TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Trần Hoàng Quân

TÌM KIẾM TRONG ĐỒ THỊ

MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

Tp. Hồ Chí Minh, tháng 10/2023

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Trần Hoàng Quân - 21120316

TÌM KIẾM TRONG ĐỒ THỊ

MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

GIÁO VIÊN HƯỚNG DẪN

GS.TS. Lê Hoài Bắc

Thầy Nguyễn Bảo Long

Tp. Hồ Chí Minh, tháng 10/2023

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Thuật toán DFS

1.1 Ý tưởng thuật toán

DFS (Deep First Search) là thuật toán tìm kiếm mù. Ta sẽ cố đi một cách ngẫu nhiên đến đỉnh xa nhất có thể, nếu không thể đi được nữa thì quay lại đỉnh cha để đi theo hướng khác. Điều kiện dừng của thuật toán là khi ta tìm thấy đích hoặc khi ta không còn đường đi nào (không tìm thấy đích).

Khi cài đặc thuật toán ta sẽ sử dụng 2 stack. OpenStack chứa các node ta đang muốn xét. ClosedStack chứa các node ta đã xét. Thuật toán dừng lại khi node đích có trong ClosedStack hoặc khi OpenStack rỗng.

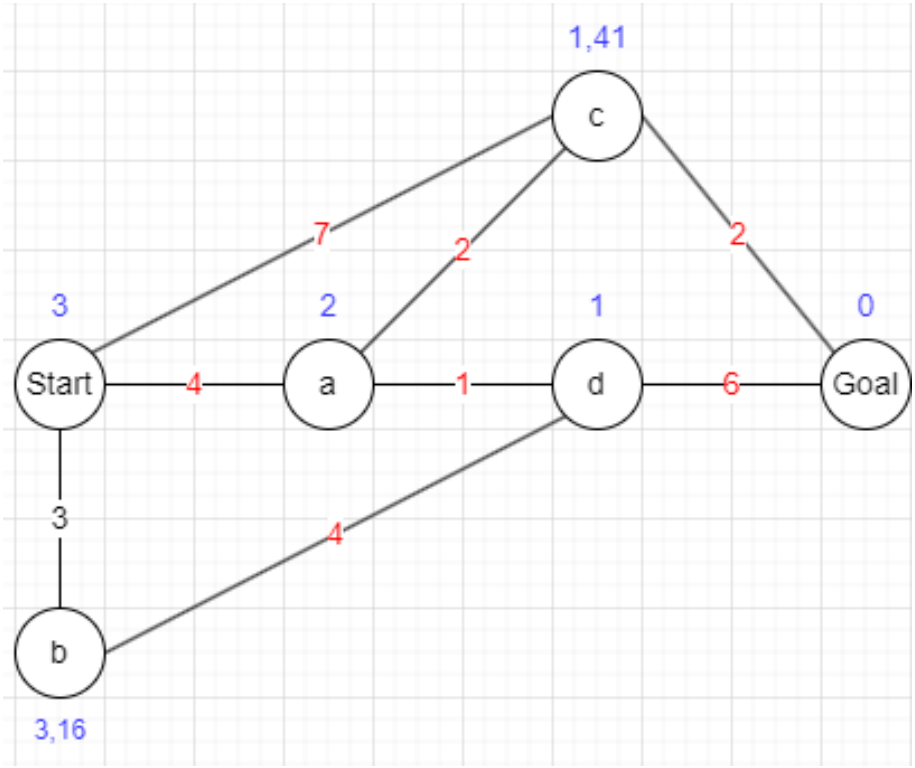
1.2 Mã giả

Algorithm 1 DFS

```
1: procedure DFS
2:     n ← Start
3:     open ← stack([n])
4:     closed ←= []
5: top:
6:     if n == Goal then return True
7:     if open is empty then return False
8:     n = open.pop()
9:     closed.append(n)
10:    neighbors ←GetNeighbor(n)
11:    i ← 0
12: loop1:
13:    if i >= sizeof(neighbors) then Goto top
14:    if neighbors[i] not in closed and neighbors[i] not in open then
       open.append(neighbors[i])
15:    i ← i + 1
16:    Goto loop1
```

1.3 Minh họa thuật toán

Giả sử ta có đồ thị vô hướng như hình, ta cần tìm đường đi từ điểm Start đến Goal.



