NHÓM5

# Bài toán Giao hàng

Ứng dụng BFS, GBFS, A*

# Nội dung

# Bài toán thực tế

Một công ty cần vận chuyển hàng hoá từ kho đến tay khách hàng trong thời gian ngắn nhất để tối ưu công việc cũng như lợi nhuận.

# Problem

Những vấn đề của bài toán này

## Đường đi

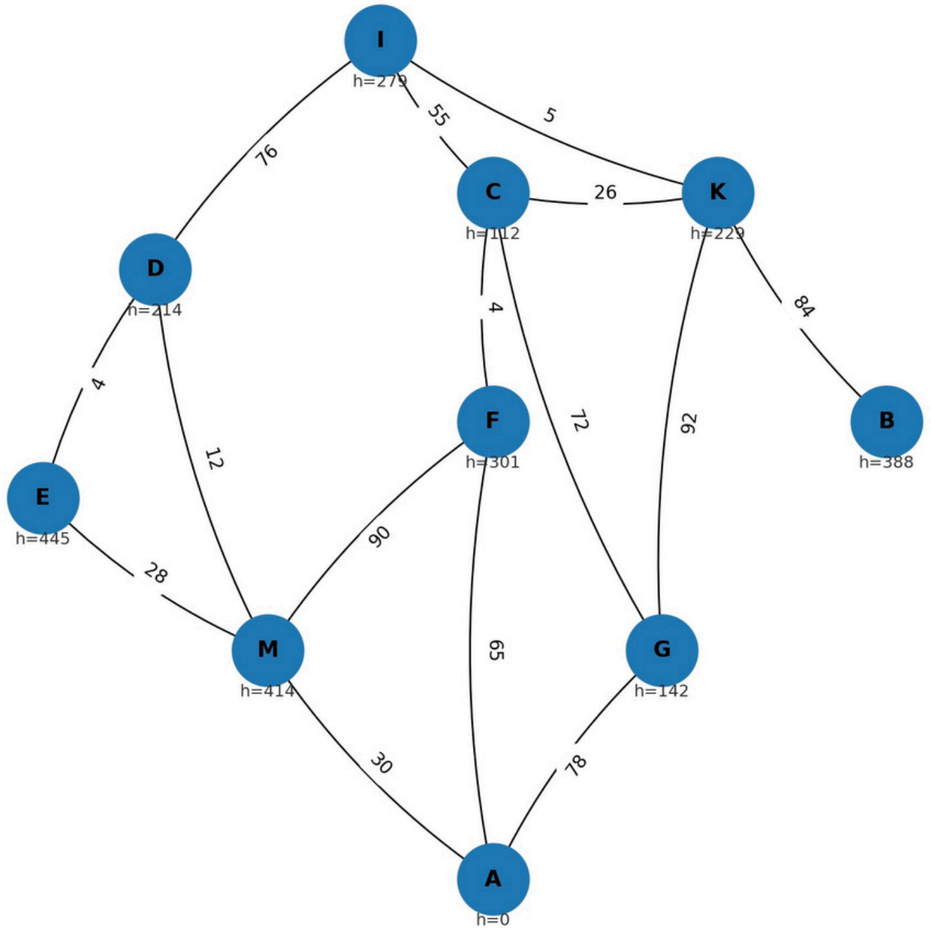
Độ dài của đường đi ảnh hưởng tới thời gian giao hàng tới tay khách hàng.

## Thời gian dừng ở các điểm

Thời gian hàng hoá lưu lại các điểm trước khi được chuyển tiếp cũng là một vấn đề cần được tính toán để tối ưu thời gian, chi phí,...

