ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ TÀI

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Nhóm sinh viên thực hiện:

Họ và tên	***MSSV***	***Lớp***
Nguyễn Đặng Anh Thư	21280111	21KDL1B
Nguyễn Nhật Minh Thư	21280112	21KDL1B

TP.Hồ Chí Minh, 12 - 2022

Mục lục

Lời mở đầu

Lời đầu tiên, chúng em xin bày tỏ sự tôn trọng và biết ơn tới thầy Nguyễn Bảo Long - giảng viên phần Thực hành bộ môn Cấu trúc dữ liệu & giải thuật lớp 21KDL1 đã tạo cơ hội cho chúng em được học tập thông qua đồ án và giải đáp những thắc mắc của chúng em trong quá trình thực hiện.

Các thuật toán tìm kiếm trên đồ thị là một vấn đề cơ bản trong cấu trúc dữ liệu và thiết kế thuật toán. Mặc dù để giải quyết chung cho một bài toán thế nhưng để áp dụng cho từng trường hợp và từng bộ dữ liệu khác nhau (đồ thị vô hướng, đồ thị có hướng, đồ thị có trọng số âm, đồ thị có trọng số không âm,...), rất nhiều thuật toán tìm kiếm trên đồ thị đã được phát triển với đa dạng cách tiếp cận. Trong đồ án này, chúng em sẽ trình bày về ý tưởng, cách cài đặt và thời gian thực hiện của tổng cộng 5 thuật toán, cụ thể là Depth First Search, Breadth First Search, Uniform Cost Search, Greedy Best First Search và A* Search.

Chúng em cam kết rằng đây là thành quả nghiên cứu của riêng chúng em chứ không sao chép từ bất kỳ nguồn nào.

Ký tên

Nguyễn Nhật Minh Thư
Nguyễn Đặng Anh Thư

Phân công công việc và đánh giá

Công việc	Mô tả công việc	Thời gian	Người thực hiện	Đánh giá (Thang 10)
Report	- Tìm hiểu về thuật toán DFS, BFS, Greedy: Ý tưởng chung, mã giả, nhận xét và ví dụ - So sánh thuật toán DFS và BFS So sánh Uniformed và Informed Search	24/11 - 30/11	Anh Thư	10
	Tìm hiểu về thuật toán UCS, A*: Ý tưởng chung, mã giả, nhận xét và ví dụ - So sánh thuật toán Dijkstra, UCS So sánh Uniformed và Informed Search	24/11 - 30/11	Minh Thư	10
	Viết báo cáo bằng Latex: - Trang bìa, mục lục, lời cảm ơn, lời kết, tham khảo - Tự trình bày lại phần tìm hiểu và so sánh được giao - Nhận xét và mô tả ngắn gọn quá trình tìm kiếm của thuật toán	1/12 - 8/12	Cả nhóm	10
Cài đặt (Code)	Cài đặt thuật toán DFS và BFS: - Tìm đường đi giữa 2 node cho trước trong đồ thị - Cài đặt thông qua current, open_set và closed_set	1/12 - 7/12	Anh Thư	10
	Cài đặt thuật toán UCS & A* & Greedy: - Tìm đường đi giữa 2 node cho trước trong đồ thị - Cài đặt thông qua current, open_set và closed_set	1/12 - 7/12	Minh Thư	10
Video	- Video được phân đoạn quay quá trình chạy thuật toán - Video mỗi thuật toán không quá 3 phút - Upload Youtube và để đính kèm link trong file link.txt	8/12 - 9/12	Cả nhóm	10
Đánh giá tổng quan				10

Bảng 1: Bảng kế hoạch và phân công công việc đồ án

Chương I

Tìm hiểu và trình bày các thuật toán tìm kiếm trên đồ thị

1 Depth First Search

1.1 Ý tưởng chung

Thuật toán DFS sẽ ưu tiên duyệt theo chiều "sâu". Bắt đầu từ đỉnh nguồn, DFS sẽ duyệt các đỉnh thuộc cùng một nhánh đến khi không thể đi xa hơn được nữa, thì mới chuyển sang duyệt nhánh tiếp theo (backtracking). Thuật toán cứ tiếp tục đến khi tìm được đường đi đến đỉnh đích. Thuật toán này sử dụng chiến lược Last In First Out (LIFO) và do đó nó được cài đặt bằng cách sử dụng ngăn xếp (stack).

1.2 Mã giả

Mã giả: Depth First Search

Input: G Đồ thị cần tìm đường đi
start Đỉnh bắt đầu
goal Đỉnh đích

Output: Path Đường đi từ start tới goal (nếu có)

```
1:  procedure DFS(G, start, goal)
2:      open_set ← Stack //  First In Last Out
3:      //  Inserting start in stack until all its neighbour vertices are marked
4:      open_set.push( start )
5:      while open_set is not empty do
6:          //  Removing that vertex from stack, whose neighbour will be visited now
7:          current ← open_set.pop()
8:          mark current as closed
9:          if current is goal then
10:             return Path
11:         end if
12:         //  Processing all the neighbours of current
```

```
13:         for each neighbor w of current in G  do
14:             if w is not in closed then
15:                 open_set.push( w )
16:             end if
17:         end for
18:     end while
19:     return Not found path
20: end procedure
```

1.3 Nhận xét

a) Tính đầy đủ

Tính chất này của DFS phụ thuộc hoàn toàn vào cách cài đặt theo graph-search hay tree-search của thuật toán. Sự khác biệt cơ bản giữa graph-search và tree-search là graph-search có thêm một mảng phụ để lưu trữ những node đã được **"thăm"** (closed set) còn tree-search thì không.

- Trong tree-search (không có closed set), khi đó thuật toán DFS sẽ không quan tâm là node hiện tại (current node) đã được thăm hay chưa, mà DFS chỉ tiến hành đi càng sâu xuống → có thể dẫn tới vòng lặp vô tận → không đầy đủ

- Trong graph-search (có closed set), đánh dấu các node đã được thăm để tránh tình trạng thăm vô tận như trên → đầy đủ

b) Tính tối ưu

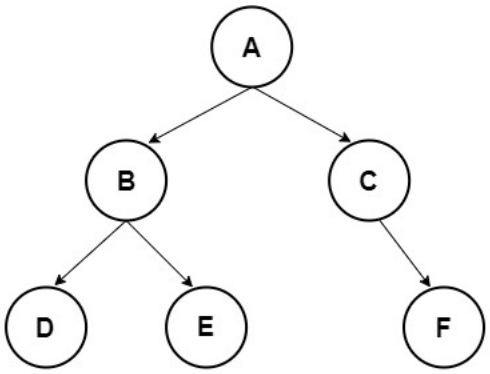
Thuật toán DFS không có tính tối ưu đối với cả hai cách cài đặt graph-search và tree-search. Thuật toán chỉ trả về kết quả là đường đi đến đỉnh đích đầu tiên mà nó tìm thấy, mà đường đi đó có thể không tối ưu. Điều này có thể xảy ra nếu có nhiều hơn một cách đi từ đỉnh nguồn đến đỉnh đích.

c) Độ phức tạp

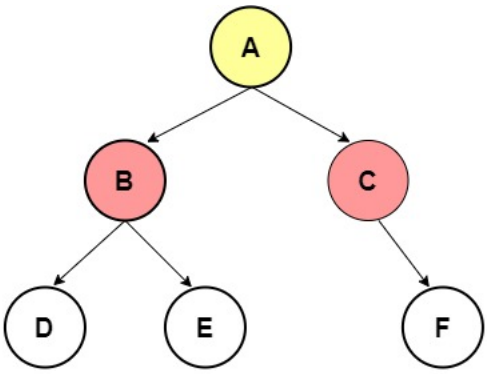
Trong trường hợp xấu nhất, DFS phải duyệt hết toàn bộ cây. Vì khi đó node đích cần tìm lại là node lá của nhánh cây được duyệt sau cùng (DFS cài đặt bằng đệ quy thì sẽ là nhánh bên phải, DFS cài đặt khử đệ quy bằng cách xét các node kề thì sẽ là nhánh bên trái).

Với **m** = độ sâu tối đa có thể duyệt từ một node và **b** = số node tối đa mà 1 node có thể mở rộng. Khi đó độ phức tạp về thời gian là: $T(b) = b^0 + b^1 + b^2 + ... + b^m = O(b^m)$.

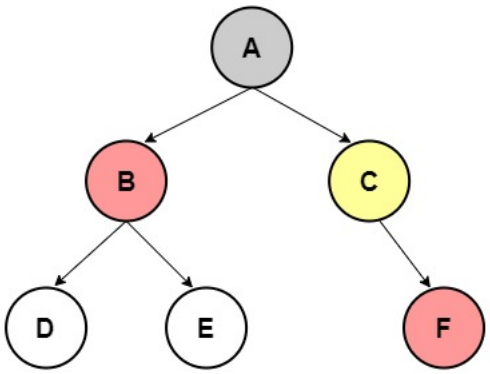
1.4 Ví dụ



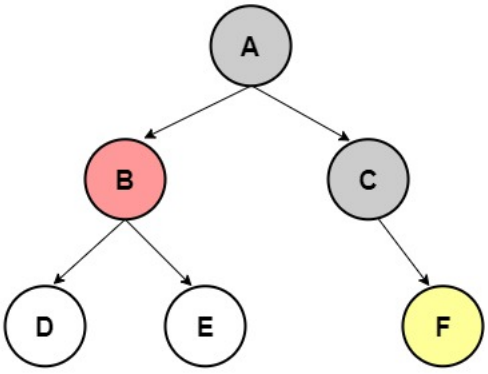
(0)



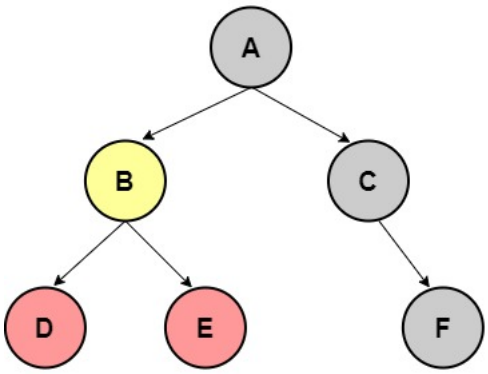
(1)



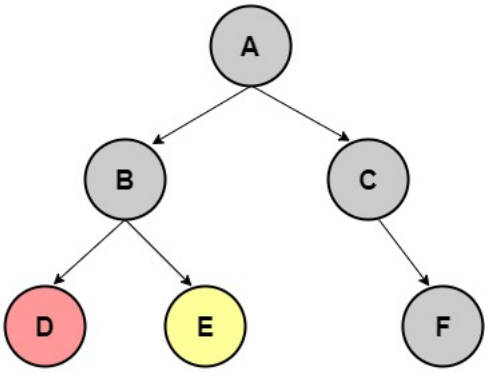
(2)