Hình 1.1: Đồ thị

- Khởi tạo: $n \leftarrow Start$, $open \leftarrow \text{stack}([Start])$, $closed \leftarrow \text{stack}([])$
- Kiểm tra n không phải đích và open không rỗng nên thực hiện:
  - $n = Start$
  - $open = []$
  - $closed = [Start]$
  - $neighbors = [c, a, b]$
- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [c, a, b]$
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  - $n = b$
  - $open = [c, a]$
  - $closed = [Start, b]$
  - $neighbors = [Start, d]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [c, a, d]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $n = d$
  + $open = [c, a]$
  + $closed = [Start, b, d]$
  + $neighbors = [a, b, Goal]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [c, a, Goal]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $n = Goal$
  + $open = [c, a]$
  + $closed = [Start, b, d, Goal]$
  + $neighbors = [d, c]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [c, a]$

- Lặp lại: Kiểm tra thấy n là đích nên dừng thuật toán.

Kết quả: path: $Start \to b \to d \to Goal$; cost = 13

Chương 2

Thuật toán BFS

2.1 Ý tưởng thuật toán

BFS (Breadth First Search) là thuật toán tìm kiếm mù. Ta sẽ lần lượt tìm các đỉnh đi đến từ đỉnh bắt đầu trong đúng 1 bước, trong đúng 2 bước, ..., trong đúng n bước. Điều kiện dừng của thuật toán là khi ta tìm thấy đích hoặc khi ta đã duyệt qua tất cả các đỉnh có thể (không tìm thấy đích).

Khi cài đặt thuật toán ta sẽ sử dụng 2 queue. OpenQueue chứa các node ta đang muốn xét. ClosedQueue chứa các node ta đã xét. Thuật toán dừng lại khi node đích có trong Closedqueue hoặc khi Openqueue rỗng.

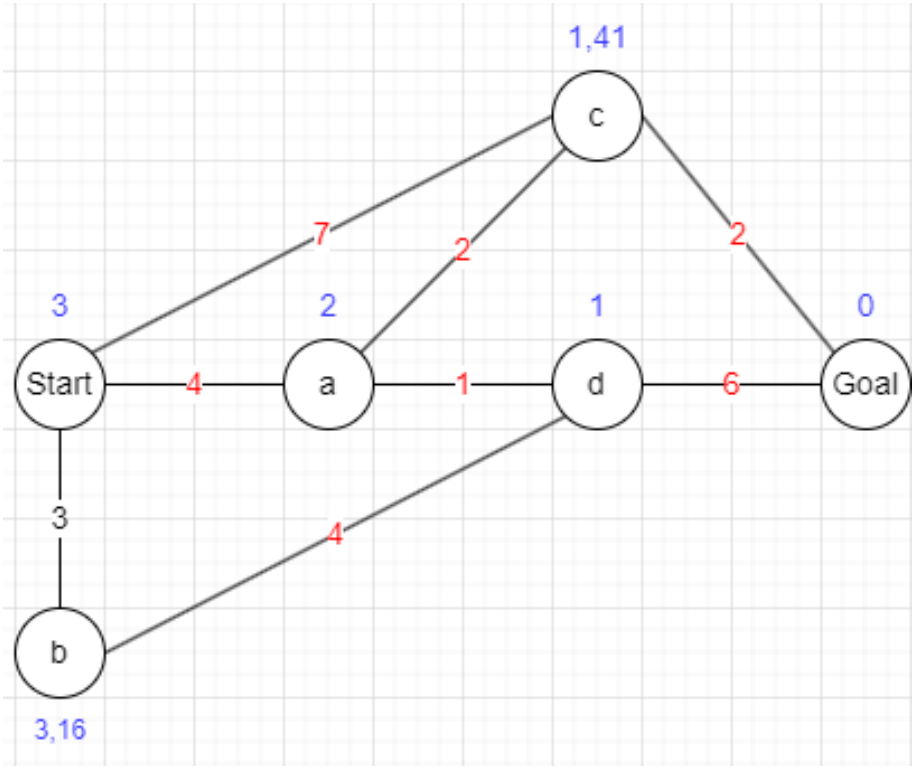
2.2 Mã giả

Algorithm 2 BFS

```
1: procedure BFS
2:     n ← Start
3:     open ← queue([n])
4:     closed ← []
5: top:
6:     if n == Goal then return True
7:     if open is empty then return False
8:     n = open.pop()
9:     closed.append(n)
10:    neighbors ←GetNeighbor(n)
11:    i ← 0
12: loop1:
13:    if i >= sizeof(neighbors) then Goto top
14:    if neighbors[i] not in closed and neighbors[i] not in open then
       open.append(neighbors[i])
15:    i ← i + 1
16:    Goto loop1
```

2.3 Minh họa thuật toán

Giả sử ta có đồ thị vô hướng như hình, ta cần tìm đường đi từ điểm Start đến Goal.



