ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Cơ Sở Trí Tuệ Nhân Tạo

La01: AI Search Algorihm

Sinh viên thực hiện:

Nguyễn Thị Thu Hằng - 18120027

I. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

II. THUẬT TOÁN TÌM KIẾM

1. BFS - Breadth First Search

a) Ý tưởng:

- Xuất phát từ 1 đỉnh bất kỳ, đi tới tất cả các đỉnh kề của nó, lưu các đỉnh này lại. Tiếp tục đem 1 đỉnh khác (từ tập đỉnh đã được lưu) ra xét và đi cho đến khi không còn đỉnh nào có thể đi.
- Trong quá trình đi từ đỉnh này sang đỉnh kia, t iến hành lưu lại đỉnh cha của đỉnh kề, để khi đi ngược lại từ đỉnh kết thúc đến đỉnh xuất phát, ta có được đường đi ngắn nhất.

b) Thuật giải:

Quy ước:

Open: là tập hợp chứa các đỉnh đang chờ được xét. Trong trường hợp sử dụng thuật toán BFS thì Open được tổ chức như một Queue

Close: là tập hợp các đỉnh đã được duyệt

Pre: là tập hợp các đỉnh liền trước của các đỉnh được lưu trữ qua quá trình duyệt

start: là đỉnh bắt đầu

goal: là đỉnh kết thúc

current_Node: là đỉnh đang xét

Mô tả :

Bước 1: Chèn đỉnh start vào hàng đợi Open. Và khởi tạo Pre[start]=start

Bước 2: Lấy ra đỉnh đầu tiên trong hàng đợi (current_Node) và thăm nó

- Nếu đỉnh này chính là goal, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được thăm trước đó vào hàng đợi.Pre của các đỉnh kề sẽ là current_Node

Bước 3: Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được thăm – dừng việc tìm kiếm và trả về "không thấy".

Bước 4: Nếu hàng đợi không rỗng thì quay về bước 2.

c) Mã nguồn:

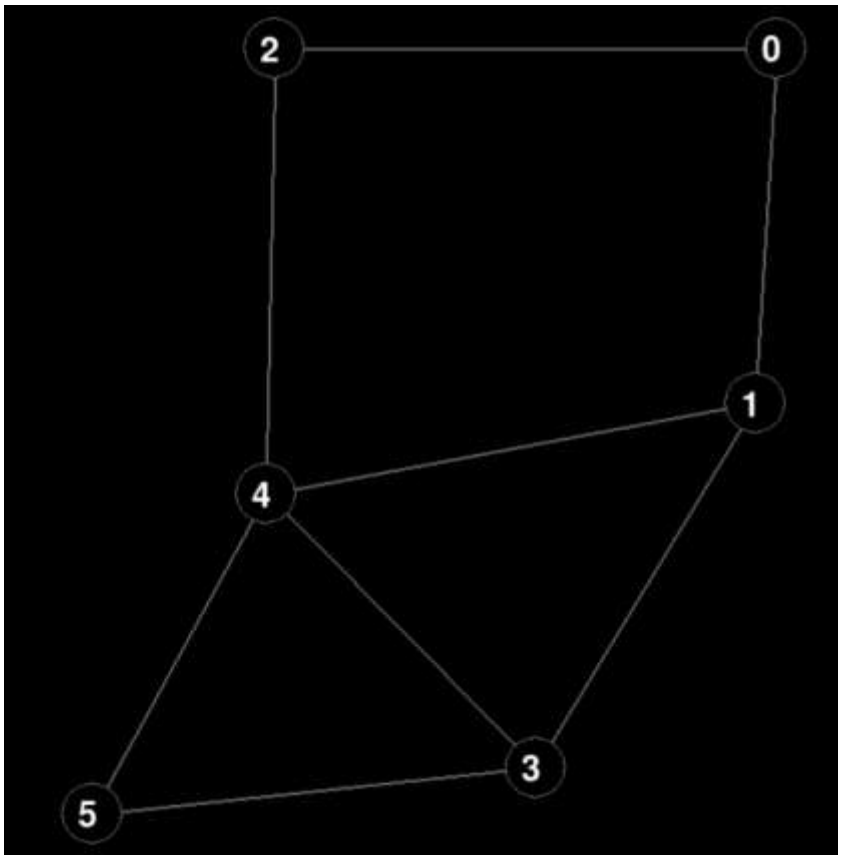
```
def BFS(graph, edges, edge_id, start, goal):
    """
    BFS search
    """
    # TODO: your code
    Close = []  # is a set of considered vertices, traversed
    Pre = [0] * 10000  # Pre[u]=v means v is the previous node of u
    Open = queue.Queue()  # is the set of vertices to be considered
in the next step in the queue
```

```
    Open.put(start)
    Pre[start] = start
    Close.append(start)
    graph[start][3] = red
    graphUI.updateUI()
    while (Open.qsize() != 0):
        current_Node = Open.get()
        graph[current_Node][3] = yellow
        graphUI.updateUI()
        adjecent_node = graph[current_Node][1]
        for i in adjecent_node:
            if (i == goal):
                edges[edge_id(current_Node, i)][1] = white
                graph[goal][3] = purple
                graphUI.updateUI()
                Pre[i] = current_Node
                while True:
                    edges[edge_id(i, Pre[i])][1] = green
                    i = Pre[i]
                    graphUI.updateUI()
                    if (i == start):
                        graph[start][3] = orange
                        graphUI.updateUI()
                        break
                return
            if (i not in Close):
                Pre[i] = current_Node
                Open.put(i)
                Close.append(i)
                graph[i][3] = red
                edges[edge_id(current_Node, i)][1] = white
                graphUI.updateUI()
        graph[current_Node][3] = blue
        graphUI.updateUI()
    print("Don't Have path from ", start, " to ", goal)
    pass
```

d) Test Case:

- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5

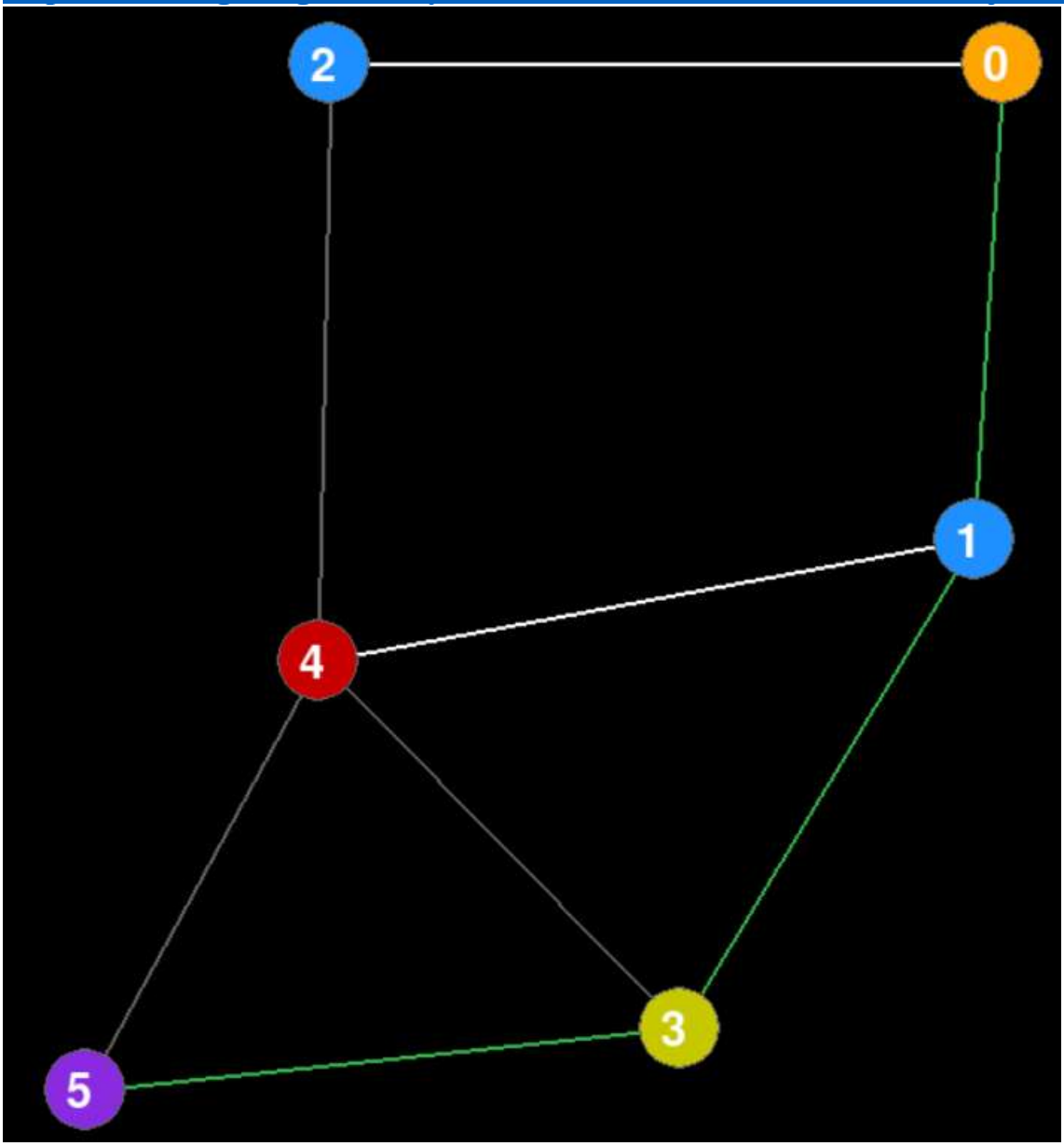


Mô tả quá trình xử lí:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase1.txt', 'bfs']
Open[] =  [0]
Close[] =  [0]
current_Node  0
Open[] =  [1, 2]
Close[] =  [0, 1, 2]
current_Node  1
Open[] =  [2, 3, 4]
Close[] =  [0, 1, 2, 3, 4]
current_Node  2
Open[] =  [3, 4]
Close[] =  [0, 1, 2, 3, 4]
current_Node  3
```

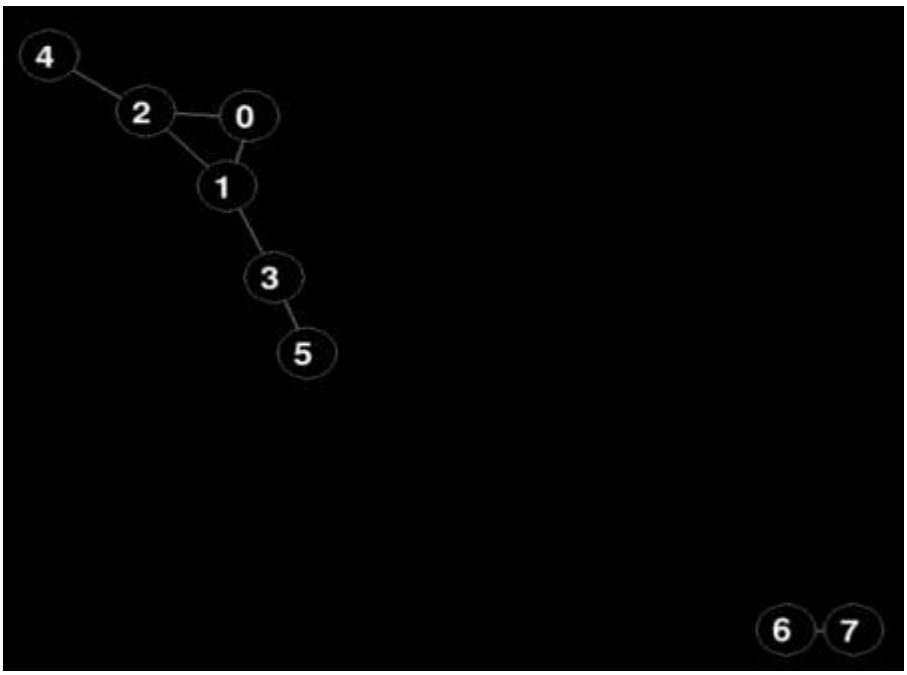
Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzIzOF8xNTA1MjczOQ



- **TestCase 2**

testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7

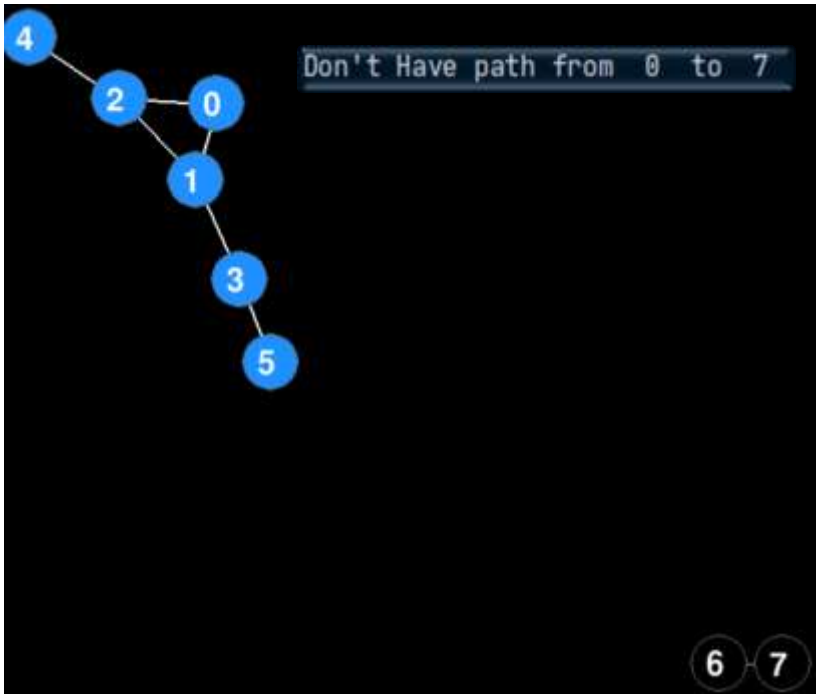


Mô tả quá trình xử lí:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase2.txt', 'bfs']
Open[] =  [0]
Close[] =  [0]
current_Node  0
Open[] =  [1, 2]
Close[] =  [0, 1, 2]
current_Node  1
Open[] =  [2, 3]
Close[] =  [0, 1, 2, 3]
current_Node  2
Open[] =  [3, 4]
Close[] =  [0, 1, 2, 3, 4]
current_Node  3
Open[] =  [4, 5]
Close[] =  [0, 1, 2, 3, 4, 5]
current_Node  4
Open[] =  [5]
Close[] =  [0, 1, 2, 3, 4, 5]
current_Node  5
Don't Have path from  0  to  7
```

Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzI0MV8xNTA1Mjc0Mg



e) Độ phức tạp

O (V + E) trong đó V là số đỉnh trong đồ thị và E là số cạnh trong đồ thị.

f) Ưu – Khuyết điểm

Ưu điểm:

- Tìm kiếm theo chiều rộng sử dụng kỹ thuật vét cạn không gian trạng thái bài toán vì vậy chắc chắn sẽ tìm được lời giải cho bài toán (nếu có)
- Tìm được đường đi qua ít đỉnh nhất

Khuyết điểm

- Tìm kiếm lời giải theo thuật toán đã định trước, do vậy tìm kiếm một cách máy móc; khi không có thông tin hỗ trợ cho quá trình tìm kiếm, không nhận ra ngay lời giải.
- Không phù hợp với không gian bài oán kích thước lớn. Đối với loại bài toán này, phương pháp tìm rộng đối mặt với các nhu cầu:
  + Cần nhiều bộ nhớ theo số nút cần lưu trữ.
  + Cần nhiều công sức xử lý các nút, nhất là khi các nhánh cây dài, số nút tăng.
  + Dễ thực hiện các thao tác không thích hợp, thừa, đưa đến việc tăng đáng kể số nút phải xử lý.
- Không hiệu qủa nếu lời giải ở sâu. Phương pháp này không phù hợp cho trường hợp có nhiều đường dẫn đến kết quả nhưng đều sâu.

- Giao tiếp với người dùng không thân thiện. Do duyệt qua tất cả các nút, việc tìm kiếm không tập trung vào một chủ đề.

2. DFS - Depth First Search

a) Ý tưởng:

Từ một đỉnh S ban đầu ta sẽ có các đỉnh kề là x, từ đỉnh x ta sẽ có các đỉnh kề là y, và nó cũng thuộc nhánh s-x-y… Chúng ta thăm các nhánh đó theo chiều sâu (thăm đến khi không còn đỉnh kề chưa duyệt). Điều đó gợi cho chúng ta tạo một stack để lưu trữ đỉnh và các đỉnh kề của đỉnh. Tính chất First In Last Out của stack giúp ta có thể mô tả việc duyệt từ đỉnh u sang đỉnh kề v chưa được thăm theo chiều sâu

b) Thuật giải:

Quy ước:

Open: là tập hợp chứa các đỉnh đang chờ được xét. Trong trường hợp sử dụng thuật toán DFS thì Open được tổ chức như một Stack

Close: là tập hợp các đỉnh đã được duyệt

Pre: là tập hợp các đỉnh liền trước của các đỉnh được lưu trữ qua quá trình duyệt

start: là đỉnh bắt đầu

goal: là đỉnh kết thúc

current_Node: là đỉnh đang xét

Mô tả :

Bước 1: Tập Open chứa đỉnh gốc start chờ được xét.

Bước 2: Kiểm tra tập Open có rỗng không

- Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét current_Node. Nếu current_Node là đỉnh goal cần tìm, kết thúc tìm kiếm.
- Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được thăm – dừng việc tìm kiếm và trả về "không thấy".

Bước 3: Đưa đỉnh current_Node vào tập Close, sau đó xác định các đỉnh kề với đỉnh current_Node vừa xét.Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open. Gán Pre[] của các đỉnh kề của current_Node là current_Node

Bước 4: Nếu hàng đợi không rỗng thì quay về bước 2.

c) Mã nguồn

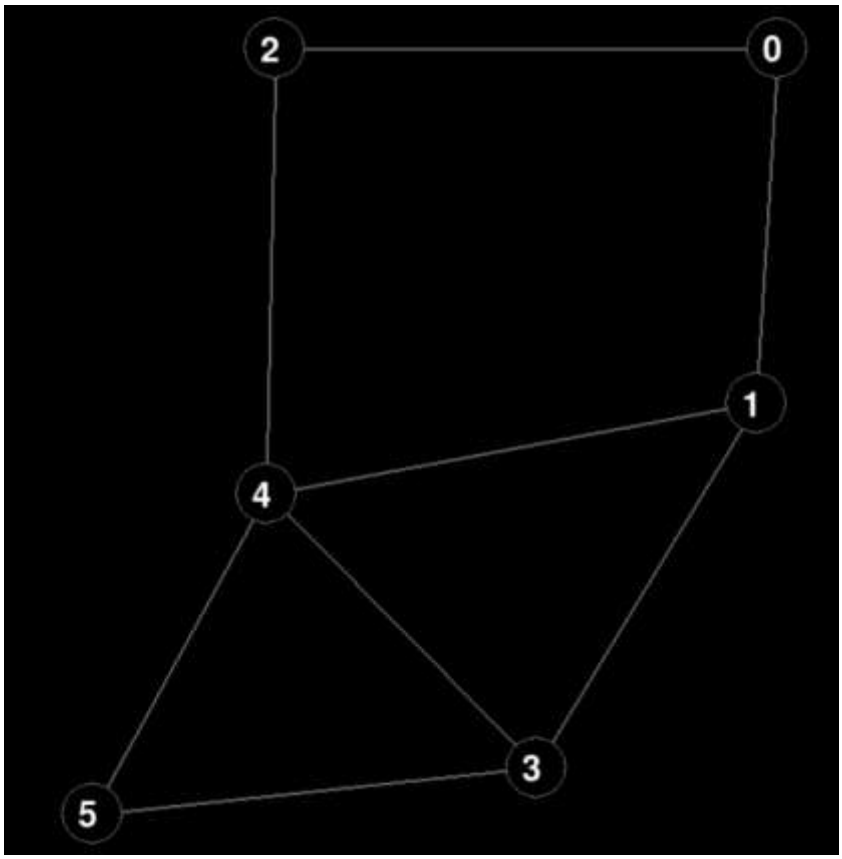
```
def DFS(graph, edges, edge_id, start, goal):
    """
    DFS search
    """
    # TODO: your code
    Close = [] # is a set of considered vertices, traversed
```

```
    Pre = [0] * 10000  # Pre[u]=v means v is the previous node of u
    Open = [] # is the set of vertices to be considered in the next step
in the stack
    # Add start into Open
    Open.append(start)
    Pre[start]=start
    #Update Graph
    graph[start][3]=red
    graphUI.updateUI()

    while (len(Open)!=0):
        current_Node = Open.pop()
        if (current_Node not in Close):
            Close.append(current_Node)
        #Update Graph
        graph[current_Node][3] = yellow
        graphUI.updateUI()
        # Take adjecent_node list
        adjecent_Node = graph[current_Node][1]
        for i in adjecent_Node:
            if (i == goal):
                edges[edge_id(current_Node, i)][1] = white
                graph[goal][3] = purple
                graphUI.updateUI()
                Pre[i] = current_Node
                while True:
                    edges[edge_id(i, Pre[i])][1] = green
                    i = Pre[i]
                    graphUI.updateUI()
                    if (i == start):
                        graph[start][3] = orange
                        graphUI.updateUI()
                        break
                return
            if (i not in Close):
                # add i into Open
                Open.append(i)
                Pre[i] = current_Node
                graph[i][3] = red
                edges[edge_id(i, current_Node)][1] = white
                graphUI.updateUI()
        graph[current_Node][3]=blue
        graphUI.updateUI()
    print("Don't Have path from ", start, " to ", goal)
    pass
```