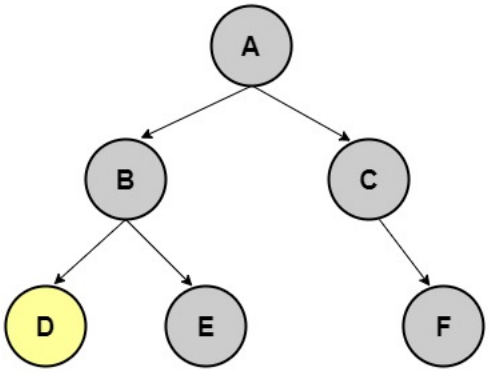
(3)



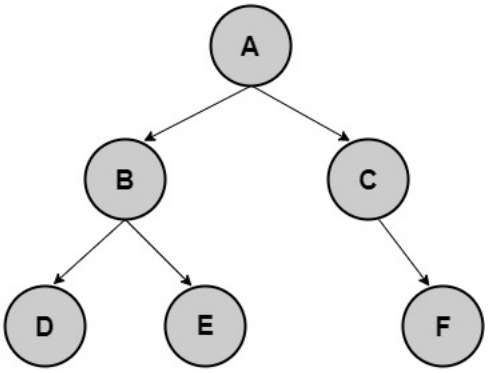
(4)



(5)



(6)



(7)

Step	Description	Current	Opened Set (Stack)	Closed Set
0	push(A) into empty Stack		A	
1	pop()= A, push(B) & push(C)	A	B, C	
2	marked A closed, pop()= C, push(F)	C	B, F	A
3	marked C closed, pop()= F	F	B	A, C
4	F closed, pop()= P, push(D), push(E)	B	D, E	A, C, F
5	marked B closed, pop()= E	E	D	A, C, F, B
6	marked E closed, pop()= D	D		A, C, F, B, E
7	Traversal: A - C - F - B - E - D			

Các bước duyệt cây bằng Depth First Search

Ví dụ ở đây là cách duyệt toàn bộ cây bằng thuật toán DFS, việc tìm kiếm đi từ node gốc đến node đích thì sẽ khác biệt một chút ở chỗ là khi node đang được duyệt (current) chính là node đích cần tìm thì ta sẽ kết thúc quá trình tìm kiếm ngay mà không cần duyệt tiếp nữa.

2 Breadth First Search

2.1 Ý tưởng chung

Ý tưởng của thuật toán BFS là bắt đầu từ đỉnh nguồn của đồ thị sau đó **"loang"** dần ra các đỉnh kề (có cạnh nối) với đỉnh nguồn, trước khi loang dần đến những đỉnh kề xa hơn. Những đỉnh nào gần với đỉnh nguồn hơn sẽ được loang tới trước. Thuật toán BFS được cài đặt bằng cách sử dụng hàng đợi (queue).

2.2 Mã giả

Mã giả: Breadth First Search

```
          G     Đồ thị cần tìm đường đi
Input:    start Đỉnh bắt đầu
          goal  Đỉnh đích
Output:   Path  Đường đi từ start tới goal (nếu có)
1:  procedure BFS(G, start, goal)
2:      open_set ← Queue
3:      // Inserting start in queue until all its neighbour vertices are marked
4:      open_set.enqueue( start )
5:      while open_set is not empty do
6:          // Removing that vertex from queue, whose neighbour will be visited now
7:          current ← open_set.dequeue()
8:          mark current as closed
9:          if current is goal then
10:             return Path
11:         end if
12:         // processing all the neighbours of current
13:         for all neighbours w of current in G do
14:             if w is not in closed and w is not opened then
15:                 open_set.enqueue( w )
16:             end if
17:         end for
18:     end while
19:     return Not found path
20: end procedure
```

2.3 Nhận xét

a) Tính đầy đủ

Thuật toán Breadth First Search (BFS) có tính đầy đủ, nếu b (= số node tối đa mà 1 node có thể mở rộng được) là hữu hạn. Khi đó BFS sẽ luôn tìm được đường đi từ node nguồn tới node đích, nếu tồn tại đường đi.

b) Tính tối ưu

Thuật toán BFS về cơ bản là không tối ưu vì nó không xét đến chi phí các cạnh để xác định cần thay thế node nào khi tìm kiếm đường đi. BFS chỉ đảm bảo được tính tối ưu khi đồ thị không có trọng số hoặc tất cả trọng số bằng nhau (chi phí tất cả các cạnh bằng nhau). Khi đó thuật toán này sẽ luôn tìm được đường đi ngắn nhất từ đỉnh nguồn tới đỉnh đích, nếu tồn tại đường đi.

c) Độ phức tạp

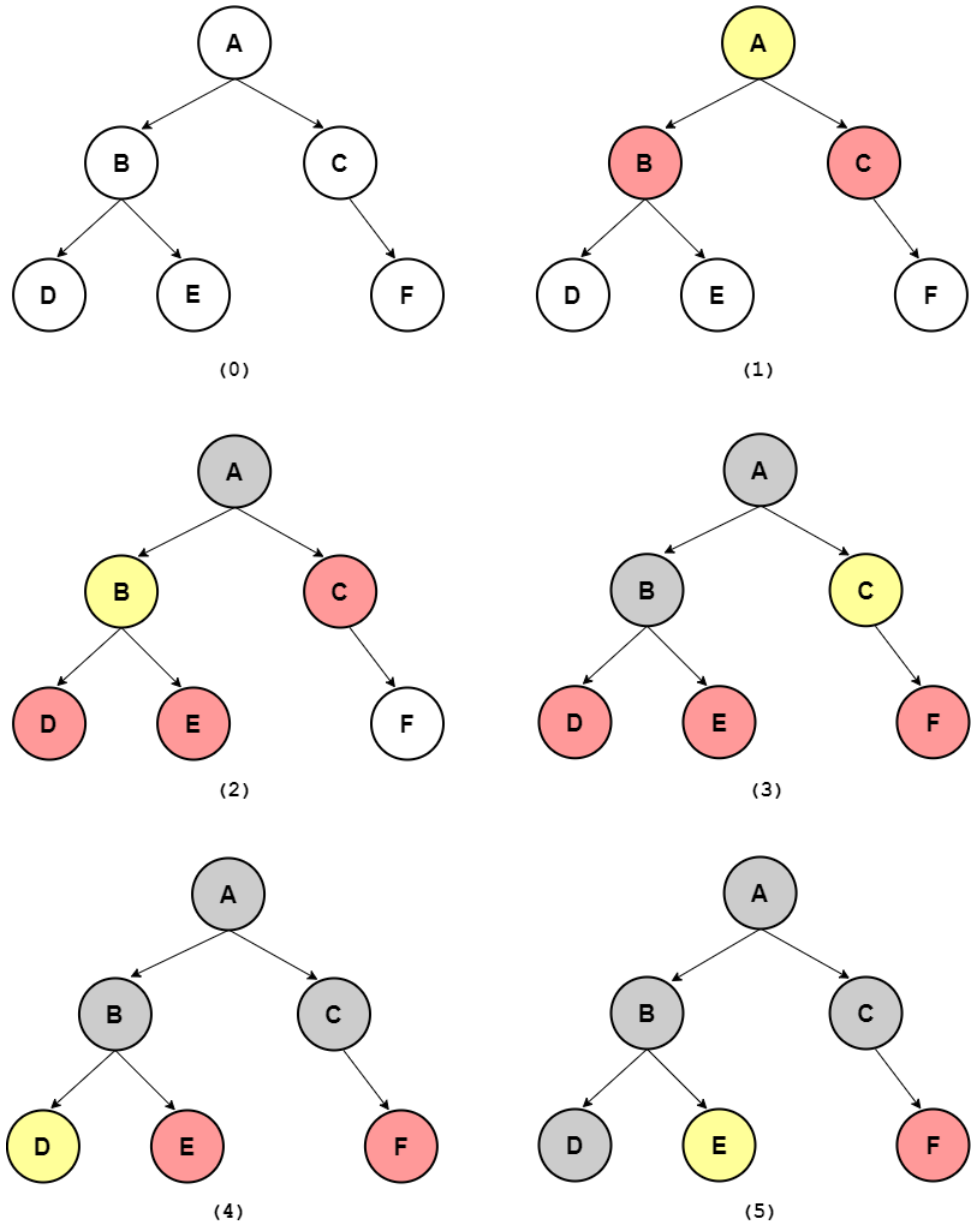
Trong trường hợp xấu nhất thì node đích là node cuối cùng được duyệt ở độ sâu **d**. Vậy nên

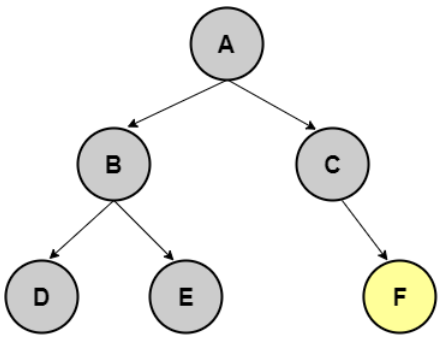
BFS sẽ phải duyệt hết tất cả các node từ độ sâu là 0 đến độ sâu của node đích.

Với **d** = độ sâu của node đích và **b** = số node tối đa mà 1 node có thể mở rộng. Khi đó độ phức tạp về thời gian là: $T(b) = b^0 + b^1 + b^2 + \ldots + b^d + b^{d+1} = O(b^{d+1})$

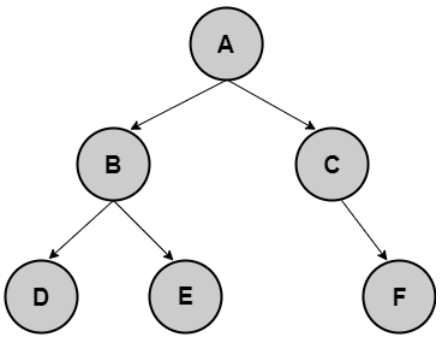
Tại sao lại là $d+1$? Vì như trong mã giả đã trình bày, chúng ta xét **current** có phải là **goal** hay không khi đã *dequeue* khỏi **opened_set**. Cho nên điều này khiến ta phải duyệt thêm một lớp nữa, chính vì vậy độ phức tạp phải là $O(b^{d+1})$ (Về cơ bản thì nó xấp xỉ $O(b^d)$ = Độ phức tạp thời gian như một số tài liệu khác có trình bày).

2.4 Ví dụ





(6)



(7)

Step	Description	Current	Opened Set (Queue)	Closed Set
0	enqueue(A) into empty Queue		A	
1	dequeue()= A, enqueue(B), enqueue(C)	A	B, C	
2	A closed, dequeue()= B, enqueue(D), enqueue(E)	B	C, D, E	A
3	B closed, dequeue()= C, enqueue(F)	C	D, E, F	A, B
4	C closed, , dequeue()= D	D	E, F	A, B, C
5	D closed, dequeue()= E	E	F	A, B, C, D
6	E closed, dequeue()= F	F		A, B, C, D, E
7	Traversal: A - B - C - D - E - F			

Các bước duyệt cây bằng Breadth First Search

Ví dụ ở đây là cách duyệt toàn bộ cây bằng thuật toán BFS, việc tìm kiếm đi từ node gốc đến node đích thì sẽ khác biệt một chút ở chỗ là khi node đang được duyệt (current) chính là node đích cần tìm thì ta sẽ kết thúc quá trình tìm kiếm ngay mà không cần duyệt tiếp nữa.