Hình 2.1: Đồ thị

- Khởi tạo: $n \leftarrow Start$, $open \leftarrow$ queue($[Start]$), $closed \leftarrow [\ ]$
- Kiểm tra n không phải đích và open không rỗng nên thực hiện:
  - $n = Start$
  - $open = [\ ]$
  - $closed = [Start]$
  - $neighbors = [c, a, b]$
- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [c, a, b]$
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  - $n = c$
  - $open = [a, b]$
  - $closed = [Start, c]$
  - $neighbors = [Start, a, Goal]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [a, b, Goal]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $n = a$
  + $open = [b, Goal]$
  + $closed = [Start, c, a]$
  + $neighbors = [c, Start, d]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [b, Goal, d]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $n = b$
  + $open = [Goal, d]$
  + $closed = [Start, c, a, b]$
  + $neighbors = [d, Start]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [Goal, d]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $n = Goal$
  + $open = [d]$
  + $closed = [Start, c, a, b, Goal]$
  + $neighbors = [c, d]$

- Duyệt qua các phần tử trong neighbors, nếu không thuộc *open* hay *closed* thì thêm vào open: $open = [d]$

- Lặp lại: Kiểm tra thấy n là đích nên dừng thuật toán.

Kết quả: path: $Start \to c \to Goal$; cost = 9

Chương 3

Thuật toán UCS

3.1 Mô tả thuật toán

UCS (Uniform Cost Search) là thuật toán tìm kiếm mù. Ta sẽ lần lượt tìm các đỉnh có chi phí đường đi thấp nhất khi đi từ điểm bắt đầu. Điều kiện dừng của thuật toán là khi ta tìm thấy đích hoặc khi ta đã duyệt qua tất cả các đỉnh có thể (không tìm thấy đích).

Khi cài đặt ta sẽ sử dụng hàng đợi ưu tiên. OpenSet sẽ là hàng đợi ưu tiên chứa các node ta đang muốn xét. ClosedSet chứa các node ta đã xét. Thuật toán dừng lại khi node đích có trong ClosedSet hoặc khi OpenSet rỗng.