d)Test Case

- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5



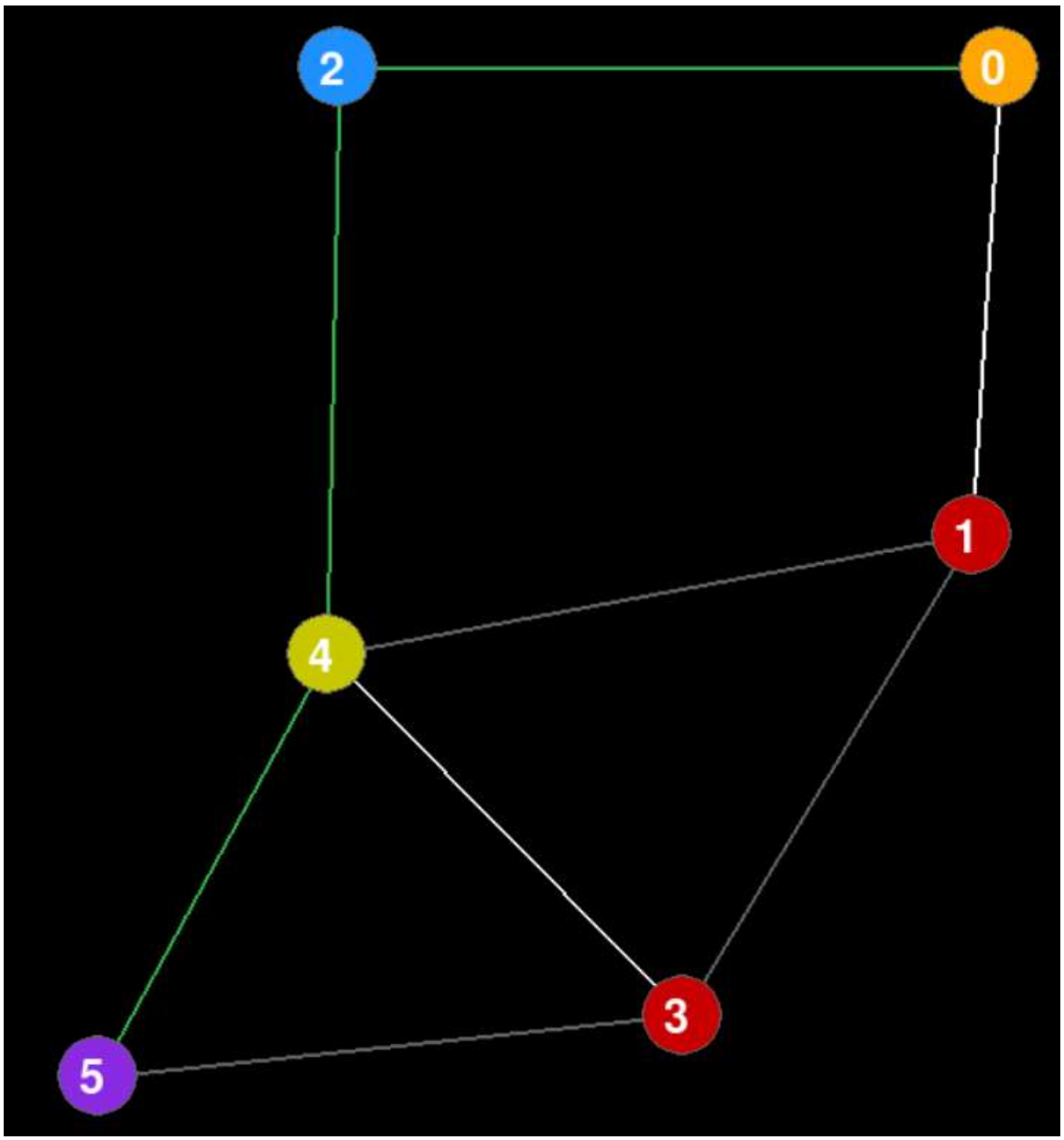
Mô tả quá trình xử lí:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase1.txt', 'dfs']
Open[] =  [0]
Close[] =  []
current_Node  0
Open[] =  [1, 2]
Close[] =  [0]
current_Node  2
Open[] =  [1, 4]
Close[] =  [0, 2]
current_Node  4
```

Kết quả:

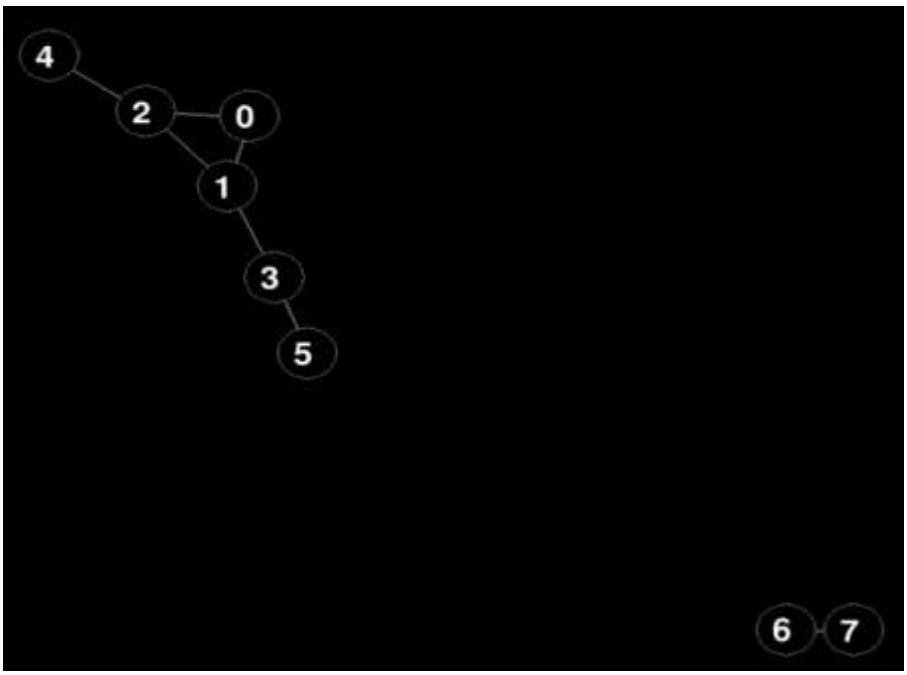
https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzE2M18xNTA1MjM5MQ

AI Search Algorithm



- **TestCase 2**

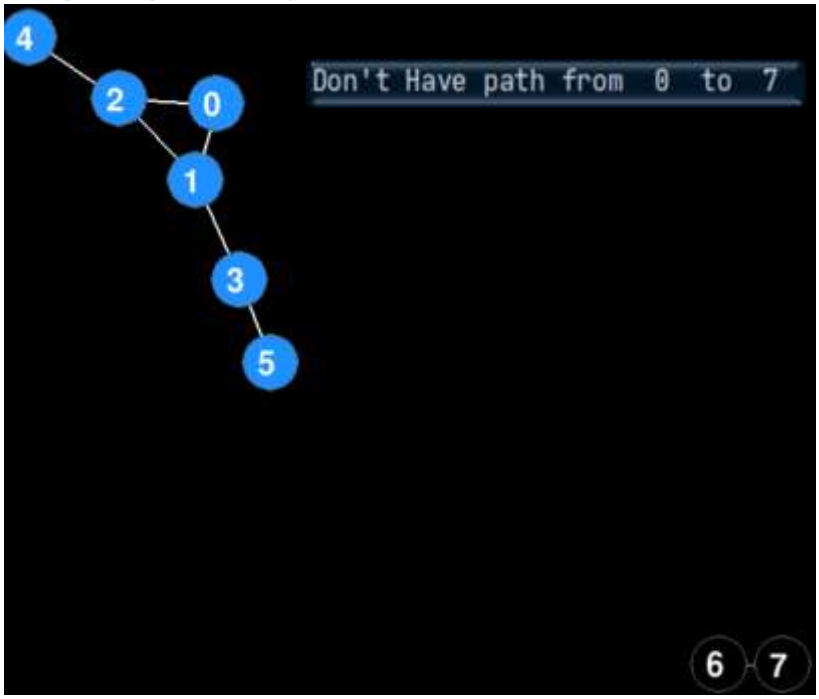
testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7



Mô tả quá trình xử lí:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase2.txt', 'dfs']
Open[] =  [0]
Close[] =  []
current_Node  0
Open[] =  [1, 2]
Close[] =  [0]
current_Node  2
Open[] =  [1, 1, 4]
Close[] =  [0, 2]
current_Node  4
Open[] =  [1, 1]
Close[] =  [0, 2, 4]
current_Node  1
Open[] =  [1, 3]
Close[] =  [0, 2, 4, 1]
current_Node  3
Open[] =  [1, 5]
Close[] =  [0, 2, 4, 1, 3]
current_Node  5
Open[] =  [1]
Close[] =  [0, 2, 4, 1, 3, 5]
current_Node  1
Don't Have path from  0  to  7
```

Kết quả:
https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzE2MF8xNTA1MjM4Ng



e) Độ phức tạp

Độ phức tạp của DFS: O(V + E) (với V là số đỉnh)

f) Ưu – Khuyết điểm

Ưu điểm:

- Xét duyệt tất cả các đỉnh để trả về kết quả.
- Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả.

Khuyết điểm:

- Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn.
- Mang tính chất mù quáng, duyệt tất cả đỉnh, không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết.

3. UCS – Uniform Cost Search

a) Ý tưởng:

- Xuất phát từ một đỉnh ban đầu (start) đi đến các đỉnh kề của nó, đồng thời dùng một hàng đợi ưu tiên để lưu lại (Priority Queue) với tiêu chí ưu tiên là chi phí đi từ đỉnh ban đầu đến đỉnh đang xét.

- Tiếp tục như vậy cho đến khi đi đến đích (goal) thì ta có được đường đi với chi phí nhỏ nhất hoặc Hàng đợi ưu tiên rỗng thì kết luận không có đường đi đến được đích

b) Thuật giải:

Quy ước:

Open: là tập hợp chứa các (node, cost) đang chờ được xét. Trong trường hợp sử dụng thuật toán UCS thì Open được tổ chức như một Priority Queue (Trong đó node là đỉnh chờ xét và cost là chi phí đi từ đỉnh start)

Close: là tập hợp các đỉnh đã được duyệt

start: là đỉnh bắt đầu

goal: là đỉnh kết thúc

current_Node: là đỉnh đang xét

cost_i: là chi phí đi từ current_Node đến i (i là các node kề của current_Node). Trong trường hợp này, chi phí là khoảng cách trong hệ tọa độ Oxy từ start đến đỉnh i

Mô tả :

Bước 1: Tập Open chứa đỉnh gốc (start,0) chờ được xét.

Bước 2: Kiểm tra tập Open có rỗng không

- Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét current_Node. Nếu current_Node là đỉnh goal cần tìm, kết thúc tìm kiếm.
- Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được thăm trước đó vào hàng đợi.

Bước 3: Đưa đỉnh current_Node vào tập Close, sau đó xác định các đỉnh kề với đỉnh current_Node vừa xét.Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open.