# Our Solutions

Mô hình hoá bài toán trên thành một bài toán tìm kiếm cụ thể




Kho: I và Khách hàng: A
Các điểm còn lại là các điểm các trạm mà hàng được lưu lại trước khi chuyển tiếp

Đường đi

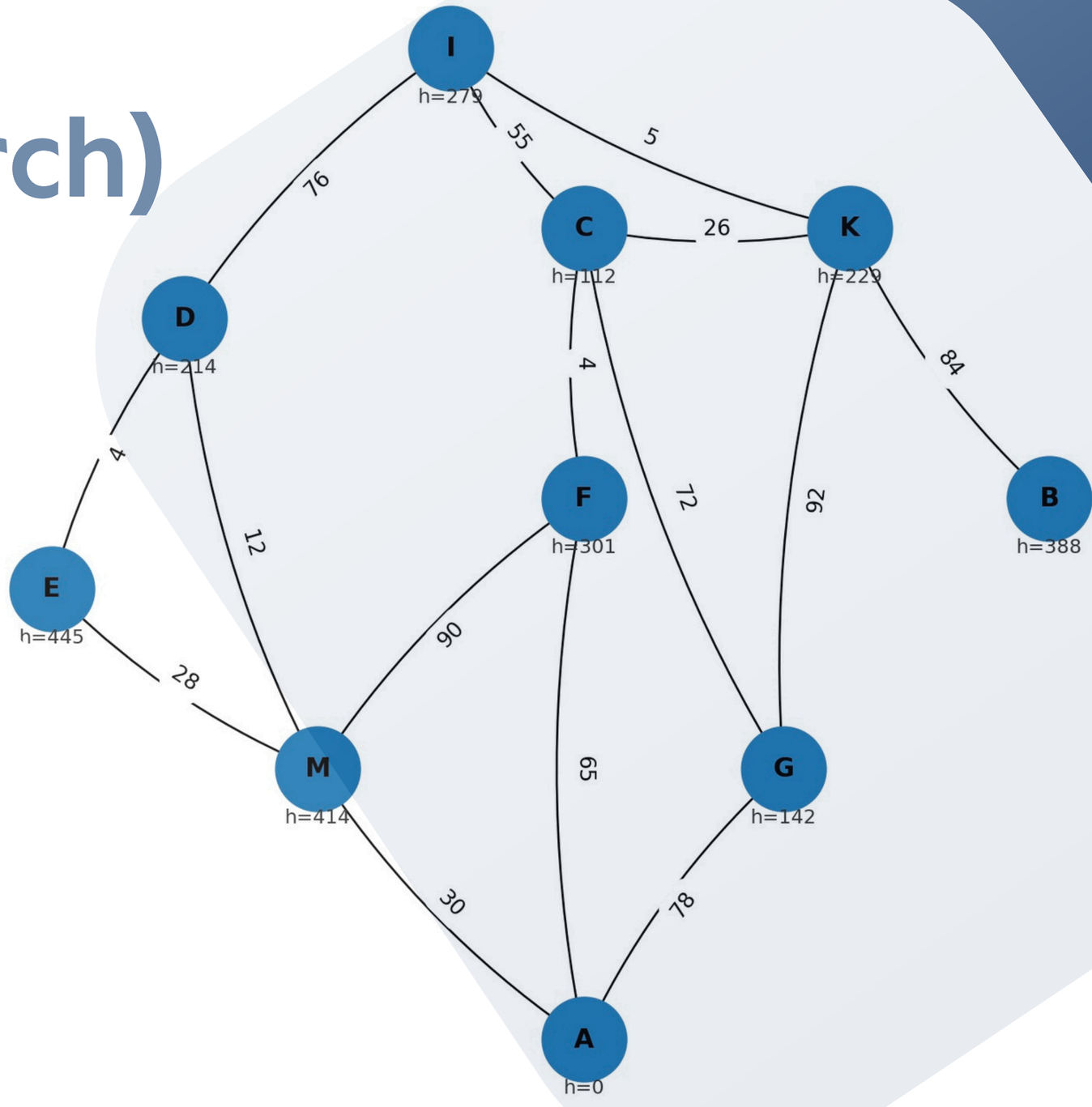
**G(X) LÀ CÁC SỐ TRÊN ĐƯỜNG**

Thời gian dừng ở các điểm

**H(X) LÀ H, CÁC SỐ DƯỚI CÁC ĐIỂM**

# BFS
# (Breadth-First Search)

**Là tìm kiếm theo chiều rộng, là một thuật toán duyệt đồ thị (graph) hoặc cây (tree). Nó bắt đầu từ một nút gốc (root) và khám phá tất cả các nút lân cận ở mức hiện tại trước khi chuyển sang các nút ở mức tiếp theo.**