3 Uniform Cost Search

3.1 Ý tưởng chung

Uniform-cost search (UCS) hay Tìm kiếm chi phí cực tiểu áp dụng thuật toán Dijkstra. Nói qua về thuật toán Dijkstra là một trong những thuật toán cổ điển để giải quyết bài toán tìm

đường đi ngắn nhất từ một điểm cho trước tới tất cả các điểm còn lại trong đồ thị có trọng số không âm. Quay trở lại với UCS thì thuật toán này đánh giá chi phí và lựa chọn node tiếp theo với mục tiêu là chi phí đường đi là thấp nhất tính từ node gốc đến node tiếp theo được chọn. Hàm tính chi phí được ký hiệu là $g(x) =$ **khoảng cách giữa 2 node**. Quá trình này dừng lại cho đến khi ta tìm được đến node đích.

3.2 Mã giả

Mã giả: Uniform Cost Search

```
          G     Đồ thị cần tìm đường đi
Input:    start Đỉnh bắt đầu
          goal  Đỉnh đích
Output:   Path  Đường đi từ start tới goal (nếu có)
// Get distance from start to goal by Euclidean distance
procedure GET_DISTANCE(start, goal)
    return sqrt((goal.x − start.x)^2 + (goal.y − start.y)^2))
end procedure
// Implement Function
procedure UCS(G, start, goal)
    open_set ← Priority Queue //  (key, value) = (node, cost)
    open_set.enqueue( (start, 0 )
    while open_set is not empty do
        //  Removing node which lowest cost from priority queue
        current ← open_set.dequeue()
        if current is goal then
            return Path
        end if
        mark current as closed
        //  Processing all neighbors of current
        for each node of current's neighbors do
            if node is not in closed and node is not in opened then
                open_set.enqueue(node, cost + get_distance(current, node))
            else if node is in opened with higher cost
                replace lower cost //  = cost[current] + get_distance(current, node)
            end if
        end for
    end while
    return Not found path
end procedure
```

3.3 Nhận xét

a) Tính đầy đủ

Uniform Cost Search có tính đầy đủ khi đồ thị có trọng số là không âm. Nếu tồn tại đường đi có chi phí hữu hạn từ đỉnh nguồn đến đỉnh đích, UCS luôn trả về được kết quả là đường đi đó,

hơn nữa chi phí của đường đi này sẽ là ngắn nhất vì trong quá trình tìm kiếm đường đi UCS luôn duyệt hết tất cả các khả năng có thể và chọn ra đỉnh có chi phí nhỏ nhất để mở rộng.

b) Tính tối ưu

Uniform Cost Search (UCS) cũng có tính tối ưu khi trọng số tất cả các cạnh của đồ thị là không âm. Từ đỉnh nguồn, UCS sẽ mở rộng dần từng đỉnh trong đường đi bằng cách xét hết tất cả các đỉnh có thể đi và chọn ra đỉnh có tổng chi phí từ đỉnh nguồn đến đỉnh đó thấp nhất để mở rộng. Và khi trọng số tất cả các cạnh là không âm, thì chi phí đường đi đang xét sẽ ngày càng tăng dần, khi đó với chiến lược giải quyết như trên UCS sẽ đảm bảo trả về được đường đi với chi phí nhỏ nhất từ đỉnh nguồn tới đỉnh đích (nếu tồn tại đường đi). Chiến lược được sử dụng trong UCS cũng giống với chiến lược của thuật toán Dijkstra, sự khác biệt chính là UCS sẽ kết thúc khi tìm thấy đường đi ngắn nhất từ đỉnh nguồn tới đỉnh đích và tất cả những đỉnh đã được xét thay vì tìm thấy đường đi ngắn nhất từ đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị.

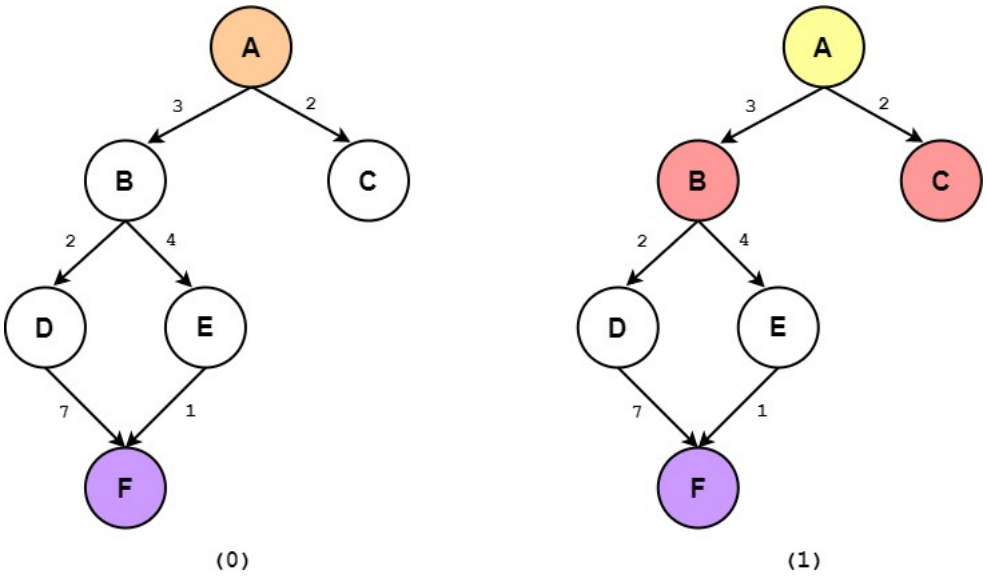
c) Độ phức tạp

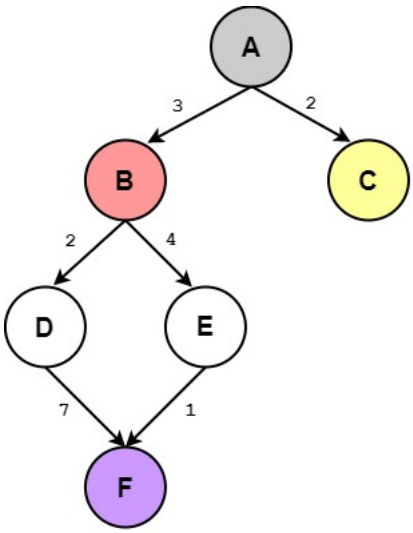
Uniform Cost Search không quan tâm đến tổng số bước của đường đi mà UCS chỉ quan tâm đến tổng chi phí của đường đi, vì vậy không thể đánh giá độ phức tạp của UCS chỉ dựa trên b (= số node tối đa mà 1 node có thể mở rộng) và d (= độ sâu của node đích). Thay vào đó, gọi:

- C* là chi phí đường đi tối ưu
- ϵ là chi phí tối thiểu cho mỗi lần mở rộng

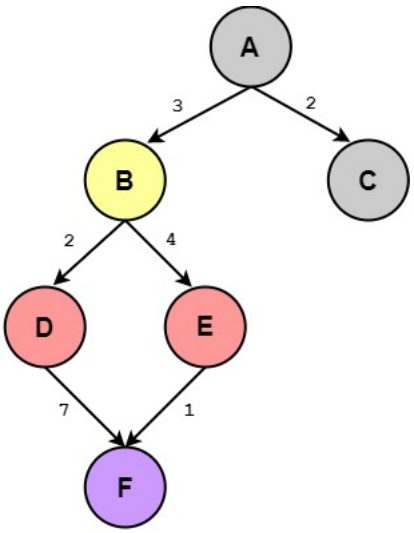
Khi đó, độ phức tạp của UCS trong trường hợp xấu nhất là $O(b^{1+\lfloor C*/\epsilon \rfloor}) = O(b^{d+1})$ khi chi phí tất cả các cạnh là bằng nhau.

3.4 Ví dụ

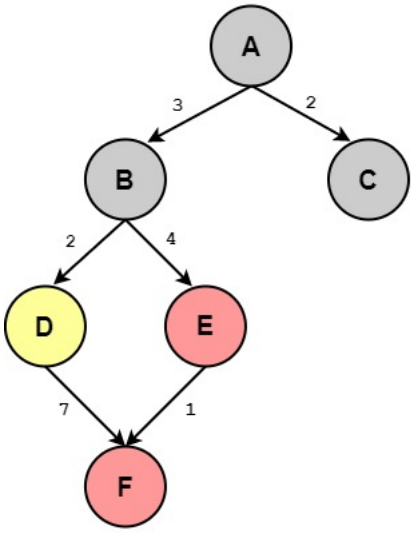



A
3
2
B
C
2
4
D
E
7
1
F

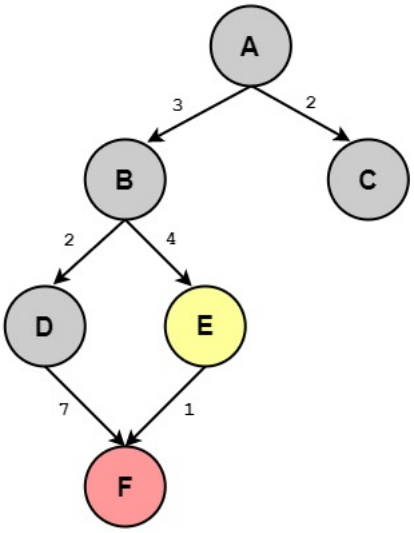
(2)


A
3
2
B
C
2
4
D
E
7
1
F

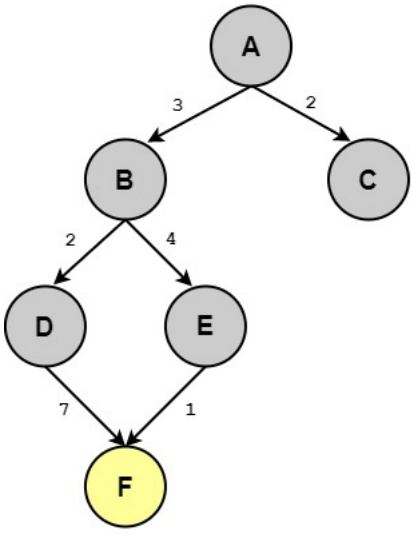
(3)


A
3
2
B
C
2
4
D
E
7
1
F

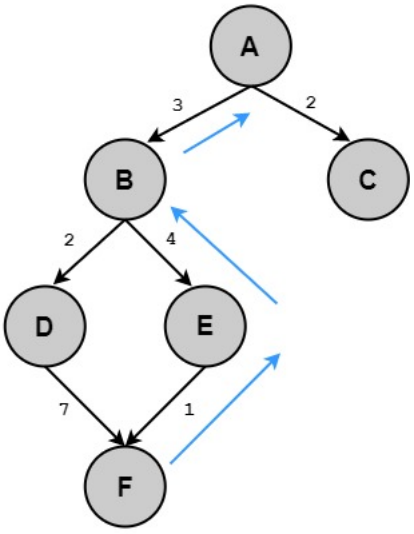
(4)


