáo	- Tổng hợp nội dung và trình bày Latex: Minh Anh - Sửa chữa, bổ sung báo cáo: Ngọc Lam	11/12/2022	100%
11/12/2022	Tổng duyệt	Kiểm duyệt văn bản báo cáo và tổng hợp bài nộp: Cả nhóm	11/12/2022	100%

ĐÁNH GIÁ VÀ NHẬN XÉT

Trong 2,5 tuần triển khai đồ án, mặc dù ban đầu nhóm chúng em có gặp đôi chút khó khăn về việc cài đặt Python (cụ thể là Pygame) và làm quen với ngôn ngữ mới - Python một cách gấp rút. Nhưng với sự nghiêm túc, chỉnh chu và tâm huyết cho bài báo cáo đồ án, nhóm chúng em đã hoàn thành đồ án tốt nhất trong khả năng của mình theo đúng tiến độ đề ra. Chính vì thế, chúng em xin tự đánh giá số điểm cho bài báo cáo của mình là 10 điểm để thật xứng đáng với sự đầu tư của mình.

Lời cuối cùng xin được gửi lời cám ơn đến thầy Bùi Tiến Lên và thầy Nguyễn Bảo Long đã tạo điều kiện để chúng em có cơ hội được học và tìm hiểu sâu hơn về các thuật toán thiên hướng Khoa học Dữ liệu, vốn là nền tảng quan trọng cho ngành học của chúng em sau này.

Trân trọng.