# Table BFS

| Bước | Queue | Đỉnh đang xét | Các đỉnh kề chưa thăm | Cập nhật Queue | Parent |
|---|---|---|---|---|---|
| 1 | [I] | I | D, C, K | [D, C, K] | D←I, C←I, K←I |
| 2 | [D, C, K] | D | E, M | [C, K, E, M] | E←D, M←D |
| 3 | [C, K, E, M] | C | F, G | [K, E, M, F, G] | F←C, G←C |
| 4 | [K, E, M, F, G] | K | B | [E, M, F, G, B] | B←K |
| 5 | [E, M, F, G, B] | E | (D, M đã thăm) | [M, F, G, B] | — |
| 6 | [M, F, G, B] | M | A (D, E, F đã thăm) | [F, G, B, A] | A←M |
| 7 | [F, G, B, A] | **A** | — (đạt đích) | — | — |

# Coding

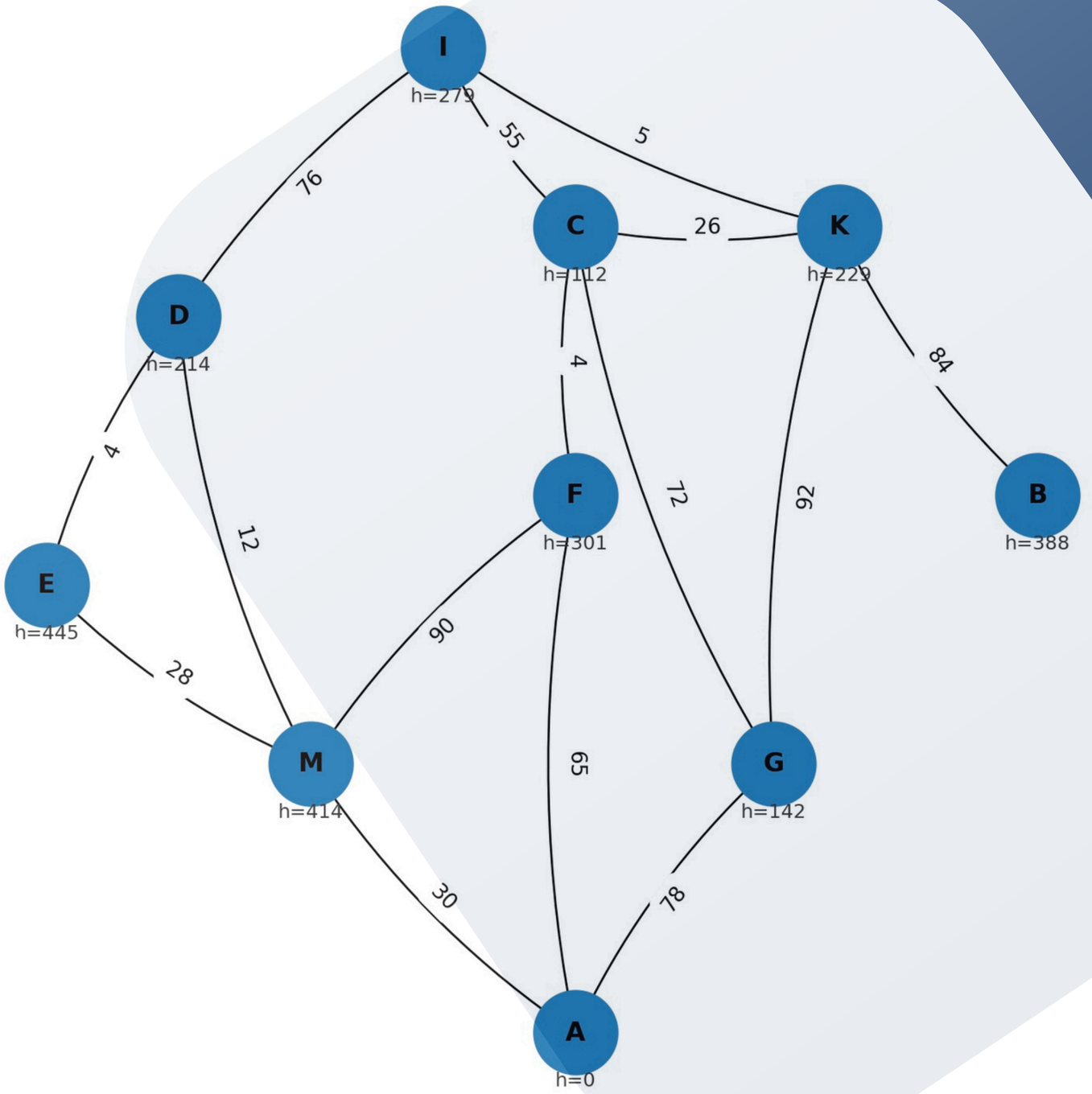
```python
from collections import deque
import time

graph = {
    'A': [('M', 30), ('G', 78)],
    'M': [('A', 30), ('E', 28), ('F', 90)],
    'G': [('A', 78), ('C', 72), ('K', 92)],
    'F': [('M', 90), ('C', 4), ('A', 65)],
    'E': [('M', 28), ('D', 4)],
    'D': [('E', 4), ('I', 76), ('M', 12)],
    'C': [('F', 4), ('I', 55), ('K', 26), ('G', 72)],
    'I': [('D', 76), ('C', 55), ('K', 5)],
    'K': [('I', 5), ('C', 26), ('B', 84)],
    'B': [('K', 84)]
}

def bfs(graph, start, goal):
    if start not in graph or goal not in graph:
        return None, 0, 0.0

    queue = deque([start])
    visited = set([start])
    parent = {start: None}
    expanded = 0
    start_time = time.time()

    while queue:
        current = queue.popleft()
        expanded += 1

        if current == goal:
            path = []