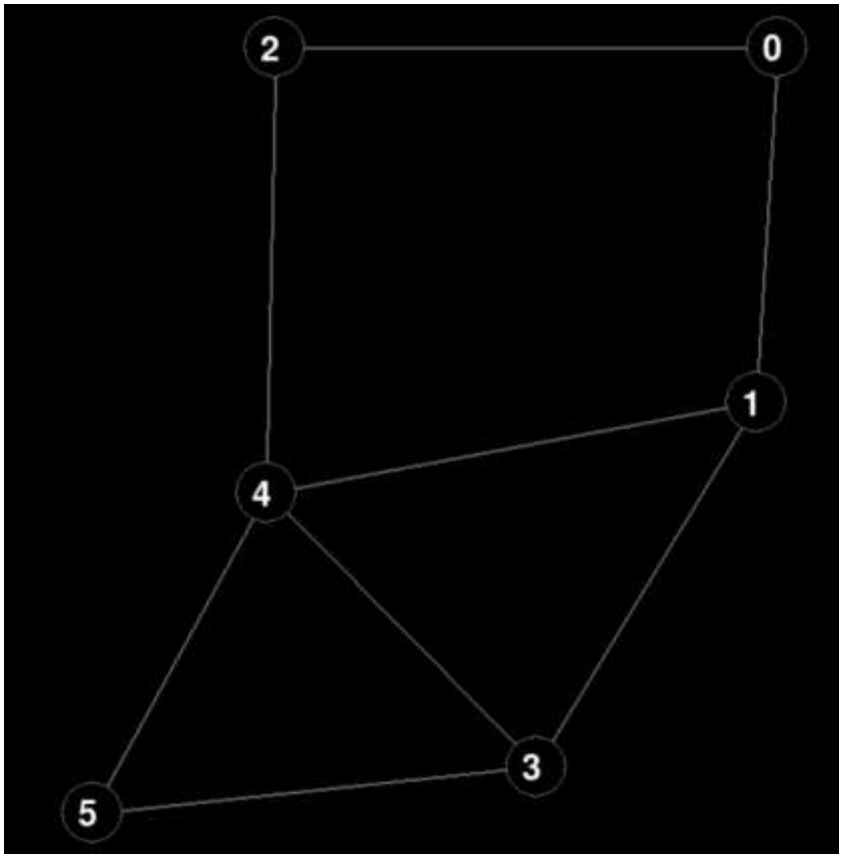
Bước 4: Nếu hàng đợi không rỗng thì quay về bước 2.

b) Mã nguồn

```
def UCS(graph, edges, edge_id, start, goal):
    """
    Uniform Cost Search search
    """
    # TODO: your code

    Close = []
    Pre = [0] * 1000
    Open = queue.PriorityQueue()
    Open.put((0, (start, start)))
    graph[start][3] = red
    graphUI.updateUI()
```

```
    while (Open.qsize() != 0):
        x = Open.get()
        cost = x[0]
        u = x[1][0]
        v = x[1][1]
        if (u not in Close):
            Close.append(u)
            Pre[u]=v
            if (u == goal):
                graph[goal][3] = purple
                graphUI.updateUI()
                Pre[u] = v
                while True:
                    edges[edge_id(u, Pre[u])][1] = green
                    graphUI.updateUI()
                    u = Pre[u]
                    if (u == start):
                        graph[start][3] = orange
                        graphUI.updateUI()
                        break
                return
            adjecent_node = graph[u][1]
            for i in adjecent_node:
                if (i not in Close):
                    Pre[i] = u
                    graph[i][3] = red
                    edges[edge_id(i, u)][1] = white
                    graphUI.updateUI()
                    x1 = graph[u][0]
                    x2 = graph[i][0]
                    cost_i = cost + int(math.sqrt(pow((x1[0] -
x2[0]), 2) + pow((x1[1] - x2[1]), 2)))
                    Open.put((cost_i, (i, u)))
        graph[u][3] = blue
        graphUI.updateUI()

    print("Don't Have path from ", start, " to ", goal)
    pass
```

c) Test Case



- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5

Mô tả:

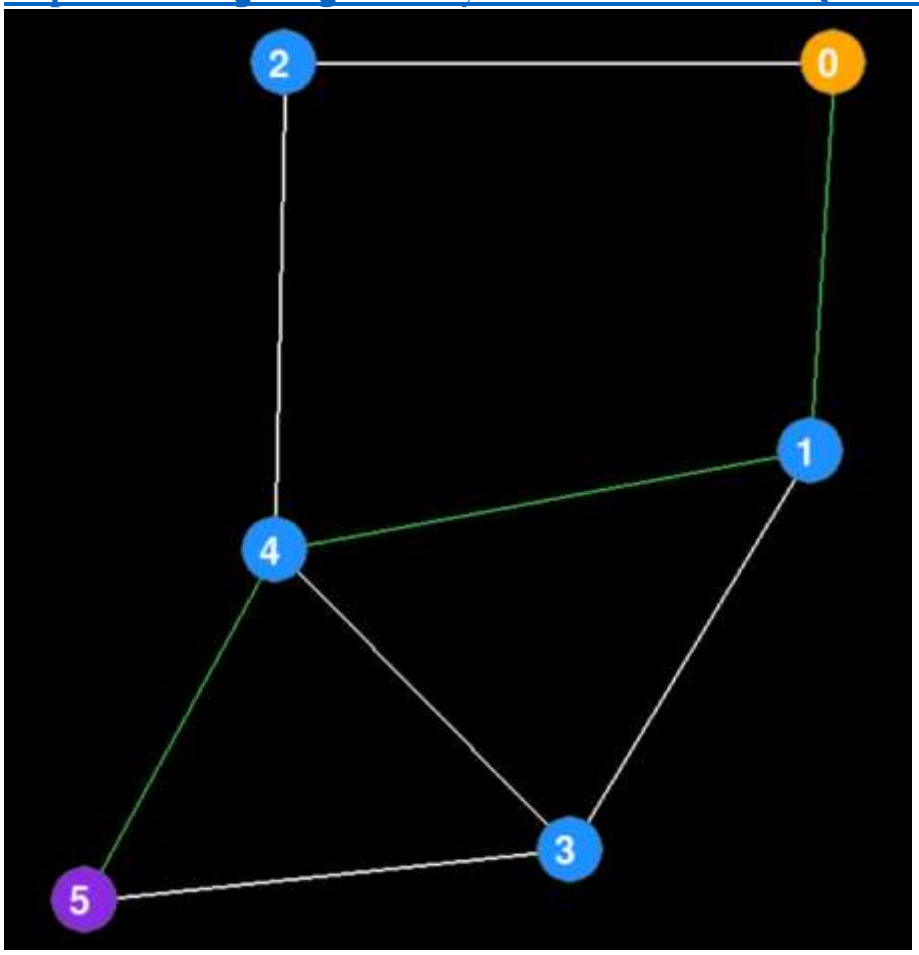
0	1	2	3	4	5
(0,0)	(∞; –)	(∞; –)	(∞; –)	(∞; –)	(∞; –)
-	(255;0)	(361;0)	(∞; –)	(∞; –)	(∞; –)
-	-	(361;0)	(560;1)	(612;1)	(∞; –)
-	-	-	(560;1)	(612;1)	(∞; –)
-	-	-	-	(612;1)	(880;3)
-	-	-	-	-	(873;4)

Open[]= [
(0, (0, 0))

]
Close[]= []
current_Node (0, (0, 0))
Open[]= [
(255, (1, 0))
(361, (2, 0))
]
Close[]= [0]
current_Node (255, (1, 0))
Open[]= [
(361, (2, 0))
(560, (3, 1))
(612, (4, 1))
]
Close[]= [0, 1]
current_Node (361, (2, 0))
Open[]= [
(560, (3, 1))
(612, (4, 1))
(681, (4, 2))
]
Close[]= [0, 1, 2]
current_Node (560, (3, 1))
Open[]= [
(612, (4, 1))
(681, (4, 2))
(836, (4, 3))
(880, (5, 3))
]
Close[]= [0, 1, 2, 3]
current_Node (612, (4, 1))
Open[]= [
(681, (4, 2))
(836, (4, 3))
(873, (5, 4))
(880, (5, 3))
]

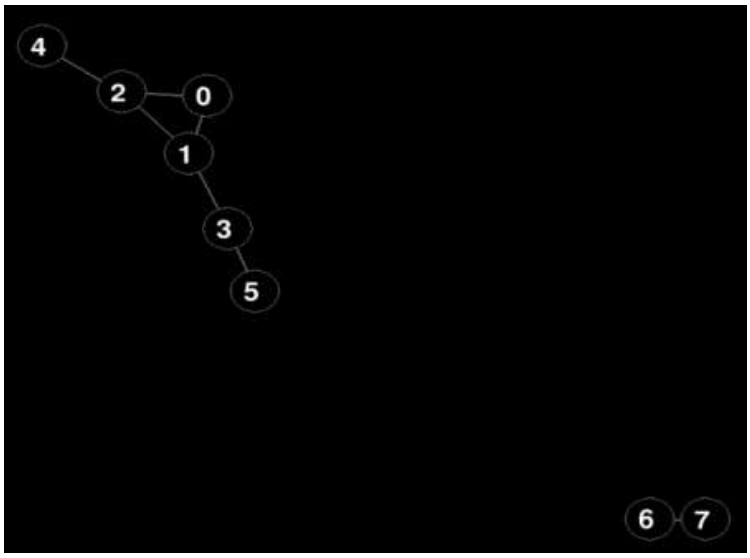
Close[]= [0, 1, 2, 3, 4]
current_Node (681, (4, 2))
Open[]= [
(836, (4, 3))
(873, (5, 4))
(880, (5, 3))
]
Close[]= [0, 1, 2, 3, 4]
current_Node (836, (4, 3))
Open[]= [
(873, (5, 4))
(880, (5, 3))
]
Close[]= [0, 1, 2, 3, 4]
current_Node (873, (5, 4))
Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzQ0OV8xNTA1Mzg5Ng



- **TestCase 2**

testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7



Mô tả:

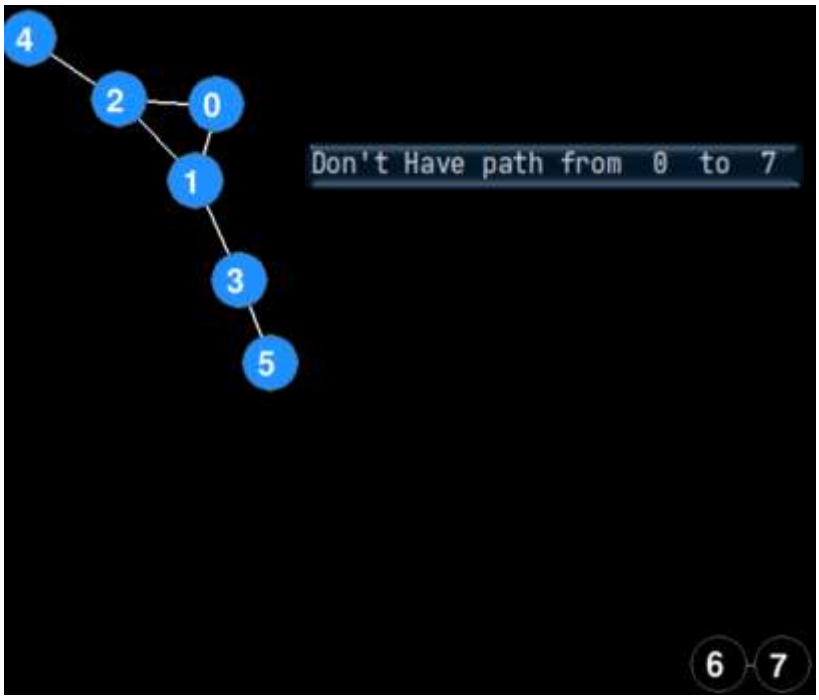
0	1	2	3	4	5	6	7
(0,0)	(∞; –)	(∞; –)	(∞; –)	(∞; –)	(∞; –)	(∞; –)	(∞; –)
-	(61;0)	(75;0)	(∞; –)	(∞; –)	(∞; –)	(∞; –)	(∞; –)
-	-	(75;0)	(145,1)	(∞; –)	(∞; –)	(∞; –)	(∞; –)
-	-	-	(145,1)	(159;2)	(∞; –)	(∞; –)	(∞; –)
-	-	-	-	(159;2)	(215,3)	(∞; –)	(∞; –)
-	-	-	-	-	(215;3)	(∞; –)	(∞; –)
-	-	-	-	-	-	(∞; –)	(∞; –)