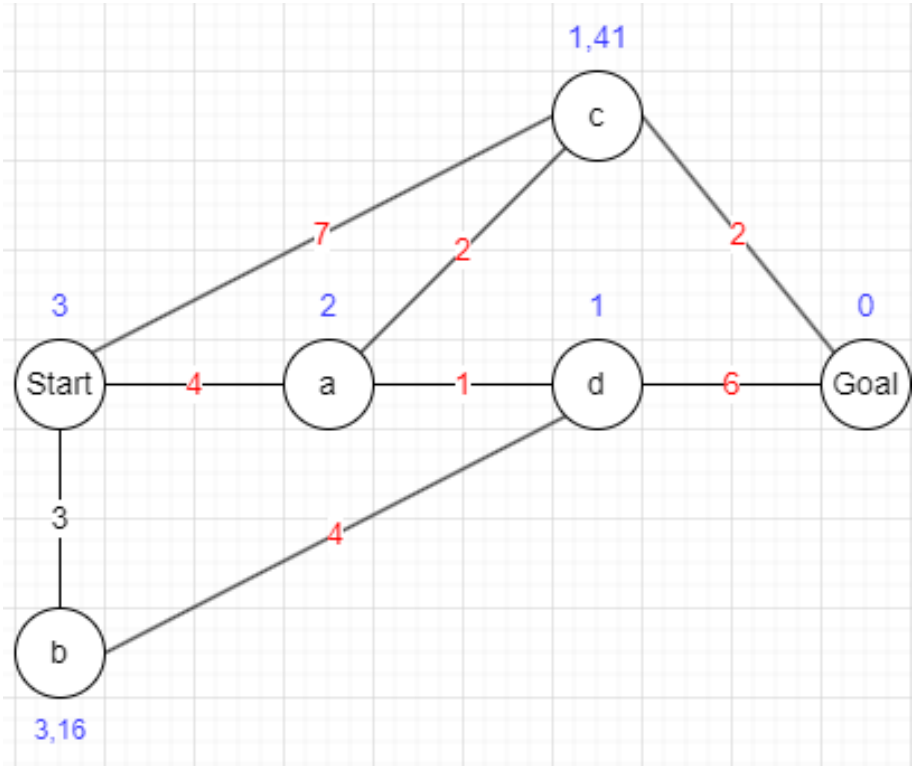
3.2 Mã giả

Algorithm 3 UCS

```
1: procedure UCS
2:     n ← Start
3:     open ← PriorityQueue([(0, n)])
4:     closed ← []
5: top:
6:     if n == Goal then return True
7:     if open is empty then return False
8:     item ← open.pop()
9:     n ← item[1]
10:    closed.append(n)
11:    neighbors ←GetNeighbor(n)
12:    i ← 0
13: loop1:
14:    if i >= sizeof(neighbors) then Goto top
15:    if neighbors[i] not in closed then
16:        if neighbors[i] in open and cost(neighbors[i]) > neighbor_cost
       then open.update((neighbor_cost, neighbors[i]))
17:        if neighbors[i] not in open then open.append((neighbor_cost,
       neighbors[i]))
18:    i ← i + 1
19:    Goto loop1
```

3.3 Minh họa thuật toán

Giả sử ta có đồ thị vô hướng như hình, ta cần tìm đường đi từ điểm Start đến Goal.



Hình 3.1: Đồ thị

- Khởi tạo: $n \leftarrow Start$, $open \leftarrow$ PriorityQueue($[Start]$), $closed \leftarrow [\ ]$
- Kiểm tra n không phải đích và open không rỗng nên thực hiện:
  - $item = (0, Start)$
  - $n = Start$
  - $open = [\ ]$
  - $closed = [Start]$
  - $neighbors = [c, a, b]$
- Duyệt qua các phần tử trong neighbors:
  - Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(3, b), (4, a), (7, c)]$
  - Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

+ $item = (3, b)$
+ $n = b$
+ $open = [(4, a), (7, c)]$
+ $closed = [Start, b]$
+ $neighbors = [Start, d]$

- Duyệt qua các phần tử trong neighbors:
  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(4, a), (7, c), (7, d)]$
  + Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  + $item = (4, a)$
  + $n = a$
  + $open = [(7, c), (7, d)]$
  + $closed = [Start, b, a]$
  + $neighbors = [Start, c, d]$
- Duyệt qua các phần tử trong neighbors:
  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*
  + Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*: $open = [(6, c), (6, d)]$
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  + $item = (6, c)$
  + $n = c$
  + $open = [(6, d)]$

+ $closed = [Start, b, a, c]$

+ $neighbors = [Start, a, Goal]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*

  + Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*: $open = [(6, d), (8, Goal)]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (6, d)$

  + $n = d$

  + $open = [(8, Goal)]$

  + $closed = [Start, b, a, c, d]$

  + $neighbors = [a, b, Goal]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*

  + Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (8, Goal)$

  + $n = Goal$

  + $open = []$

  + $closed = [Start, b, a, c, d, Goal]$

  + $neighbors = [c, d]$

- Duyệt qua các phần tử trong neighbors:

+ Nếu id không có trong *open* hay *closed* thì thêm vào *open*

+ Nếu không thuộc *closed* và có id trong *open* và cost' $>$ cost thì cập nhập *open*

- Lặp lại: Kiểm tra thấy n là đích nên dừng thuật toán

Kết quả: path: $Start \rightarrow a \rightarrow c \rightarrow Goal$; cost = 8

Chương 4

Thuật giải AStar

4.1 Mô tả thuật giải

AStar là thuật giải tìm kiếm đã biết trước đích. Ta sẽ có một hàm f = g + h đại diện cho thứ tự ưu tiên khi duyệt qua các đỉnh. Trong đó g là chi phí đường đi từ điểm bắt đầu đến đỉnh hiện tại, h là hàm Heuristic. Điều kiện dừng của thuật giải là khi ta tìm thấy đường đi tối ưu nhất tới đích hoặc khi ta đã duyệt qua tất cả các đỉnh có thể (không tìm thấy đích).