            while current:
                path.append(current)
                current = parent[current]
            path.reverse()
            end_time = time.time()
            runtime = end_time - start_time
            return path, expanded, runtime

        for neighbor, _ in graph[current]:
            if neighbor not in visited:
                visited.add(neighbor)
                parent[neighbor] = current
                queue.append(neighbor)

    end_time = time.time()
    runtime = end_time - start_time
    return None, expanded, runtime

def calculate_cost(graph, path):
    """
    Tính tổng chi phí (trọng số) trên đường đi.
    """
    if not path:
        return float('inf')

    cost = 0
    for i in range(len(path) - 1):
        for neigh, weight in graph[path[i]]:
            if neigh == path[i + 1]:
                cost += weight
                break
    return cost

start_node = 'I'
goal_node = 'A'
path, expanded, runtime = bfs(graph, start_node, goal_node)

if path:
    print(f"Đường đi (BFS unweighted): {' → '.join(path)}")
    total_cost = calculate_cost(graph, path)
    print(f"Tổng chi phí: {total_cost}")
    print(f"Số node mở rộng: {expanded}")
    print(f"Thời gian chạy: {runtime:.6f} giây")
else:
    print("Không tìm thấy đường đi!")
```

# Kết quả

```
Đường đi (BFS unweighted): I → D → M → A
Tổng chi phí: 118
Số node mở rộng: 10
Thời gian chạy: 0.000037 giây
```