A
3
2
B
C
2
4
D
E
7
1
F

(5)


A
3
2
B
C
2
4
D
E
7
1
F

(6)


A
3
2
B
C
2
4
D
E
7
1
F

(7)

Step	Description	Current	Opened Set (Priority Queue)	Closed Set
0	enqueue(A,0) into empty Queue		{(A,0)}	{-}
1	dequeue()= (A,0), enqueue(B,3), enqueue(C,2)	(A,0)	{(C,2),(B,3)}	{-}
2	A closed, dequeue()= (C,2)	(C,2)	{(B,3)}	{(A,0)}
3	C closed, dequeue()= (B,3), enqueue(D,3+2), enqueue(E,3+4)	(B,3)	{(D,5),(E,7)}	{(A,0),(C,2)}
4	B closed, dequeue()= (D,5) enqueue(F,5+7)	(D,5)	{(E,7),(F,12)}	{(A,0),(C,2),(B,3)}
5	D closed, dequeue()= (E,7), replace (F,12) with (F,7+1)	(E,7)	{(F,8)}	{(A,0),(C,2),(B,3), (D,5)}
6	E closed, dequeue()= (F,8) = **goal** Return Path	(F,8)		{(A,0),(C,2),(B,3), (D,5),(E,7)}
7	Path A $\rightarrow$ F: A $\rightarrow$ B $\rightarrow$ E $\rightarrow$ F			

Các bước tìm kiếm bằng Uniform Cost Search

Để dễ hình dung về ví dụ trên, ta cần lưu ý một số điểm như sau:

- **Opened Set** được cài đặt bằng Priority Queue với phần tử có dạng (Vertex, Cost). Mức độ ưu tiên là Min Cost, tức là phần tử có Cost nhỏ nhất trong Priority Queue sẽ được lấy ra (dequeue) trước.
- Khi một phần tử đã được xét một (vài) lần không đồng nghĩa với việc sẽ không xét đến nữa như hai thuật toán DFS và BFS. Phần tử đó chỉ không được xét nữa khi được lấy ra (dequeue) từ **Opened Set**. Việc xét nhiều lần và có thể gán lại Cost vì có thể tồn tại đường đi khác với chi phí thấp hơn chi phí ban đầu được gán.

4 Greedy Best First Search

4.1 Ý tưởng chung

Greedy Best First Search là một thuật toán tìm kiếm đồ thị nằm trong nhóm Tìm kiếm dựa

trên kinh nghiệm (Informed Search) với chiến lược tìm kiếm dùng tri thức bổ sung từ việc sử dụng các tri thức cụ thể của bài toán.

Thuật toán sẽ sử dụng một hàm đánh giá là hàm heuristic $h(x)$ để đánh giá chi phí để đi từ node hiện tại x. Trong quá trình tìm kiếm, Greedy best-first search sẽ mở rộng node "có vẻ" gần với node đích (mục tiêu) nhất.

4.2 Mã giả

Mã giả: Greedy Best First Search

Input: G Đồ thị cần tìm đường đi
start Đỉnh bắt đầu
goal Đỉnh đích

Output: Path Đường đi từ start tới goal (nếu có)

```
// Get heuristic from start to goal by Euclidean distance
procedure GET_HEURISTIC(node)
    return sqrt((goal.x − node.x)^2 + (goal.y − node.y)^2))
end procedure
// Implement Function
procedure GREEDY(G, start, goal)
    open_set ← Priority Queue //  (key, value) = (node, heuristic)
    open_set.enqueue( (start, get_heuristic(start) )
    while open_set is not empty do
        //  Removing node which lowest heuristic from priority queue
        current ← open_set.dequeue()
        if current is goal then
            return Path
        end if
        mark current as closed
        //  Processing all neighbors of current
        for each node of current's neighbors do
            if node is not in closed and node is not in opened then
                open_set.enqueue(node, get_heuristic(node))
            end if
        end for
    end while
    return Not found path
end procedure
```

4.3 Nhận xét

a) Tính đầy đủ

Tương tự như BFS, tính chất này của Greedy BFS cũng phụ thuộc vào cách cài đặt của thuật toán.

- Nếu cài đặt không có closed set, khi đó thuật toán Greedy BFS sẽ không quan tâm là node hiện tại (current node) đã được thăm hay chưa, mà Greedy BFS chỉ tiến hành mở

rộng những đỉnh có chi phí gần với đỉnh đích nhất → có thể dẫn tới vòng lặp vô tận → không đầy đủ

- Nếu cài đặt có closed set, đánh dấu các node đã được thăm để tránh tình trạng thăm vô tận như trên → đầy đủ

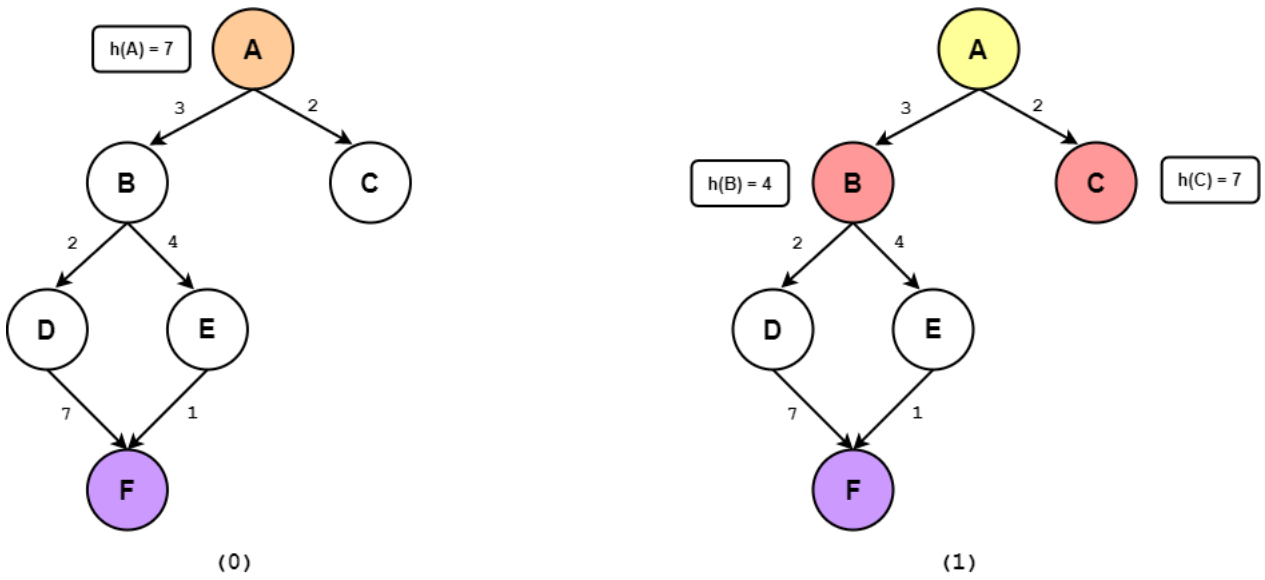
b) Tính tối ưu

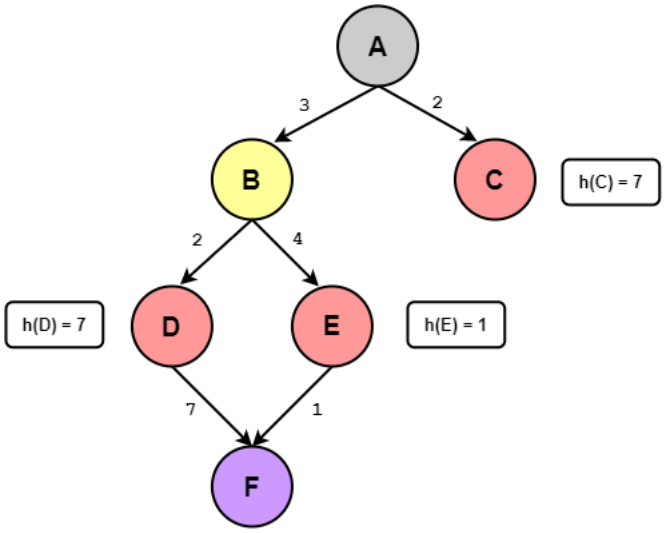
Thuật toán Greedy Best First Search không có tính tối ưu. Hàm heuristic $h(x)$ của Greedy BFS chỉ đơn giản chọn node "có vẻ" gần node nhất từ node x hiện tại để mở rộng. Điều này đôi khi sẽ trả về con đường có chi phí tối ưu, nhưng cũng có thể xảy ra vấn đề ở trường hợp khác. Ví dụ: trên bản đồ, hai điểm "có vẻ" rất gần nhau khi chỉ tính đến khoảng cách, nhưng thực tế việc di chuyển giữa chúng có thể cần thêm thời gian hoặc thậm chí có thể là không thể vì gặp vật cản. Hàm heuristic này được gọi là tham lam bởi vì, theo một nghĩa nào đó, nó tham lam "chộp" lấy giải pháp tốt nhất mà không cố gắng tính toán chi phí dài hạn. Đặc điểm nổi bật của nó là trong quá trình tìm kiếm đường đi đến node đích, nó bắt đầu từ node hiện tại chứ không phải là node xuất phát.

c) Độ phức tạp

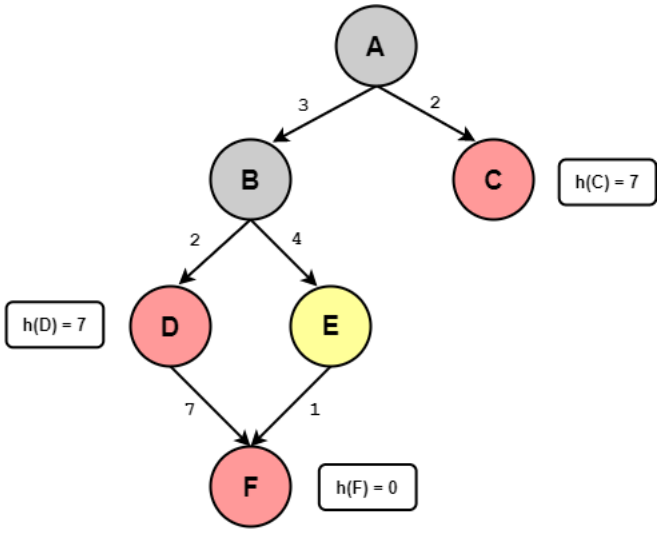
Độ phức tạp thời gian trong trường hợp xấu nhất của Greedy Best First Search là $O(b^m)$, trong đó **m** là độ sâu tối đa của không gian tìm kiếm. Tuy nhiên, với một hàm heuristic tốt có thể mang lại cải thiện lớn. Độ cải thiện phụ thuộc vào từng vấn đề cụ thể và vào chất lượng của hàm heuristic.