Khi cài đặt ta sẽ sử dụng hàng đợi ưu tiên. OpenSet sẽ là hàng đợi ưu tiên chứa các node ta đang muốn xét. ClosedSet chứa các node ta đã xét. Thuật toán dừng lại khi node đích có trong ClosedSet hoặc khi OpenSet rỗng.

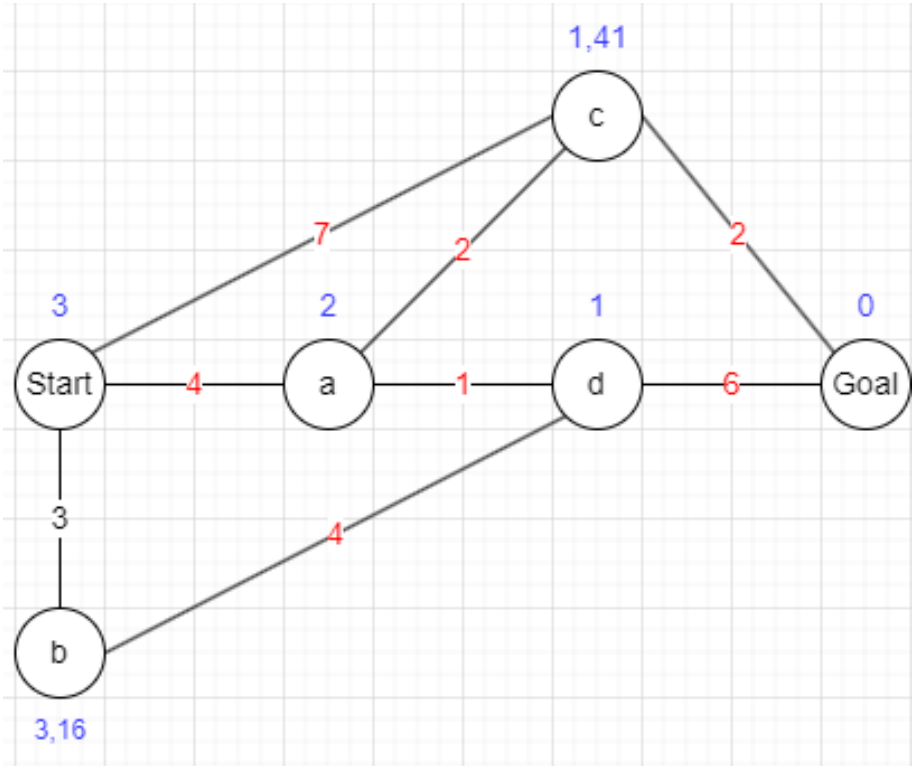
4.2 Mã giả

Algorithm 4 AStar

```
1: procedure HEURISTIC(n: node)
2:     return Euclidean distance between n and Goal
3: procedure ASTAR
4:     n ← Start
5:     open ← PriorityQueue([(0 + Heuristic(n), n)])
6:     closed ← []
7: top:
8:     if n == Goal then return True
9:     if open is empty then return False
10:    item ← open.pop()
11:    n ← item[1]
12:    closed.append(n)
13:    neighbors ←GetNeighbor(n)
14:    i ← 0
15: loop1:
16:    if i >= sizeof(neighbors) then Goto top
17:    f ← neighbor_cost + Heuristic(neighbors[i])
18:    if neighbors[i] not in closed then
19:        if neighbors[i].id in open and f' > f then
20:            open.update((f, neighbors[i]))
21:        if neighbors[i].id not in open then open.append((f,
       neighbors[i]))
22:    i ← i + 1
23:    Goto top
```

4.3 Minh họa thuật toán

Giả sử ta có đồ thị vô hướng như hình, ta cần tìm đường đi từ điểm Start đến Goal.