Open[]= [
(0, (0, 0))
]
Close[]= []
current_Node (0, (0, 0))
Open[]= [
(61, (1, 0))
(75, (2, 0))
]
Close[]= [0]
current_Node (61, (1, 0))
Open[]= [
(75, (2, 0))
(145, (3, 1))
(147, (2, 1))
]

Close[]= [0, 1]
current_Node (75, (2, 0))
Open[]= [
(145, (3, 1))
(147, (2, 1))
(159, (4, 2))
]
Close[]= [0, 1, 2]
current_Node (145, (3, 1))
Open[]= [
(147, (2, 1))
(159, (4, 2))
(213, (5, 3))
]
Close[]= [0, 1, 2, 3]
current_Node (147, (2, 1))
Open[]= [
(159, (4, 2))
(213, (5, 3))
]
Close[]= [0, 1, 2, 3]
current_Node (159, (4, 2))
Open[]= [
(213, (5, 3))
]
Close[]= [0, 1, 2, 3, 4]
current_Node (213, (5, 3))
⇨ Don't Have path from 0 to 7

Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc2MzQ4N18xNTA1NDE0Ng



d) Độ phức tạp

Độ phức tạp: O (m ^ (1 + tầng (l / e)))
trong đó,
m là số lân cận lớn nhất mà một nút có
l là độ dài của đường đi ngắn nhất đến trạng thái mục tiêu
e là chi phí cạnh nhỏ nhất

4. A*

A* là một thuật toán tìm kiếm trong đồ thị, tìm đường đi từ một nút khởi đầu đến một nút đích cho trước sử dụng một hàm heuristic ước lượng khoảng cách từ nút hiện tại đến nút đích (trạng thái đích), và nó sẽ duyệt đồ thị theo thứ tự ước lượng Heuristic này.

a) Ý tưởng:

Xét bài toán tìm đường, A* sẽ xây dựng tăng dần các tuyến đường từ đỉnh xuất phát đến khi nó tìm thấy đường đi chạm đến đích. Để xác định khả năng dẫn đến đích, A* sử dụng một đánh giá heuristic về khoảng cách từ một điểm bất kì cho trước đến đích. Trong ví dụ về bài toán tìm đường đi giao thông này thì đánh giá heuristic là khoảng cách đường chim bay từ điểm cho trước đến điểm đích.
A* đảm bảo tính đầy đủ và tối ưu, nó luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế. Đầy đủ và tối ưu hơn các thuật toán tìm đường đi khác

ở chỗ nó không chỉ ước lượng khoảng cách còn lại (nhờ đánh giá heuristic) mà còn tính khoảng cách đã đi qua để tìm được đường đi ngắn nhất.

b) Thuật giải:

A* lưu giữ một tập các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (priority queue). Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm f(x) = g(x) + h(x).

Trong đó, g(x) là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. h(x) là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ x. Trong trường hợp này "chi phí" được tính là khoảng cách đã đi qua, khoảng cách đường chim bay giữa hai điểm trên một bản đồ là một đánh giá heuristic cho khoảng cách còn phải đi tiếp. f(x) có giá trị càng thấp thì độ ưu tiên của x càng cao, ta sử dụng f(x) này để xác định nút tiếp theo.

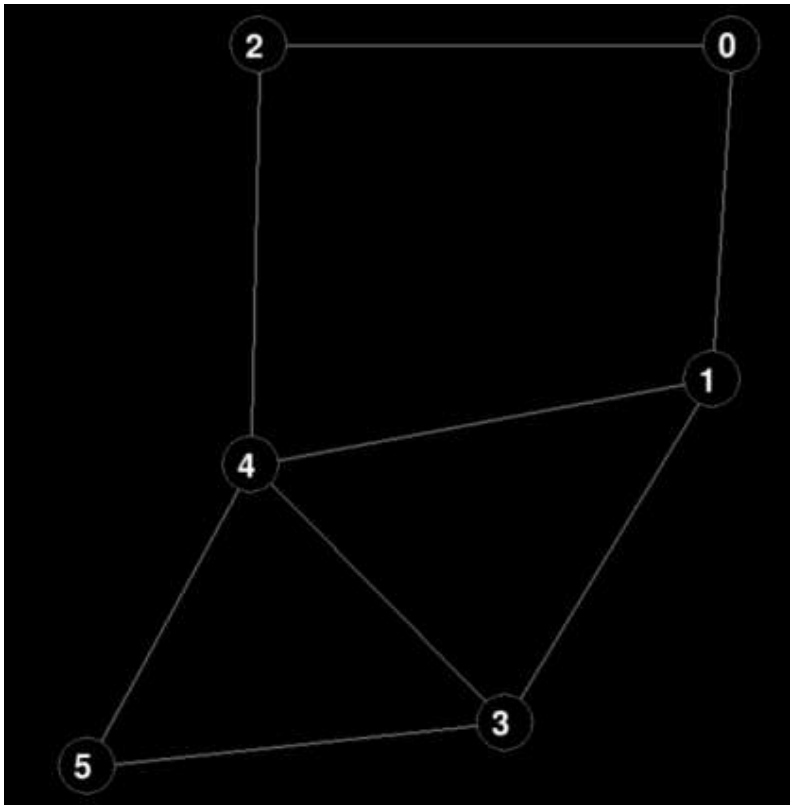
c) Mã nguồn

```
def AStar(graph, edges, edge_id, start, goal):
    """
            A star search
            """
    # TODO: your code
    # hx means the estimated movement cost to move from this vertice
to the goal
    # gx means the movemen cost from start to this vertice

    # Create lists for open nodes and closed nodes
    open = []
    close = []
    # Compute  the estimated movement cost
    N = len(graph)  # N is the number of Vertices
    hx = []
    fx = [10e7] * N
    gx = [0] * N

    Pre = [-1] * N
    for i in range(N):
        hx.append(dist(i, goal, graph))

    # Add the start node
    open.append((start, 0, start))
    graph[start][3] = red
    graphUI.updateUI()
    while (len(open) != 0):
        # Sort the open list to get the node with the lowest cost
first
        open.sort(key=Key_1)
        # Get the node with the lowest cost
        current_node = open.pop(0)
```

```
        graph[current_node[0]][3] = yellow
        graphUI.updateUI()
        # Add the current node to the closed list
        close.append(current_node[0])
        Pre[current_node[0]] = current_node[2]
        # Check if we have reached the goal, return the path
        if (current_node[0] == goal):
            graph[goal][3] = purple
            graphUI.updateUI()
            u = current_node[0]
            v = current_node[2]
            Pre[u] = v
            while True:
                edges[edge_id(u, Pre[u])][1] = green
                graphUI.updateUI()
                u = Pre[u]
                if (u == start):
                    graph[start][3] = orange
                    graphUI.updateUI()
                    break
            return

        adjecent_node = graph[current_node[0]][1]
        for next_node in adjecent_node:
            if (next_node not in close):
                if (next_node in open and fx[next_node] >
gx[current_node[0]] + dist(current_node[0], next_node,

graph) + hx[next_node]):
                    gx[next_node] = gx[current_node] +
dist(current_node[0], next_node, graph)
                    fx[next_node] = gx[next_node] + hx[next_node]
                    Pre[next_node] = current_node[0]
                    for node in open:
                        if (node[0] == next_node):
                            node[1] = fx[next_node]
                            node[2] = current_node[0]


                else:
                    graph[next_node][3] = red
                    edges[edge_id(current_node[0], next_node)][1] =
white
                    graphUI.updateUI()
                    gx[next_node] = gx[current_node[0]] +
dist(current_node[0], next_node, graph)
                    fx[next_node] = gx[next_node] + hx[next_node]
                    Pre[next_node] = current_node[0]
                    open.append((next_node, fx[next_node],
current_node[0]))
        graph[current_node[0]][3] = blue
```

```
        graphUI.updateUI()

    print("Don't Have path from ", start, " to ", goal)
    pass
```

d) Test Case



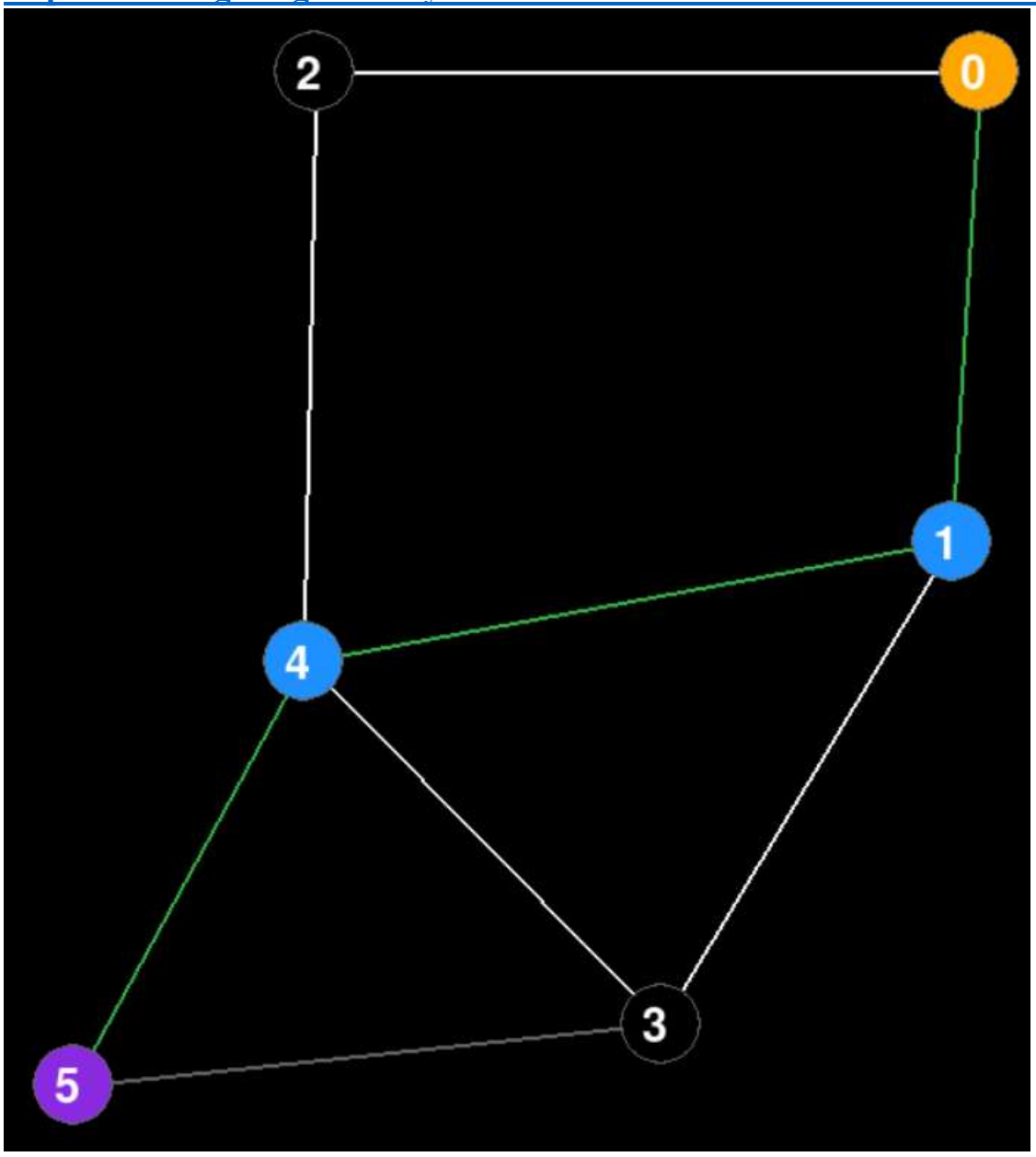
- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5