4.4 Ví dụ

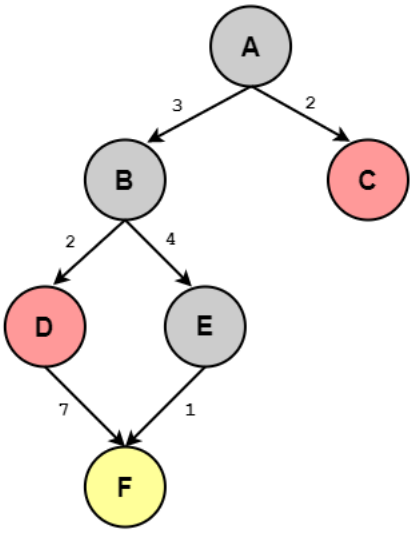




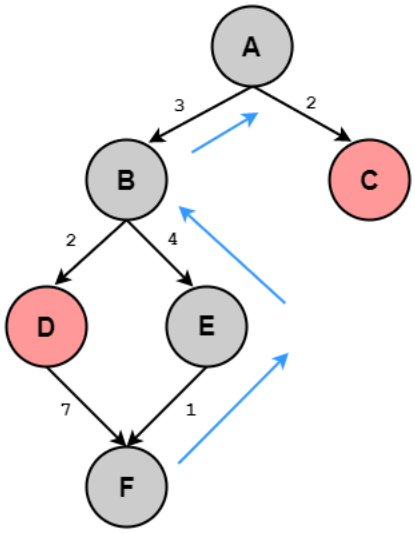
(2)



(3)



(4)



(5)

Step	Description	Current	Opened Set (Priority Queue)	Closed Set
0	enqueue(A,7) into empty Queue		{(A,7)}	{-}
1	dequeue()= (A,7), enqueue(B,4), enqueue(C,7)	(A,7)	{(B,4),(C,7)}	{-}
2	A closed, dequeue()= (B,4) enqueue(D,7), enqueue(E,1)	(B,7)	{(E,1),(C,7) (D,7)}	{(A,7)}
3	B closed, dequeue()= (E,1), enqueue(F,0)	(E,1)	{(F,0),(C,7), (D,7)}	{(A,7),(B,4)}

4	E closed, dequeue()= (F,0) = **goal** Return Path	(F,0)	{(C,7),(D,7)}	{(A,7),(B,4),(E,1)}
5	Path A → F: A → B → E → F			

Các bước tìm kiếm bằng Greedy Best First Search

Để dễ hình dung về ví dụ trên, ta cần lưu ý một số điểm như sau:

- Mức độ ưu tiên để lấy ra phần tử nào là giá trị $h(x)$ = Ước lượng khoảng cách giữa node đang xét và node đích.
- Cách duyệt cũng tương tự với thuật toán UCS là duyệt theo giá trị nhỏ nhất lấy ra (dequeue) được từ **Opened Set**.

5 A* Search

5.1 Ý tưởng chung

A* Search là một thuật toán tìm kiếm trong đồ thị. Thuật toán sẽ tìm kiếm đường đi từ đỉnh nguồn đến đỉnh đích sao cho chi phí là thấp nhất và số bước duyệt là ít nhất bằng cách sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

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

- $g(x)$ là chi phí đường đi từ đỉnh nguồn cho đến đỉnh x hiện tại.
- $h(x)$ là chi phí được ước lượng để đến đích từ x.
- $f(x)$ là chi phí tổng thể ước lượng của đường đi qua đỉnh x hiện tại để đến đích. $f(x)$ có giá trị càng thấp thì độ ưu tiên của x càng cao.

5.2 Mã giả

Mã giả: A* Search

Input:	G	Đồ thị cần tìm đường đi
	start	Đỉnh bắt đầu
	goal	Đỉnh đích
Output:	Path	Đường đi từ start tới goal (nếu có)

```
1: // Get distance from start to goal by Euclidean distance
2: procedure GET_DISTANCE(start, goal)
3:     return sqrt((goal.x − start.x)^2 + (goal.y − start.y)^2))
4: end procedure
5: procedure GET_HEURISTIC(node)
6:     return abs(goal.x − node.x)+abs(goal.y − node.y) // Diagonal distance
7: end procedure
8: // Implement Function
9: procedure ASTAR(G, start, goal)
10:     open_set ← Priority Queue //  (key, value) = (node, fcost), fcost = f(n) = g(n) +h(n)
11:     open_set.enqueue(start, get_heuristic(start))
12:     while open_set is not empty do
13:         //  Removing node which lowest fcost from priority queue
14:         current ← open_set.dequeue()
15:         if current is goal then
16:             return Path
17:         end if
18:         mark current as closed
19:         //  Processing all neighbors of current
20:         for each node of current's neighbors do
21:             if node is not in closed then
22:                 totalcost ← gcost[current] + get_distance(current, node)
23:                 if node  not in  opened_set then
24:                     opened_set.enqueue((node, totalcost + get_heuristic(node)))
25:                 else if totalcost < gcost[node]
26:                     gcost[node] ← totalcost
27:                     // replace by lower fcost
28:                     replace (node, fcost) with  (node, totalcost+get_heuristic(node))
29:                 end if
30:             end if
31:         end for
32:     end while
33:     return Not found path
34: end procedure
```

5.3 Nhận xét

a) Tính đầy đủ

A* Search có tính đầy đủ nếu đồ thị là hữu hạn với chi phí giữa tất cả các cạnh trong đồ thị là không âm. Điều kiện để đảm bảo tính đầy đủ của thuật A* Search tương tự với điều kiện của thuật toán Uniform Cost Search.

b) Tính tối ưu

A* Search có tính tối ưu khi hàm heuristic $h(x)$ là **admissible**, khi đó $h(x)$ phải thỏa: $\forall x$, $h(x) \leq h^*(x)$. Trong đó:

- x là 1 đỉnh trong đồ thị.

- h là hàm heuristic

- $h(x)$ là chi phí được ước lượng bởi h để đến đích từ x.

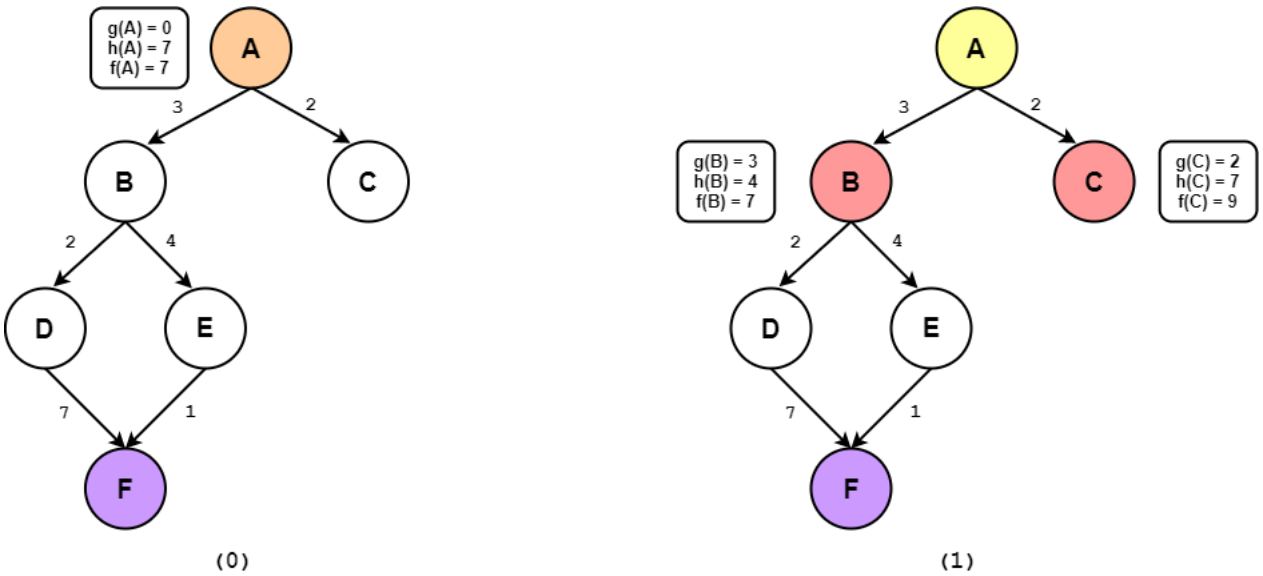
- $h^*(x)$ là chi phí tối ưu thực tế để đến đích từ x

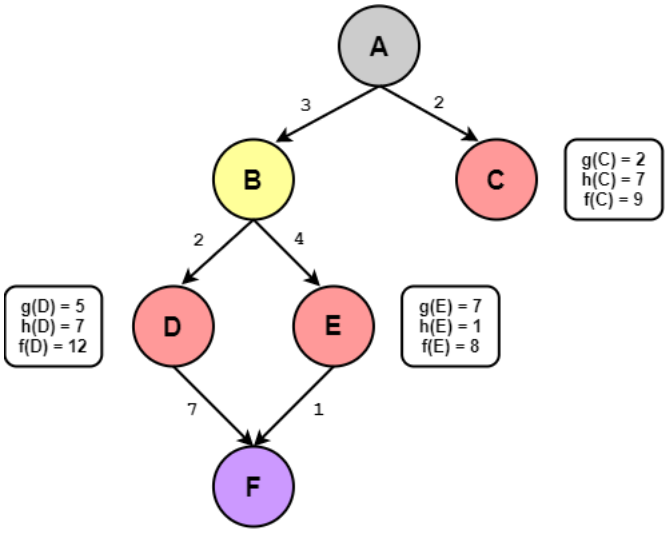
Bởi vì $g(x)$ là chi phí thực tế để đi từ đỉnh xuất phát đến đỉnh x hiện tại và $f(x) = g(x) + h(x)$, do đó $h(x)$ phải thỏa mãn điều kiện trên để chi phí ước lượng tổng thể $f(x)$ không vượt quá chi phí tối ưu thực tế của đường đi từ đỉnh xuất phát đến đỉnh đích qua đỉnh x.

c) Độ phức tạp

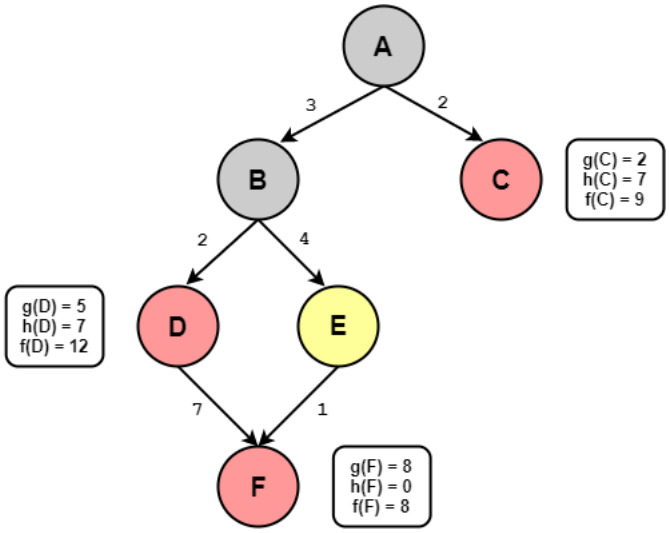
Độ phức tạp thời gian của thuật toán A* Search phụ thuộc vào hàm heuristic được sử dụng. Một hàm heuristic tốt sẽ giúp A* loại bỏ nhiều node không cần thiết phải mở rộng thay vì cứ mở rộng hết tất cả các node có thể như các thuật toán uninformed search. Trong trường hợp xấu nhất, hàm heuristic được sử dụng hoàn toàn **mù** thông tin và kiến thức, khi đó A* hoạt động giống như BFS với độ phức tạp = $O(b^d)$, với **d** = độ sâu của node đích và **b** = số node tối đa mà 1 node có thể mở rộng