Hình 4.1: Đồ thị

- Khởi tạo: $n \leftarrow Start$, $open \leftarrow$ PriorityQueue($[Start]$), $closed \leftarrow [\ ]$
- Kiểm tra n không phải đích và open không rỗng nên thực hiện:
  - $item = (3, Start)$
  - $n = Start$
  - $open = [\ ]$
  - $closed = [Start]$
  - $neighbors = [c, a, b]$
- Duyệt qua các phần tử trong neighbors:
  - Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(6, a), (6.16, b), (8.41, c)]$
  - Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

+ $item = (6, a)$

+ $n = a$

+ $open = [(6.16, b), (8.41, c)]$

+ $closed = [Start, a]$

+ $neighbors = [Start, d, c]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(6, d), (6.16, b), (8.41, c)]$

  + Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*: $open = [(6, d), (6.16, b), (7.41, c)]$

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (6, d)$

  + $n = d$

  + $open = [(6.16, b), (7.41, c)]$

  + $closed = [Start, a, d]$

  + $neighbors = [a, b, Goal]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(6.16, b), (7.41, c), (10, Goal)]$

  + Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (6.16, b)$

  + $n = b$

+ $open = [(7.41, c), (10, Goal)]$
+ $closed = [Start, a, d, b]$
+ $neighbors = [Start, d]$

- Duyệt qua các phần tử trong neighbors:
  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*
  + Nếu không thuộc *closed* và có id trong *open* và f' > f thì cập nhập *open*
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  + $item = (7.41, c)$
  + $n = c$
  + $open = [(10, Goal)]$
  + $closed = [Start, a, d, b, c]$
  + $neighbors = [Start, a, Goal]$
- Duyệt qua các phần tử trong neighbors:
  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*
  + Nếu không thuộc *closed* và có id trong *open* và f' > f thì cập nhập *open* $open = [(9, Goal)]$
- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:
  + $item = (9, Goal)$
  + $n = Goal$
  + $open = []$
  + $closed = [Start, a, d, b, c, Goal]$
  + $neighbors = [c, d]$
- Duyệt qua các phần tử trong neighbors:

+ Nếu id không có trong *open* hay *closed* thì thêm vào *open*

+ Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*

- Lặp lại: Kiểm tra thấy n là đích nên dừng thuật toán.

Kết quả: path: $Start \rightarrow a \rightarrow c \rightarrow Goal$; cost = 8

Chương 5

Thuật giải Greedy

5.1 Mô tả thuật giải

Greedy là thuật giải tìm kiếm đã biết trước đích. Ta luôn xem đường đi hiện tại của mình là tốt nhất (theo hàm Heuristic), sau đó chọn hướng đi tốt nhất trong các đỉnh có thể đi. Điều kiện dừng của thuật giải là khi ta tìm thấy đường đi tối ưu nhất tới đích hoặc khi ta đã duyệt qua tất cả các đỉnh có thể (không tìm thấy đích).

Khi cài đặt ta sẽ sử dụng hàng đợi ưu tiên. OpenSet sẽ là hàng đợi ưu tiên chứa các node ta đang muốn xét. ClosedSet chứa các node ta đã xét. Thuật toán dừng lại khi node đích có trong ClosedSet hoặc khi OpenSet rỗng.