# Greedy Best-First Search

**Thuật toán này sẽ chỉ xét h(x), xem thời gian hàng hoá được lưu lại ở đâu ngắn nhất hay h nhỏ nhất thì chúng ta chọn**



# Table GBFS

| Step | u (đang xét) | Edge (node kề với u) | Open (sau khi thêm neighbors, theo h tăng dần) |
|---|---|---|---|
| 1 | **I** | D (76), C (55), K (5) | {C (112), K (229), D (214)} |
| 2 | **C** (h=112) | F (4), I (55), K (26), G (72) | {G (142), K (229), D (214), F (301)} |
| 3 | **G** (h=142) | A (78), C (72), K (92) | {A (0), K (229), D (214), F (301)} |
| 4 | **A** (h=0)** ✅ ** | M (30), G (78) | → **Đã tới đích (A)** |

# Coding

```python
import heapq
import time
start_time = time.time()

graph = {
    'A': [('M', 30), ('G', 78)],
    'M': [('A', 30), ('E', 28), ('F', 90)],
    'G': [('A', 78), ('C', 72), ('K', 92)],
    'F': [('M', 90), ('C', 4), ('A', 65)],
    'E': [('M', 28), ('D', 4)],
    'D': [('E', 4), ('I', 76), ('M', 12)],
    'C': [('F', 4), ('I', 55), ('K', 26), ('G', 72)],
    'I': [('D', 76), ('C', 55), ('K', 5)],
    'K': [('I', 5), ('C', 26), ('B', 84)],
    'B': [('K', 84)]
}

h = {
    'A': 0,
    'G': 142,
    'M': 414,
    'E': 445,
    'F': 301,
    'D': 214,
    'C': 112,
    'I': 279,
    'K': 229,
    'B': 388
}
```

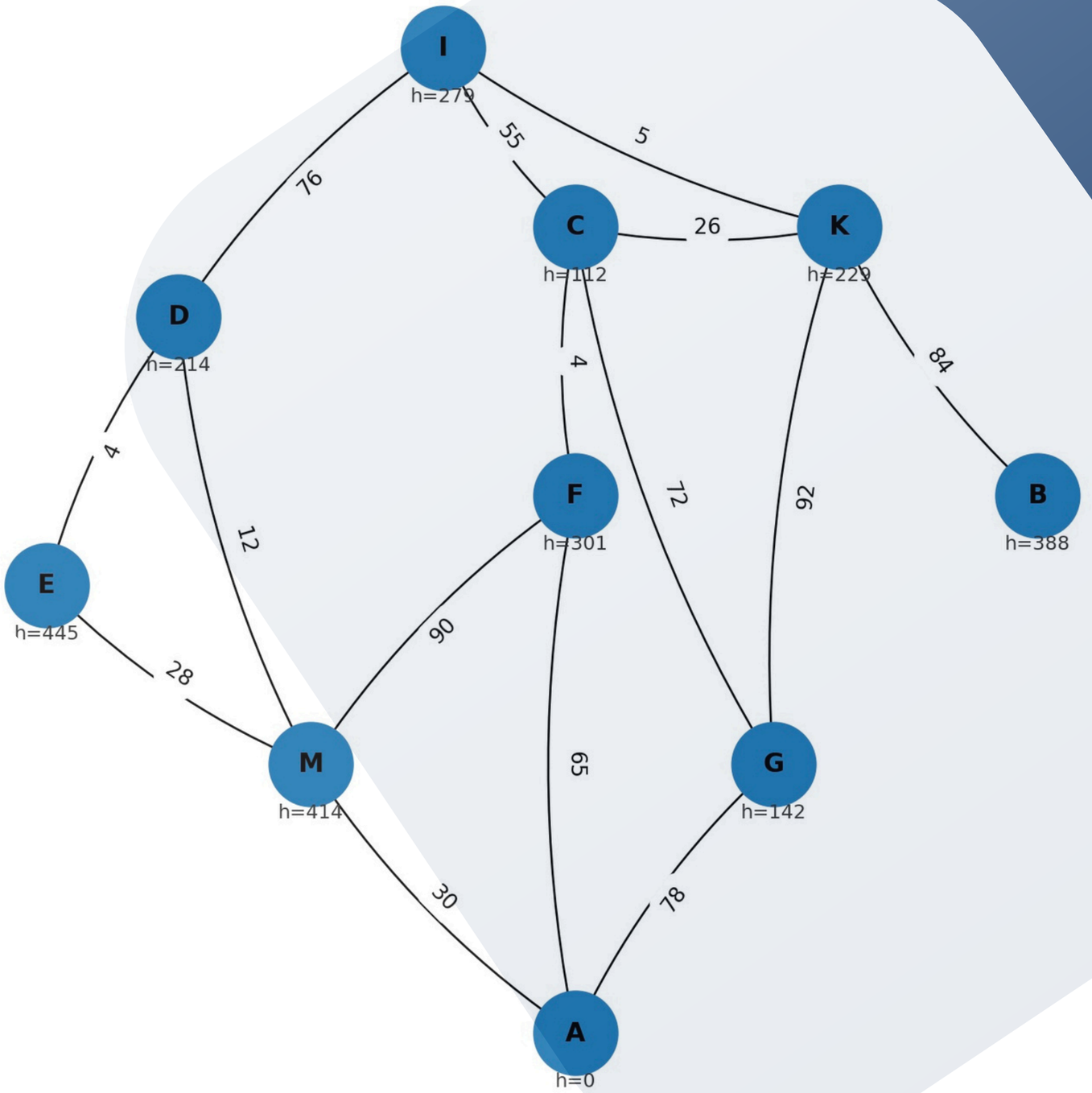
```python
def gbfs(start, goal):
    priority_queue = []
    heapq.heappush(priority_queue, (h[start], start, []))
    visited = set()
    expanded_nodes = set()
    steps = []
    expanded_count = 0

    while priority_queue:
        current_h, current, path = heapq.heappop(priority_queue)
        path = path + [current]
        steps.append(f"Popped: {current} (h={current_h}), Path so far: {path}")
        expanded_count += 1

        if current in visited:
            continue
        visited.add(current)
        expanded_nodes.add(current)

        if current == goal:
            steps.append(f"Goal reached: {path}")
            steps.append(f"Nodes expanded: {expanded_count}")
            sum_h = sum(h[node] for node in expanded_nodes)
            steps.append(f"Tổng h của các node đã expand: {sum_h}")
            return '\n'.join(steps), path

        for neighbor, cost in graph.get(current, []):
            if neighbor not in visited:
                heapq.heappush(priority_queue, (h[neighbor], neighbor, path))
                steps.append(f"  Added neighbor: {neighbor} (h={h[neighbor]})")

    steps.append("No path found")
    return '\n'.join(steps), None


result, path = gbfs('I', 'A')
print(result)

end_time = time.time()
execution_time = end_time - start_time
print(f"Thời gian chạy: {execution_time} giây")
```