5.4 Ví dụ

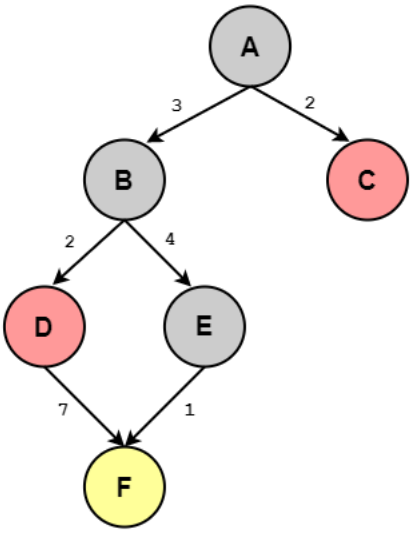




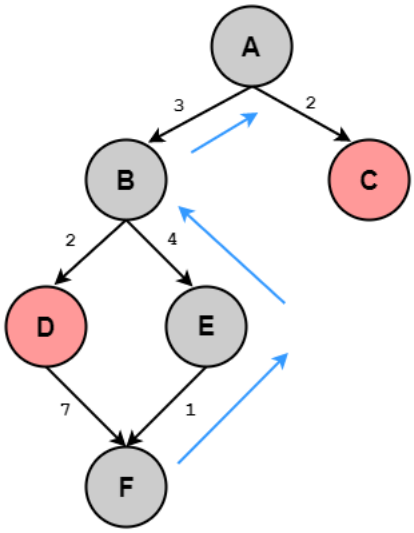
(2)



(3)



(4)



(5)

Step	Description	Current	Opened Set (Priority Queue)	Closed Set
0	enqueue(A,7) into empty Queue		{(A,7)}	{-}
1	dequeue()= (A,7), enqueue(B,7), enqueue(C,9)	(A,7)	{(B,7),(C,9)}	{-}
2	A closed, dequeue()= (B,7) enqueue(D,12), enqueue(E,8)	(B,7)	{(E,8),(C,9), (D,12)}	{(A,7)}
3	B closed, dequeue()= (E,8), enqueue(F,8)	(E,8)	{(F,8),(C,9), (D,12)}	{(A,7),(B,7)}

4	E closed, dequeue()= (F,8) = **goal** Return Path	(F,8)	{(E,7),(F,12)}	{(A,7),(B,7),(E,8)}
5	Path A $\rightarrow$ F: A $\rightarrow$ B $\rightarrow$ E $\rightarrow$ F			

Các bước tìm kiếm bằng Greedy Search

Để dễ hình dung về ví dụ trên, ta cần lưu ý một số điểm như sau:

- Một vài lưu ý tương tự UCS đã trình bày như ở trên. Đặc biệt mức độ ưu tiên để lấy ra phần tử nào là giá trị $f(x) = g(x) + h(x)$.
- Khi có 2 phần tử có giá trị $f(x)$ bằng nhau thì A* sẽ hoạt động giống như DFS giữa các phần tử đó để tránh khám phá nhiều hơn một giải pháp tối ưu.

Chương II

So sánh các thuật toán

1 Sự khác biệt giữa Depth First Search (DFS) và Breadth First Search (BFS)

Tiêu chí so sánh	**Depth First Search (DFS)**	**Breadth First Search (BFS)**
Cấu trúc dữ liệu sử dụng	Duyệt lần lượt tất cả các node trên cùng một nhánh cho đến khi không còn đi sâu hơn được nữa, sau đó mới chuyển sang nhánh kế tiếp	Duyệt lần lượt tất cả các node trên cùng mức sau đó mới chuyển sang duyệt mức kế tiếp
Bộ nhớ	Ít tốn bộ nhớ hơn	Tốn bộ nhớ hơn
Thời gian	$O(b^m)$ - b = số node tối đa mà 1 node có thể mở rộng - m = độ sâu tối đa	$O(b^s)$ - b = số node tối đa mà 1 node có thể mở rộng - s = độ sâu của node đích
Tính tối ưu	Không tối ưu	Tối ưu
Khi nào nên dùng?	Khi node đích ở xa node nguồn, DFS hoạt động tốt hơn	Khi node đích ở gần node nguồn, BFS hoạt động tốt hơn

2 Sự khác biệt giữa Uniform Cost Search (UCS) và Dijkstra

Cả thuật toán Dijkstra và thuật toán Uniform Cost Search đều có hướng giải quyết bài toán tìm đường đi ngắn nhất tương tự nhau với cùng độ phức tạp về thời gian. Hai thuật toán cũng sử dụng cùng một công thức, $dist[v] = min(dist[v], dist[u] + w(u, v))$, để cập nhật giá trị khoảng cách của mỗi đỉnh.

Tiêu chí so sánh	Uniform Cost Search (UCS)	Dijkstra
Cài đặt	Dừng khi tìm được đường đi từ đỉnh nguồn tới đỉnh đích	Dừng khi tìm được đường đi từ đỉnh nguồn tới tất cả các đỉnh còn lại trong đồ thị
Yêu cầu về bộ nhớ	Chỉ lưu trữ những đỉnh cần thiết	Phải lưu trữ toàn bộ đồ thị
Tốc độ chạy	Nhanh hơn vì ít tốn bộ nhớ hơn	Chậm hơn vì tốn bộ nhớ hơn
Ứng dụng	Ứng dụng với cả đồ thị tường minh (explicit graph) và đồ thị không tường minh (implicit graph)	Chỉ ứng dụng với đồ thị tường minh (explicit graph)

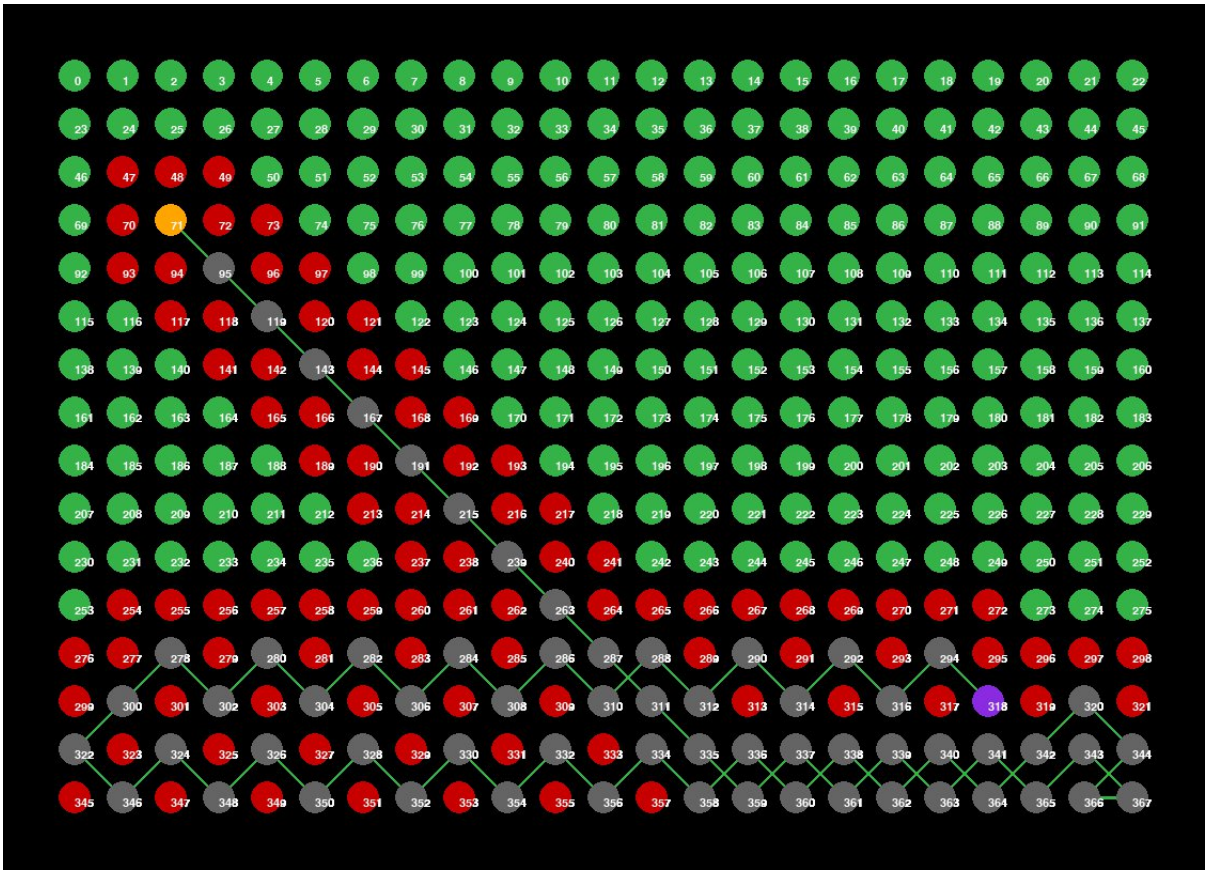
3 Sự khác biệt 2 nhóm thuật toán Informed Search và Uninformed Search

Tiêu chí so sánh	Informed Search (Heuristic Search / Tìm kiếm dựa trên kinh nghiệm)	Uninformed Search (Blind Search / Tìm kiếm mù)
Phương pháp tìm kiếm	Sử dụng kinh nghiệm và thông tin biết trước về vấn đề cần giải quyết để xây dựng nên hàm đánh giá hướng dẫn tìm kiếm	Không yêu cầu sử dụng bất kỳ thông tin gì về các đối tượng để có 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
Tốc độ tìm kiếm	Tìm kiếm được đường đi nhanh hơn	Tìm kiếm được đường đi chậm hơn
Hiệu quả	Hiệu quả hơn khi xét đến chi phí và thời gian. Chi phí phát sinh ít hơn và tốc độ tìm đường đi nhanh hơn	Kém hiệu quả hơn vì chi phí phát sinh nhiều hơn và tốc độ tìm đường đi chậm hơn, điển hình như BFS
Gợi ý/Đề xuất	Dựa trên hàm đánh giá để đề xuất đường đi	Không có bất kỳ đề xuất nào mà chỉ tiến hành mở rộng cho tới khi tìm được đỉnh đích
Thuật toán áp dụng	- Greedy Search - A* Search	- Breadth First Search (BFS) - Depth First Search (DFS) - Uniform Cost Search (UCS)