Mô tả:

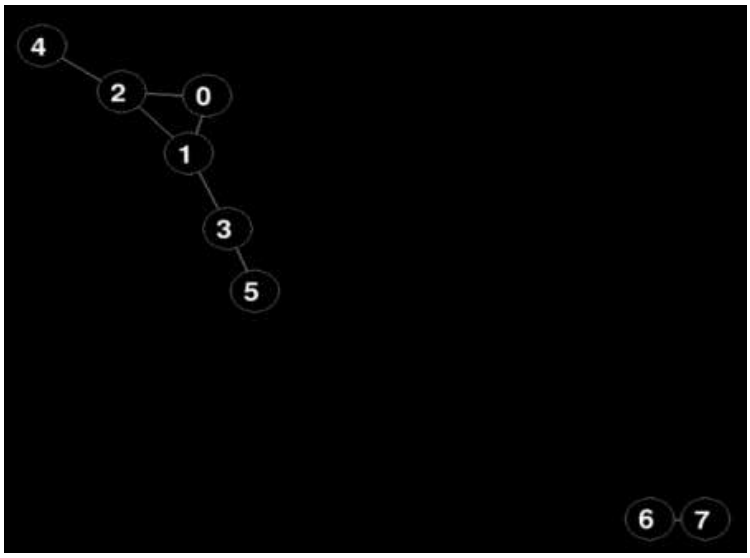
```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase1.txt', 'a_star']
Open =  [(0, 0, 0)]
Close=  []
Current_Node =  (0, 0, 0)
Open =  [(1, 815, 0), (2, 926, 0)]
Close=  [0]
Current_Node =  (1, 815, 0)
Open =  [(4, 873, 1), (3, 880, 1), (2, 926, 0)]
Close=  [0, 1]
Current_Node =  (4, 873, 1)
Open =  [(5, 873, 4), (3, 880, 1), (2, 926, 0), (3, 1208, 4), (2, 1497, 4)]
Close=  [0, 1, 4]
Current_Node =  (5, 873, 4)
```

Kết quả:
https://thuhangtnh-gmail.tinytake.com/tt/NDc4MzA0NF8xNTEwNzI2NA



- **TestCase 2**

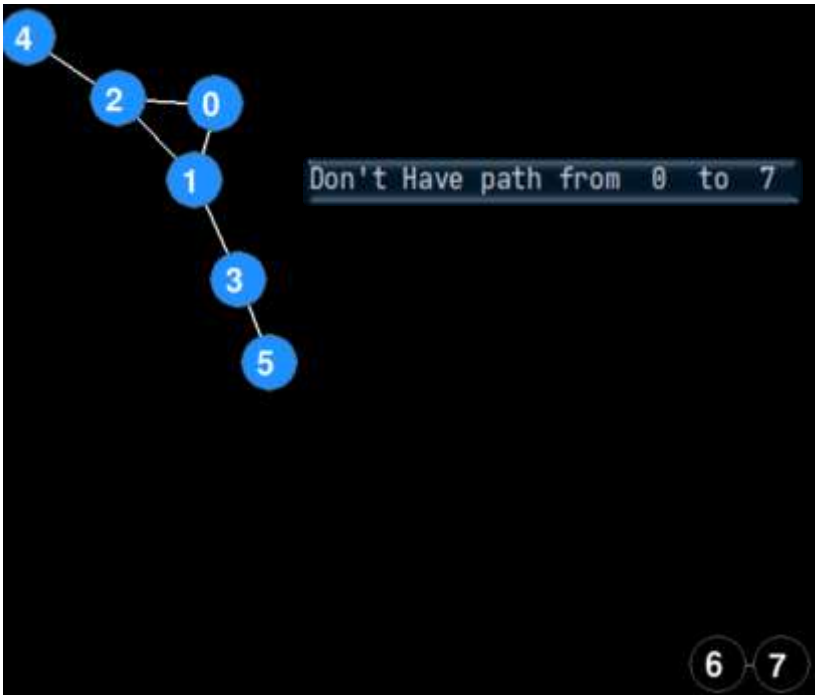
testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7



Mô tả:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase2.txt', 'a_star']
Open =  [(0, 0, 0)]
Close=  []
Current_Node =  (0, 0, 0)
Open =  [(1, 649, 0), (2, 748, 0)]
Close=  [0]
Current_Node =  (1, 649, 0)
Open =  [(3, 659, 1), (2, 748, 0), (2, 820, 1)]
Close=  [0, 1]
Current_Node =  (3, 659, 1)
Open =  [(5, 672, 3), (2, 748, 0), (2, 820, 1)]
Close=  [0, 1, 3]
Current_Node =  (5, 672, 3)
Open =  [(2, 748, 0), (2, 820, 1)]
Close=  [0, 1, 3, 5]
Current_Node =  (2, 748, 0)
Open =  [(2, 820, 1), (4, 988, 2)]
Close=  [0, 1, 3, 5, 2]
Current_Node =  (2, 820, 1)
Open =  [(4, 988, 2), (4, 988, 2)]
Close=  [0, 1, 3, 5, 2, 2]
Current_Node =  (4, 988, 2)
Open =  [(4, 988, 2)]
Close=  [0, 1, 3, 5, 2, 2, 4]
Current_Node =  (4, 988, 2)
Don't Have path from  0  to  7
```

Kết quả:
https://thuhangtnh-gmail.tinytake.com/tt/NDc4MzA1M18xNTEwNzI3Mg



e) Độ phức tạp:

Độ phức tạp: O(n)

f) Ưu – Khuyết điểm

Ưu điểm:

Một thuật giải linh động, tổng quát, trong đó hàm chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic khác. Nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói A* chính là thuật giải tiêu biểu cho Heuristic.

Khuyết điểm:

A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua.

5. Greedy Search

a) Ý tưởng:

Trong tìm kiếm tham lam, chúng tôi mở rộng nút gần nhất với nút mục tiêu. "Sự gần gũi" được ước tính bằng phương pháp heuristic h(x) .

***Heuristic*:** Một heuristic, h được định nghĩa như:
h(x) = Ước tính khoảng cách của nút x từ nút mục tiêu.
Giảm giá trị của h(x), gần hơn là nút từ mục tiêu.

Chiến lược: Mở rộng nút gần nhất với trạng thái mục tiêu, tức là mở rộng nút có hgiá trị thấp hơn .

b) Thuật giải:

Quy ước:

Open: là tập hợp chứa các (node, cost) đang chờ được xét. Trong trường hợp sử dụng thuật toán Greedy thì Open được tổ chức như một Priority Queue (Trong đó node là đỉnh chờ xét và cost là chi phí ước tính đến goal)

Close: là tập hợp các đỉnh đã được duyệt

start: là đỉnh bắt đầu

goal: là đỉnh kết thúc

current_Node: là đỉnh đang xét

cost_i: là chi phí đi từ current_Node đến i (i là các node kề của current_Node). Trong trường hợp này, chi phí là chi phỉ ước tính khoảng cách trong hệ tọa độ Oxy từ đỉnh I đến goal.

Mô tả :

Bước 1: Tập Open chứa đỉnh gốc (start,0) chờ được xét.

Bước 2: Kiểm tra tập Open có rỗng không

- Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét current_Node. Nếu current_Node là đỉnh goal cần tìm, kết thúc tìm kiếm.
- Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được thăm trước đó vào hàng đợi.

Bước 3: Đưa đỉnh current_Node vào tập Close, sau đó xác định các đỉnh kề với đỉnh current_Node vừa xét.Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open.

Bước 4: Nếu hàng đợi không rỗng thì quay về bước 2.

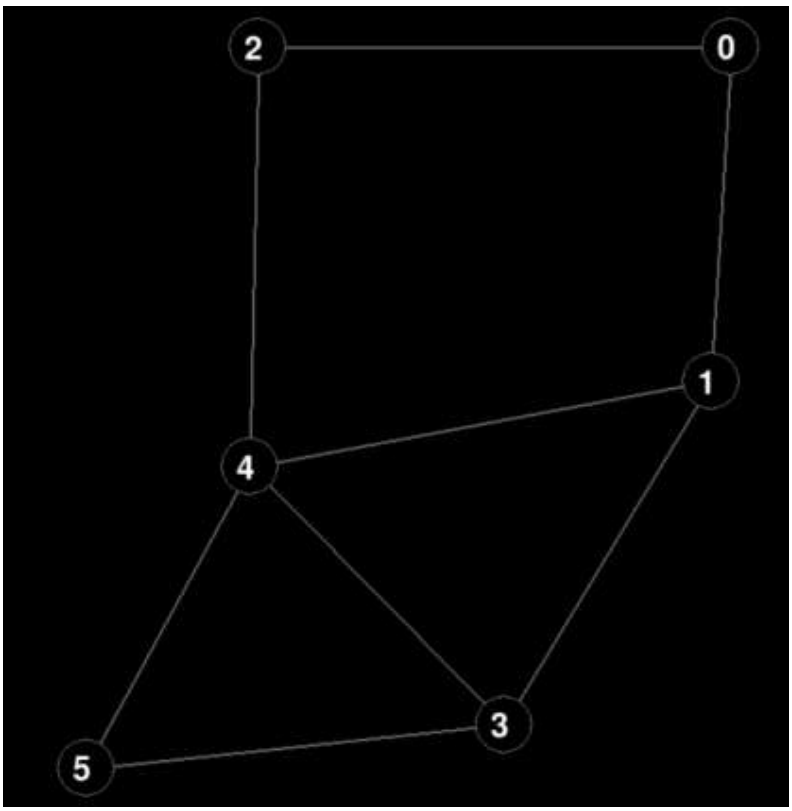
c) Mã nguồn

```
def GreedySearch(graph, edges, edge_id, start, goal):
    """
    Greedy Search
    """
    # TODO: your code