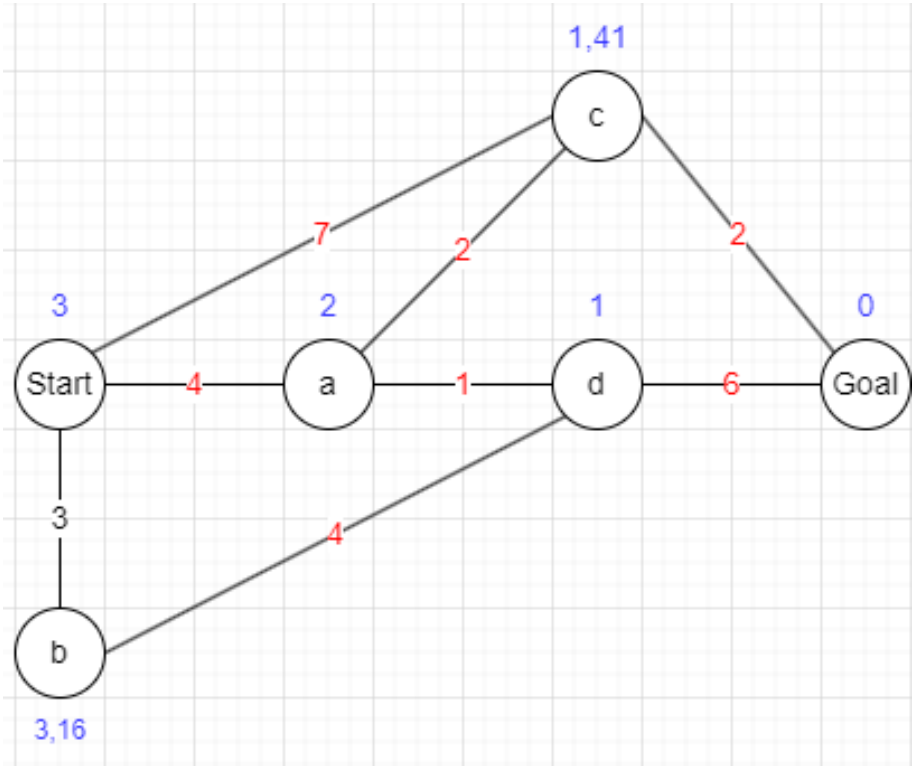
5.2 Mã giả

Algorithm 5 Greedy

```
1: procedure HEURISTIC(n: node)
2:     return Euclidean distance between n and Goal
3: procedure GREEDY
4:     n ← Start
5:     open ← PriorityQueue([(Heuristic(n), n)])
6:     closed ← []
7: top:
8:     if n == Goal then return True
9:     if open is empty then return False
10:    item ← open.pop()
11:    n ← item[1]
12:    closed.append(n)
13:    neighbors ←GetNeighbor(n)
14:    i ← 0
15: loop1:
16:    if i >= sizeof(neighbors) then Goto top
17:    h ←Heuristic(neighbors[i])
18:    if neighbors[i] not in closed then
19:        if neighbors[i].id in open and h' > h then
20:            open.update((h, neighbors[i]))
21:        if neighbors[i].id not in open then open.append((f, neighbors[i]))
22:    i ← i + 1
23:    Goto top
```

5.3 Minh họa thuật toán

Giả sử ta có đồ thị vô hướng như hình, ta cần tìm đường đi từ điểm Start đến Goal.



Hình 5.1: Đồ thị

- Khởi tạo: $n \leftarrow Start$, $open \leftarrow$ PriorityQueue($[Start]$), $closed \leftarrow [\ ]$
- Kiểm tra n không phải đích và open không rỗng nên thực hiện:
  + $item = (3, Start)$
  + $n = Start$
  + $open = [\ ]$
  + $closed = [Start]$
  + $neighbors = [c, a, b]$
- Duyệt qua các phần tử trong neighbors:
  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(1.41, c), (2, a), (3.16, b)]$
  + Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*

;

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (1.41, c)$
  + $n = c$
  + $open = [(2, a), (3.16, b)]$
  + $closed = [Start, c]$
  + $neighbors = [a, Goal]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(0, Goal), (2, a), (3.16, b)]$
  + Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*:

  ;

- Lặp lại: Kiểm tra n không phải là đích và open không rỗng nên thực hiện:

  + $item = (0, Goal)$
  + $n = Goal$
  + $open = [(2, a), (3.16, b)]$
  + $closed = [Start, c, Goal]$
  + $neighbors = [c, d]$

- Duyệt qua các phần tử trong neighbors:

  + Nếu id không có trong *open* hay *closed* thì thêm vào *open*: $open = [(1, d), (2, a), (3.16, b)]$
  + Nếu không thuộc *closed* và có id trong *open* và f' $>$ f thì cập nhập *open*

- Lặp lại: Kiểm tra thấy n là đích nên dừng thuật toán.

Kết quả: path: $Start \to c \to Goal$; cost = 9

5.4 Đánh giá các thuật toán, thuật giải

Các số liệu được tham khảo từ sách [1]

	Khả thi	Tối ưu	Độ phức tạp thời gian	Độ phức tạp không gian
DFS	Không	Không	N/A	N/A
BFS	Có	Có	$O(min(N, B^L)$	$O(min(N, B^L)$
UCS	Có	Có	$O(\log Q * min(N, B^L))$	$O(min(N, B^L)$
AStar	Có	Có		
Greedy	Có	Có		

Bảng 5.1: Đánh giá thuật toán DFS

Tài liệu tham khảo

Tiếng Việt

[1] Lê Hoài Bắc Tô Hoài Việt. *Cơ sở Trí tuệ nhân tạo*. Nhà xuất bản khoa học và kĩ thuật, 2014.