Chương III

Cài đặt

1 Kết quả của thuật toán Depth First Search

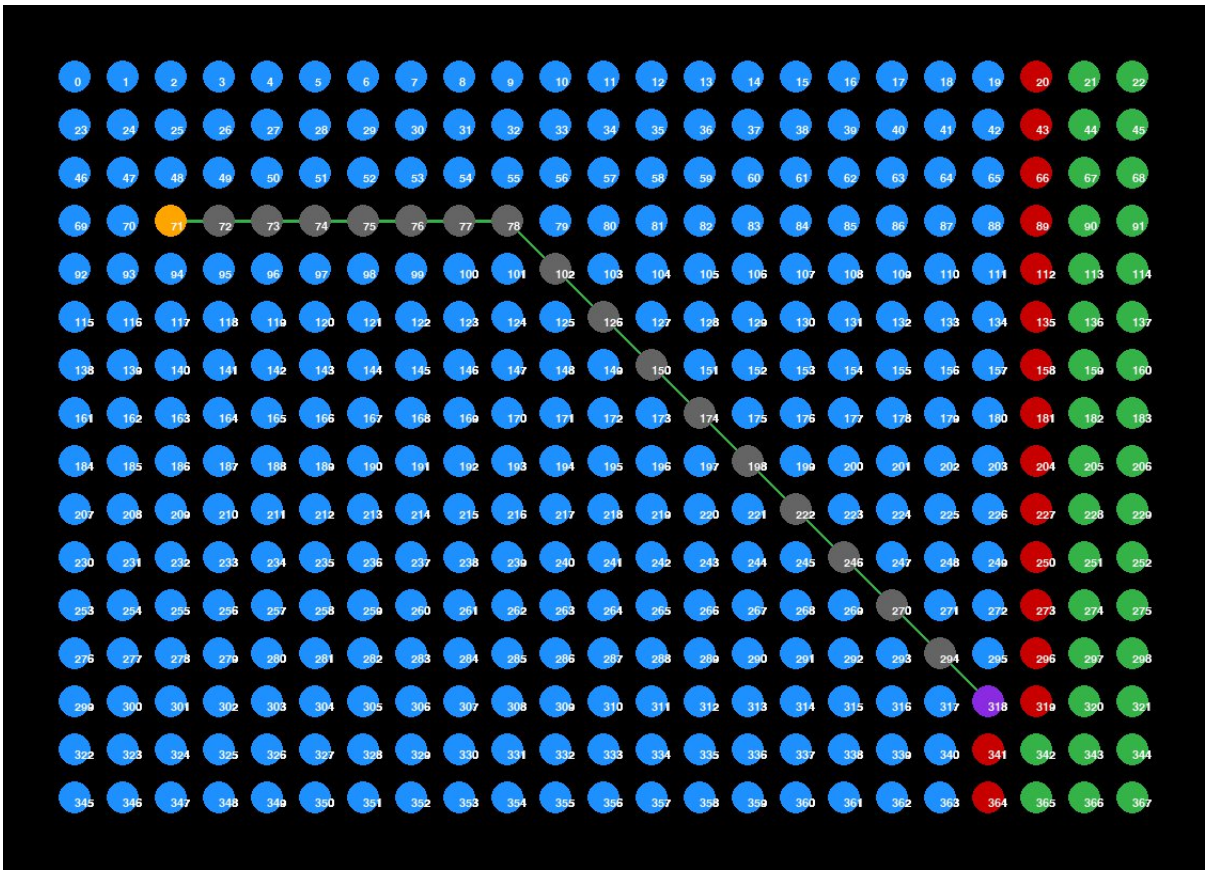


Hình 1: Kết quả tìm kiếm của thuật toán Depth First Search

- **Mô tả đường đi**: Ta dễ dàng nhận thấy rằng DFS sẽ duyệt về một hướng sâu nhất có thể (71-359) như cái tên của nó. Quá trình đó diễn ra bằng cách phát triển các node lân cận theo chiều sâu. Các node được xét theo thứ tự độ sâu giảm dần cho đến khi tìm được đến đích. Đường đi từ node gốc tới node đích sẽ là quá trình mà thuật toán duyệt lần lượt từng node.
- **Nhận xét**:

– Có thể tồn tại nhiều đường đi từ node gốc tới node đích, nhưng DFS luôn trả về kết quả tìm kiếm là đường đi đầu tiên mà nó tìm thấy chứ không quan tâm đến số bước của đường đi đó.

2 Kết quả của thuật toán Breadth First Search



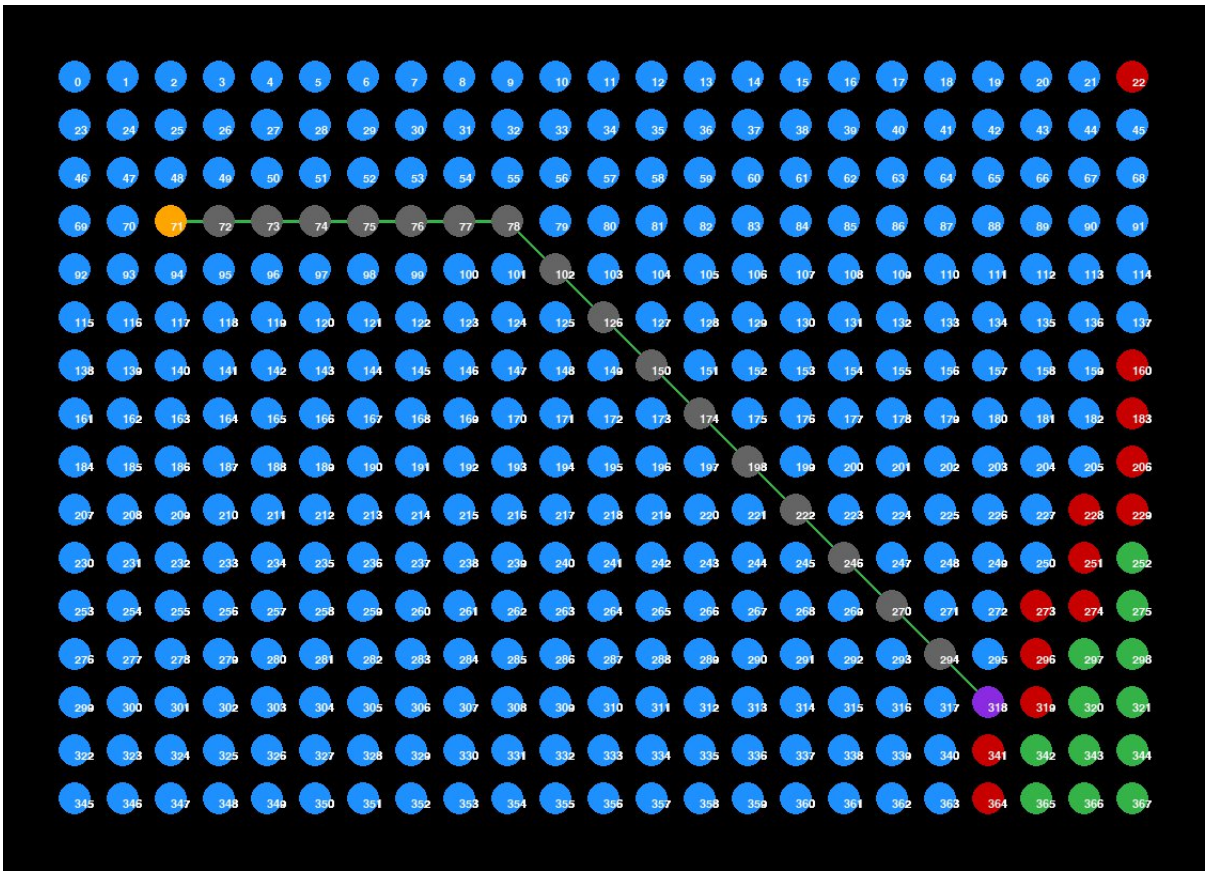
Hình 2: Kết quả tìm kiếm của thuật toán Breadth First Search

- **Mô tả đường đi**: Ta dễ dàng nhận thấy rằng BFS sẽ duyệt các phần tử xung quanh node gốc. Quá trình đó diễn ra bằng cách phát triển các node lân cận theo chiều rộng. Các node được xét theo thứ tự độ sâu tăng dần cho đến khi tìm được đến đích. Đường đi từ node gốc tới node đích sẽ là đường đi có số bước ngắn nhất có thể.

- **Nhận xét**:

  – Có thể tồn tại nhiều đường đi từ node gốc tới node đích, nhưng BFS luôn trả về kết quả tìm kiếm là đường đi có số bước ngắn nhất có thể. Điều đó đồng nghĩa với việc nếu đồ thị có trọng số bằng nhau thì đây cũng là đường đi có chi phí nhỏ nhất.