    Close = []
    Pre = [0] * 1000
    Open = queue.PriorityQueue()
    Open.put((dist(start,goal,graph), (start, start)))
    graph[start][3] = red
    graphUI.updateUI()
    while (Open.qsize() != 0):
        x = Open.get()
        cost = x[0]
        u = x[1][0]
        v = x[1][1]
        if (u not in Close):
            Close.append(u)
            Pre[u] = v
            if (u == goal):
                graph[goal][3] = purple
                graphUI.updateUI()
                Pre[u] = v
                while True:
                    edges[edge_id(u, Pre[u])][1] = green
                    graphUI.updateUI()
                    u = Pre[u]
                    if (u == start):
                        graph[start][3] = orange
                        graphUI.updateUI()
                        break
                return
            adjecent_node = graph[u][1]
            for i in adjecent_node:
                if (i not in Close):
                    Pre[i] = u
                    graph[i][3] = red
                    edges[edge_id(i, u)][1] = white
                    graphUI.updateUI()
                    x1 = graph[u][0]
                    x2 = graph[i][0]
                    cost_i = cost + dist(i,goal,graph)
                    Open.put((cost_i, (i, u)))
        graph[u][3] = blue
        graphUI.updateUI()

    print("Don't Have path from ", start, " to ", goal)
    pass
```

d) Test Case

- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5

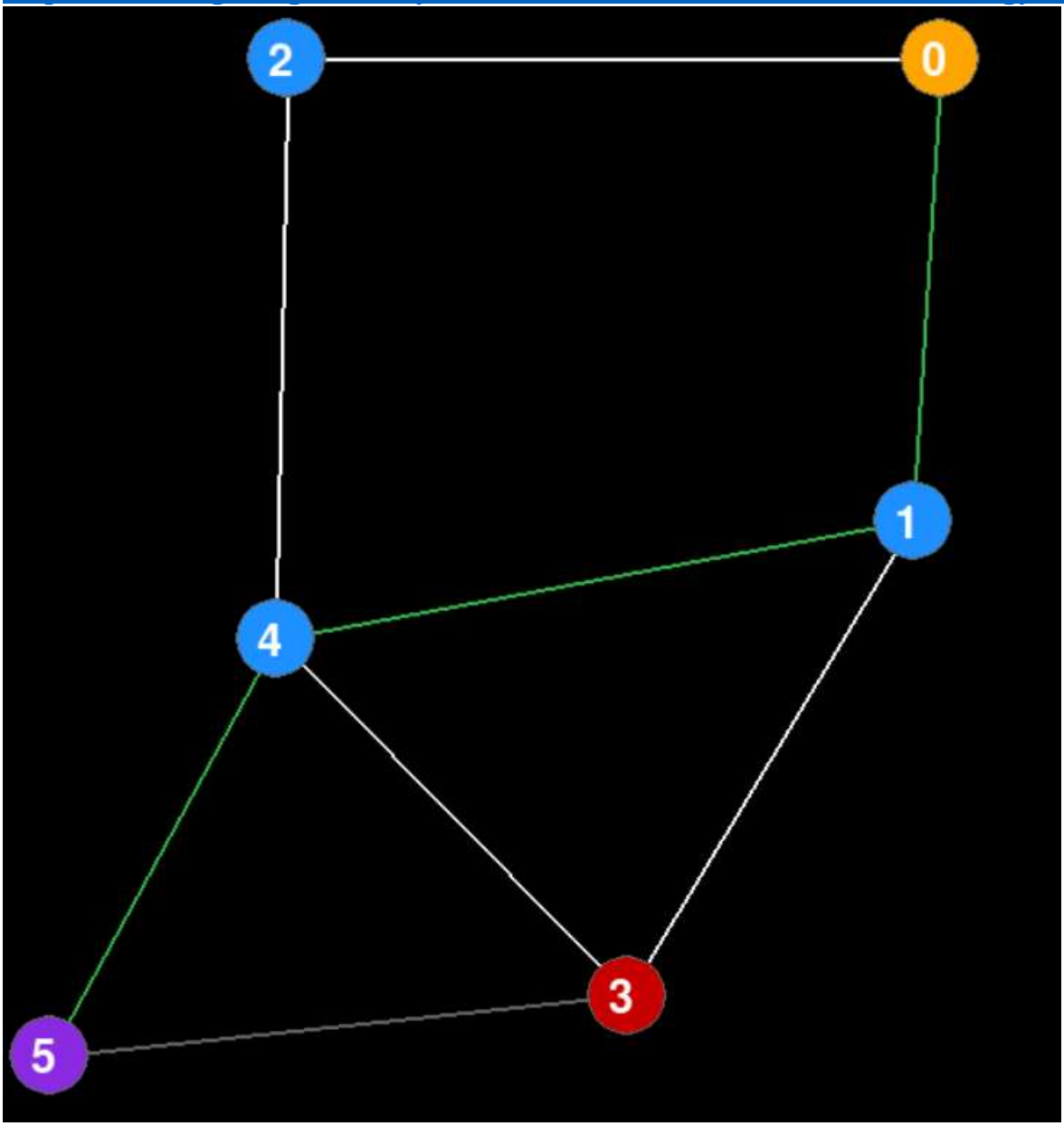


Mô tả:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase1.txt', 'GreedySearch']
Open[] = [
(737, (0, 0))
Close[] =  []
Current_Node =  0
Open[] = [
(1297, (1, 0))
(1302, (2, 0))
Close[] =  [0]
Current_Node =  1
Open[] = [
(1302, (2, 0))
(1558, (4, 1))
(1617, (3, 1))
Close[] =  [0, 1]
Current_Node =  2
Open[] = [
(1558, (4, 1))
(1563, (4, 2))
(1617, (3, 1))
Close[] =  [0, 1, 2]
Current_Node =  4
Open[] = [
(1558, (5, 4))
(1563, (4, 2))
(1617, (3, 1))
(1878, (3, 4))
Close[] =  [0, 1, 2, 4]
Current_Node =  5
```

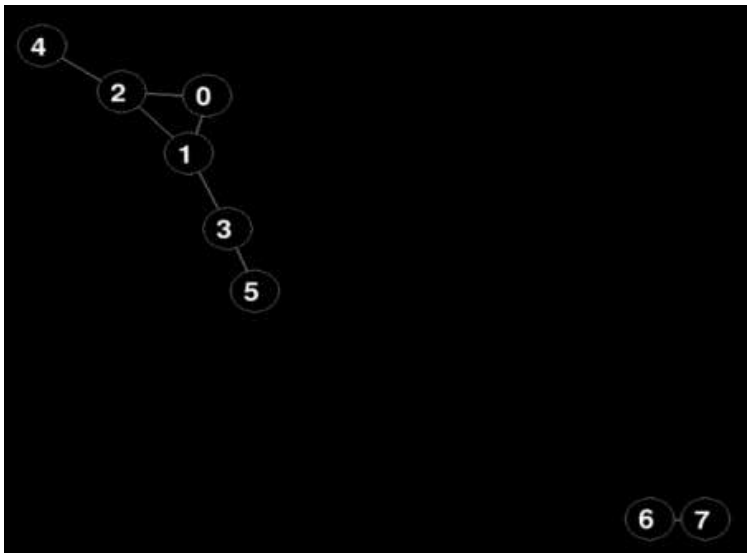
Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc4NDQ5M18xNTExMDgyOA



- **TestCase 2**

testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7

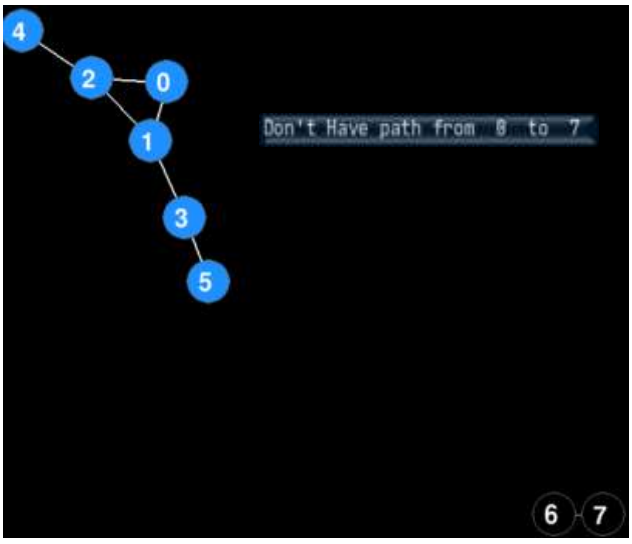


Mô tả:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase2.txt', 'GreedySearch']
Open[] = [
(615, (0, 0))
Close[] =  []
Current_Node =  0
Open[] = [
(1203, (1, 0))
(1288, (2, 0))
Close[] =  [0]
Current_Node =  1
Open[] = [
(1288, (2, 0))
(1717, (3, 1))
(1876, (2, 1))
Close[] =  [0, 1]
Current_Node =  2
Open[] = [
(1717, (3, 1))
(1876, (2, 1))
(2045, (4, 2))
Close[] =  [0, 1, 2]
Current_Node =  3
Open[] = [
(1876, (2, 1))
(2045, (4, 2))
(2176, (5, 3))
Close[] =  [0, 1, 2, 3]
Current_Node =  2
```

```
Open[] = [
(2045, (4, 2))
(2176, (5, 3))
Close[] =  [0, 1, 2, 3]
Current_Node =  4
Open[] = [
(2176, (5, 3))
Close[] =  [0, 1, 2, 3, 4]
Current_Node =  5
Don't Have path from  0  to  7
```

Kết quả:
https://thuhangtnh-gmail.tinytake.com/tt/NDc4NDUwNF8xNTExMDg1Mw



e) Độ phức tạp:

Độ phức tạp: O (m ^ (1 + tầng (l / e)))
trong đó,
m là số lân cận lớn nhất mà một nút có
l là độ dài của đường đi ngắn nhất đến trạng thái mục tiêu
e là chi phí cạnh nhỏ nhất

6. Best First Search

a) Ý tưởng:

Trong BFS và DFS, khi chúng ta đang ở một nút, chúng ta có thể coi bất kỳ nút nào liền kề là nút tiếp theo. Vì vậy, cả BFS và DFS đều khám phá các đường dẫn một cách mù quáng mà không xem xét bất kỳ hàm chi phí nào. Ý tưởng của Tìm kiếm đầu tiên tốt nhất là sử dụng chức năng đánh giá để quyết định vùng lân cận nào có triển vọng nhất và sau đó khám phá. Tìm kiếm đầu tiên tốt nhất thuộc danh mục Tìm kiếm theo kinh nghiệm hoặc Tìm kiếm có thông tin.

b) Thuật giải:

Best-First-Search (Grah g, Node start)

1) Tạo một PriorityQueue trống
   PriorityQueue pq ;
2) Chèn "start" trong pq.
   pq.insert (bắt đầu)
3) Cho đến khi PriorityQueue trống
   u = PriorityQueue.DeleteMin
   Nếu bạn là mục tiêu
   Lối ra
   Khác
   Báo trước hàng xóm v của bạn
   Nếu v "Không được mời"
   Đánh dấu v "Đã ghé thăm"
   pq.insert (v)
   Đánh dấu u là "Đã kiểm tra"

Kết thúc thủ tục

c) Mã nguồn

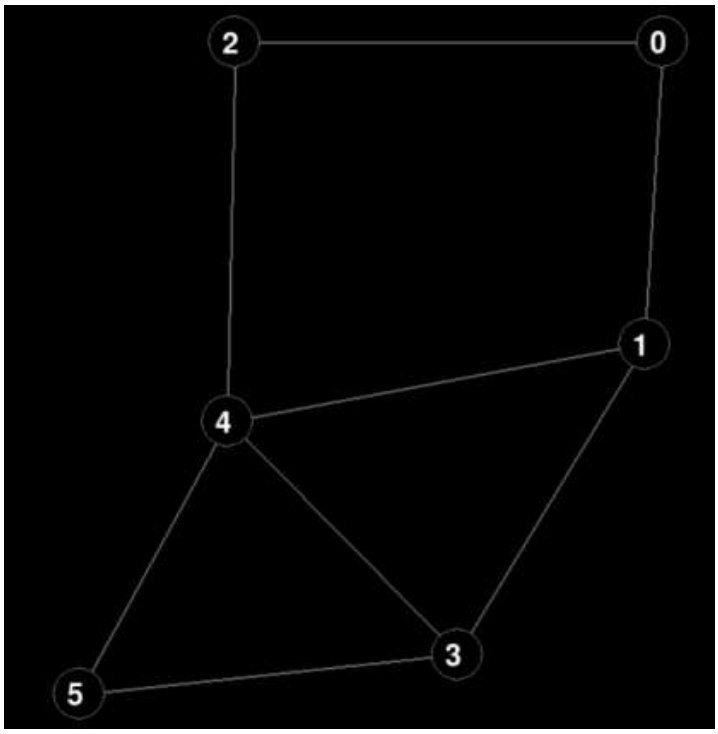
```
def BestFirstSearch(graph, edges, edge_id, start, goal):
    """
    Best First Search
    """
    Close = []
    Open = queue.PriorityQueue()
    Pre = [-1] * len(graph)

    Open.put((0,start))
    Pre[start]=start
    Close.append(start)
    graph[start][3]=red
    graphUI.updateUI()
    while (Open.qsize()!=0):
        current_node = Open.get()[1]
        graph[current_node][3]=yellow
```

```
        graphUI.updateUI()
        if (current_node==goal):
            graph[current_node][3] = purple
            graphUI.updateUI()
            while True:
                edges[edge_id(current_node, Pre[current_node])][1] =
green
                graphUI.updateUI()
                current_node = Pre[current_node]
                if (current_node == start):
                    graph[current_node][3] = orange
                    graphUI.updateUI()
                    break
            return
        adjecent_node = graph[current_node][1]
        for next_node in adjecent_node:
            if (next_node not in Close):
                graph[next_node][3]=red
                edges[edge_id(current_node,next_node)][1]=white
                graphUI.updateUI()

Open.put((dist(current_node,next_node,graph),next_node))
                Close.append(next_node)
                Pre[next_node]=current_node
        graph[current_node][3]=blue
        graphUI.updateUI()

    print("Don't Have path from ", start, " to ", goal)
    pass
```

d) Test Case

- **Test Case 1:**

testcase1.txt
0
5
0 1
0 2
1 3
1 4
2 4
3 4
3 5
4 5

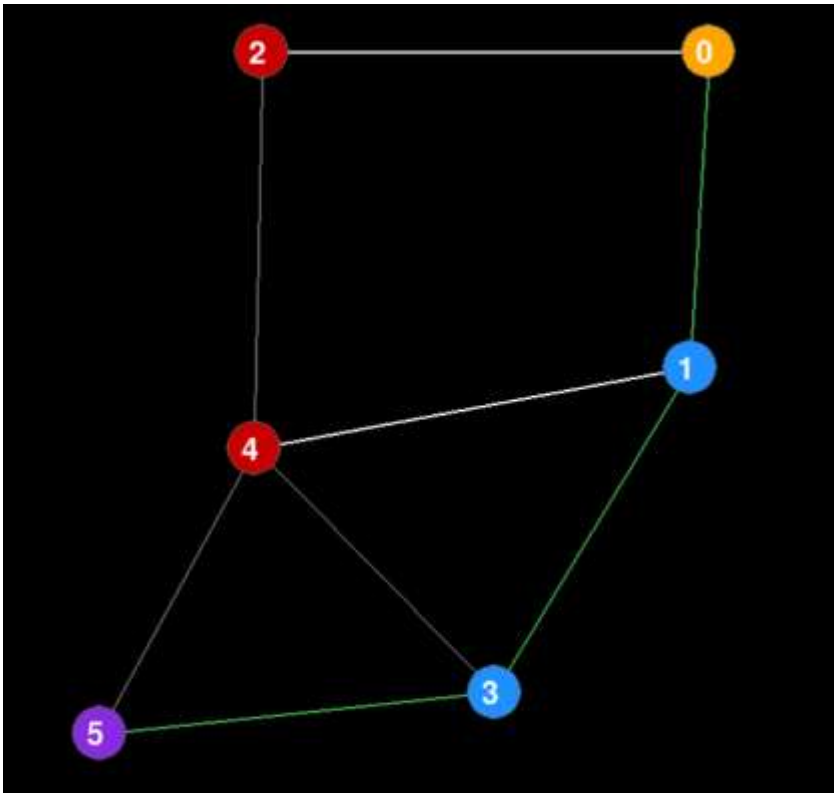


Mô tả:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase1.txt', 'BestFirstSearch']
Open[] = [
(0, 0)
]
Close[] =  [0]
Current_Node =  0
Open[] = [
(255, 1)
(361, 2)
]
Close[] =  [0, 1, 2]
Current_Node =  1
Open[] = [
(305, 3)
(357, 4)
(361, 2)
]
Close[] =  [0, 1, 2, 3, 4]
Current_Node =  3
Open[] = [
(320, 5)
(357, 4)
(361, 2)
]
Close[] =  [0, 1, 2, 3, 4, 5]
Current_Node =  5
```

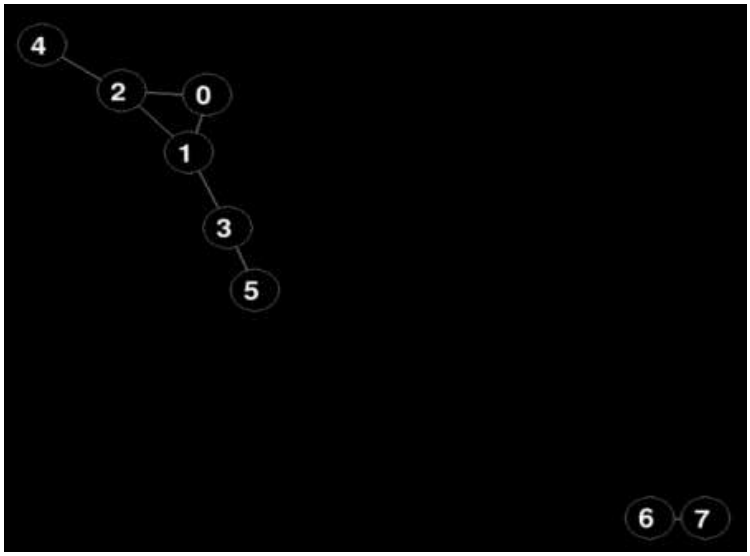
Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc4NDAyMF8xNTEwOTYwOA



- **TestCase 2**

testcase2.txt
0
7
0 1
0 2
1 2
1 3
2 4
3 5
6 7



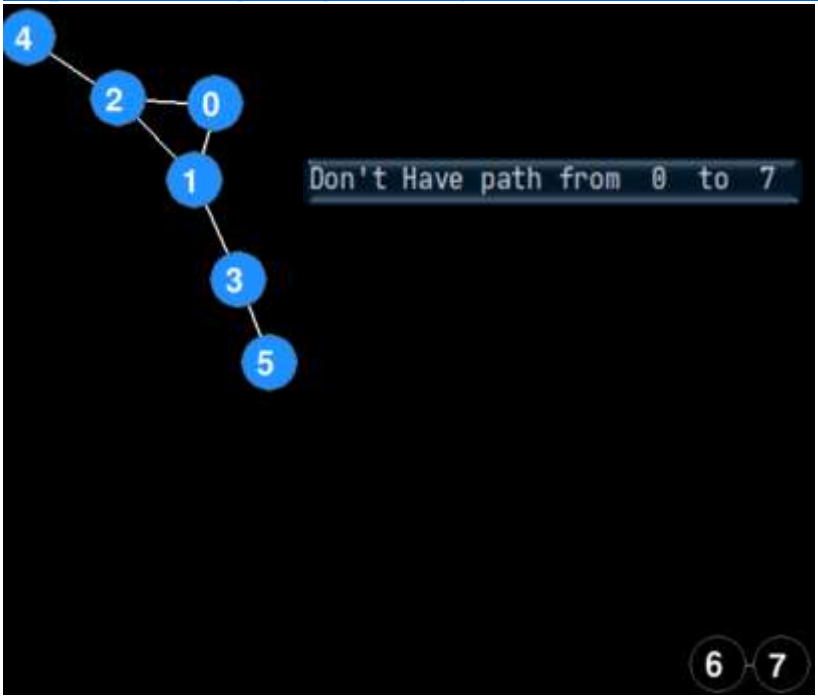
Mô tả:

```
Number of arguments: 3 arguments. Argument List: ['main.py', 'testcase2.txt', 'BestFirstSearch']
Open[] = [
(0, 0)
]
Close[] =  [0]
Current_Node =  0
Open[] = [
(61, 1)
(75, 2)
]
Close[] =  [0, 1, 2]
Current_Node =  1
Open[] = [
(75, 2)
(84, 3)
]
Close[] =  [0, 1, 2, 3]
Current_Node =  2
Open[] = [
(84, 3)
(84, 4)
]
Close[] =  [0, 1, 2, 3, 4]
Current_Node =  3
```

```
Open[] = [
(68, 5)
(84, 4)
]
Close[] =  [0, 1, 2, 3, 4, 5]
Current_Node =  5
Open[] = [
(84, 4)
]
Close[] =  [0, 1, 2, 3, 4, 5]
Current_Node =  4
Don't Have path from  0  to  7
```

Kết quả:

https://thuhangtnh-gmail.tinytake.com/tt/NDc4NDAzNF8xNTEwOTY1Mw



e) Độ phức tạp:

Độ phức tạp thời gian trong trường hợp xấu nhất cho Tìm kiếm đầu tiên tốt nhất là O (n * Nhật ký n) trong đó n là số nút. Trong trường hợp xấu nhất, chúng tôi có thể phải truy cập tất cả các nút trước khi đạt được mục tiêu. Lưu ý rằng hàng đợi ưu tiên được thực hiện bằng cách sử dụng Min (hoặc Max) Heap, và các hoạt động chèn và loại bỏ mất O (log n) thời gian.

.

III. TÀI LIỆU THAM KHẢO

1. https://www.stdio.vn/giai-thuat-lap-trinh/thuat-giai-a-DVnHj
2. https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
3. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
4. https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/
5. https://www.geeksforgeeks.org/search-algorithms-in-ai/
6. https://www.geeksforgeeks.org/best-first-search-informed-search/?ref=rp
7.