3 Kết quả của thuật toán Uniform Cost Search



Hình 3: Kết quả tìm kiếm của thuật toán Uniform Cost Search

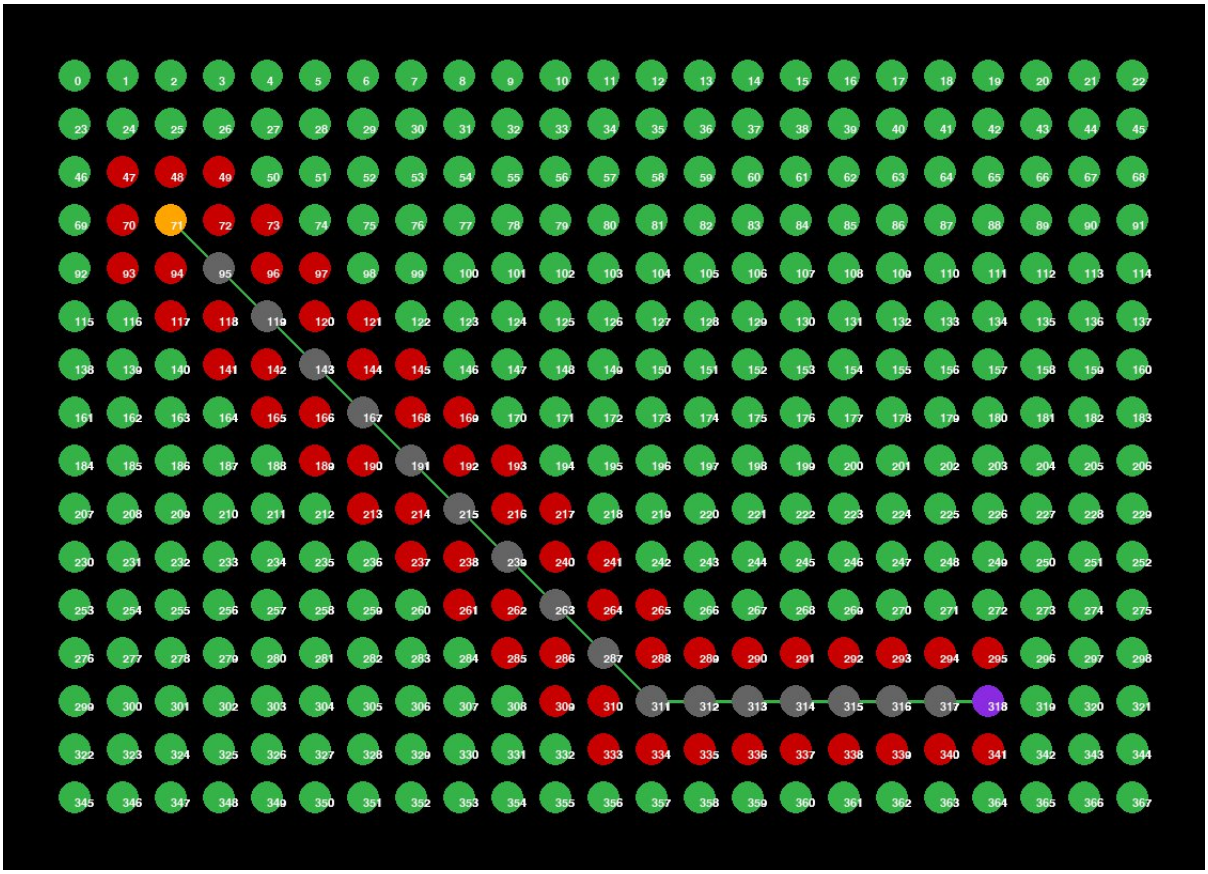
- **Mô tả đường đi**: Ta thấy rằng UCS duyệt các phần tử xung quanh node gốc tương tự như cách BFS thực hiện bằng cách phát triển các node lân cận theo chiều rộng. Tuy nhiên các node được chọn để mở rộng không chỉ xét dựa trên chiều rộng mà còn dựa trên chi phí từ node gốc tới node đó, Các node được xét theo thứ tự độ sâu tăng dần cho đến khi tìm được đến đích. Đường đi từ node gốc tới node đích sẽ là đường đi có số bước ngắn nhất có thể.

- **Nhận xét**:

  – UCS lựa chọn node tiếp theo để duyệt như cách một ngọn lửa lan rộng ra xung quanh. Nên về cơ bản thì nó khá giống như cách duyệt của BFS, khác ở chỗ BFS duyệt các phần tử lân cận theo một thứ tự nào đó còn UCS lại duyệt dựa vào chi phí thấp nhất.

  – Ta cũng có thể dễ dàng nhận thấy UCS trả về một một đường đi có chi phí nhỏ nhất nhưng cũng phải bỏ ra rất nhiều thời gian để duyệt và tính toán nhiều lần.

4 Kết quả của thuật toán Greedy Best First Search

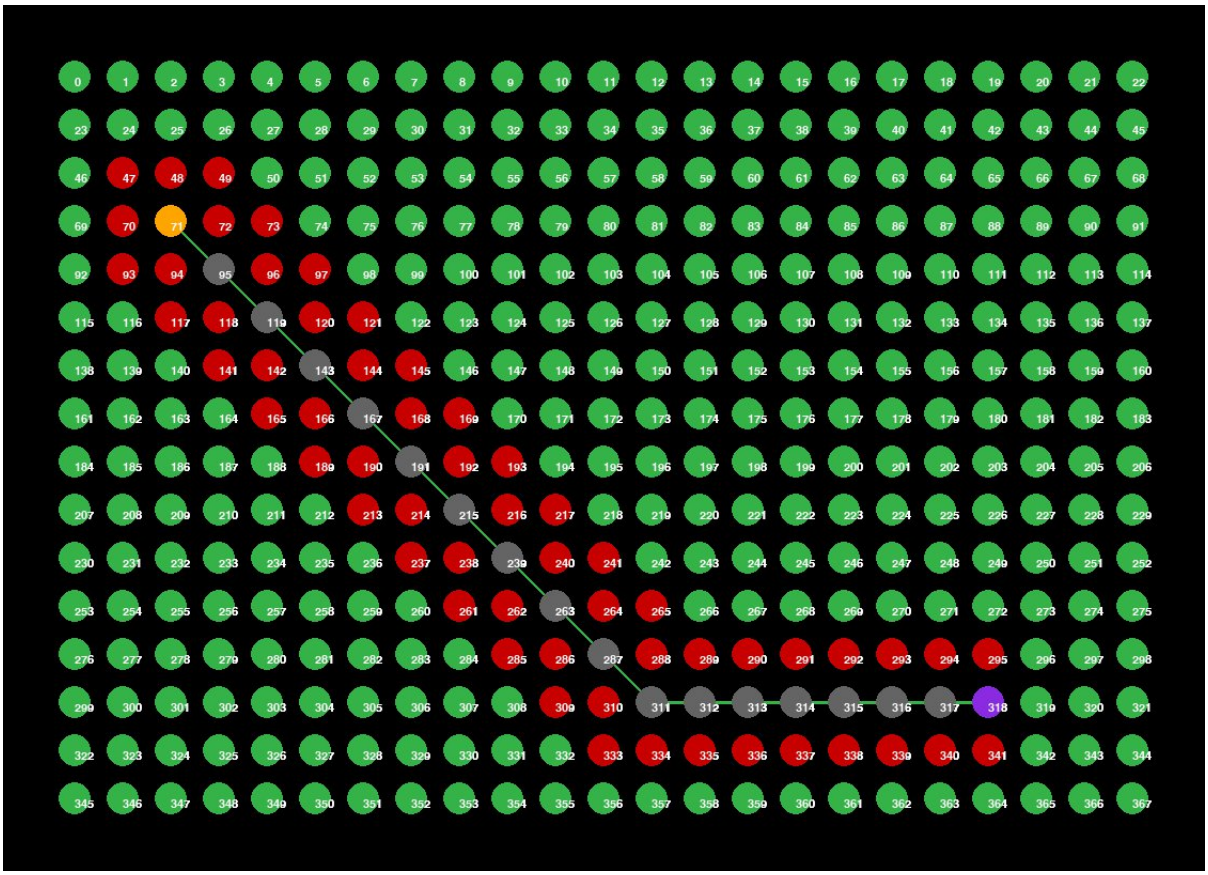


Hình 4: Kết quả tìm kiếm của thuật toán Greedy Best First Search

- **Mô tả đường đi**: Ở đồ thị này thì thuật toán Greedy BFS dễ dàng tính toán chi phí **heuristic** để có thể tìm được đường đi ngắn nhất mà không phải duyệt quá nhiều phần tử khác. Hay nói cách khác là vì ta dễ dàng mở rộng các node về 8 hướng khác nhau nên đi chéo sẽ khiến giá trị hàm $h(x)$ giảm rất nhanh. Sau đó khi đến cùng độ sâu với node đích ta chỉ cần đi ngang là tìm được node đích.

- **Nhận xét**: Giống như cái tên thuật toán, nó **"tham lam"** tính chi phí tới đích ngay từ đầu mà không quan tâm tới chi phí thực tế. Mặc dù ở đây Greedy tìm ra được đường đi tối ưu tức chi phí nhỏ nhất cho đường đi từ node gốc đến node đích. Nhưng đồ thị ta xét ở đây không có những vật cản như trong thực tế hoặc những node có heuristic rất nhỏ nhưng thực tế lại cần đi rất xa mới tới được đích hoặc đôi khi bị kẹt không thể tìm được tới đích.

5 Kết quả của thuật toán A* Search



Hình 5: Kết quả tìm kiếm của thuật toán A* Search

- **Mô tả đường đi**: Là một thuật toán tìm kiếm có kinh nghiệm thì A* đã lựa chọn node tiếp theo để duyệt một cách rất "thông minh". Ban đầu đi theo chiều sâu để $f(x)$ giảm nhanh nhất có thể, khi đến node cùng độ sâu với node đích thì việc đi chéo sẽ có chi phí cao hơn nên A* đã đi **ngang** để tìm đến đích.
- **Nhận xét**: A* tìm được đường đi có chi phí ngắn nhất mà chỉ phải duyệt rất ít phần tử nhờ kết hợp ưu điểm của cả hai thuật toán UCS và Greedy như đã tìm hiểu ở trên.

Lời kết

Trải nghiệm ba tuần thực hiện đồ án: Các thuật toán tìm kiếm trên đồ thị đã giúp chúng em học hỏi được nhiều điều mới mẻ. Từ nguyên lý hoạt động của từng thuật toán tìm kiếm cho đến cách áp dụng chúng vào giải quyết cho từng trường hợp khác nhau, đồng thời chúng em cũng đã có cơ hội rèn luyện thêm những kĩ năng mềm như kĩ năng tự nghiên cứu, tìm kiếm thông tin và làm việc nhóm.

Trong quá trình làm việc, do thời gian thực hiên và chỉnh sửa không nhiều cũng như trình độ của nhóm còn có hạn, nên dù chúng em đã rất cố gắng nhưng khó tránh khỏi sai sót xuất hiện. Vì vậy chúng em rất mong nhận được những góp ý và chia sẻ từ quý thầy cô để nhóm có thể cải thiện bài báo cáo, rút kinh nghiệm và thực hiện các đồ án sau này tốt hơn.

Chúng em xin chân thành cảm ơn!

Ký tên

Nguyễn Nhật Minh Thư
Nguyễn Đặng Anh Thư

Tài liệu tham khảo

[1] Stuart J. Russell, Peter Norvig, *"Artificial Intelligence: A Modern Approach"*, Third Edition. Prentice Hall, 12-2009.

[2] Nilsson, Nils J.,*"Principles of artificial intelligence"*.Tioga Pub. Co, 1980.

[3] Bùi Tiến Lên, *"DSA"*. Truy cập: 25/11/2022 tại `https://drive.google.com/drive/folders/0B3x--lrLA0r2a2dhakVwbE5FMUU?resourcekey=0-k4aLP77yxfPIRHwNtWSwug`.

[4] John DeNero, Dan Klein, Pieter Abbeel and many others, *"CS188/Introduction to Artificial Intelligence"*. Truy cập: 26/11/2022 tại `https://inst.eecs.berkeley.edu/~cs188/fa18/`.

[5] GeeksforGeeks, *"Dijkstra's Shortest Path Algorithm"*. Truy cập: 1/12/2022 tại `https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/`.

[6] GeeksforGeeks, *"Iterative Depth First Search traversal"*. Truy cập: 2/12/2022 tại `https://www.geeksforgeeks.org/iterative-depth-first-traversal/?ref=gcse`.

[7] GeeksforGeeks, *"Breadth First Search or BFS for a Graph"*. Truy cập: 3/12/2022 tại `https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/?ref=gcse`.

[8] Fatima Hasan, *"https://www.educative.io/answers/what-is-uniform-cost-search"*. Truy cập: 3/12/2022 tại `https://www.educative.io/answers/what-is-uniform-cost-search`.

[9] Wikipedia, *"A* search algorithm"*. Truy cập: 3/12/2022 tại `https://en.wikipedia.org/wiki/A*_search_algorithm`.