# Kết quả

```
Popped: I (h=279), Path so far: ['I']
  Added neighbor: D (h=214)
  Added neighbor: C (h=112)
  Added neighbor: K (h=229)
Popped: C (h=112), Path so far: ['I', 'C']
  Added neighbor: F (h=301)
  Added neighbor: K (h=229)
  Added neighbor: G (h=142)
Popped: G (h=142), Path so far: ['I', 'C', 'G']
  Added neighbor: A (h=0)
  Added neighbor: K (h=229)
Popped: A (h=0), Path so far: ['I', 'C', 'G', 'A']
Goal reached: ['I', 'C', 'G', 'A']
Nodes expanded: 4
Tổng h của các node đã expand: 533
Thời gian chạy: 0.00100874900817871 giây
```

# A* Algorithm

**Thuật toán A* được sử dụng để tìm đường đi ngắn nhất từ nút bắt đầu đến nút đích trong một đồ thị có trọng số dựa trên hàm đánh giá f(n) = g(n) + h(n)**



| Bước | Nút đang xét | Láng giềng mở rộng | g(n) hiện tại | h(n) | f(n)=g+h | Open list sau bước này | Closed list |
|---|---|---|---|---|---|---|---|
| 1 | **I** | D(76), C(55), K(5) | 0 | 279 | 279 | {D(290), C(167), K(234)} | {I} |
| 2 | **C** (nhỏ nhất f=167) | F(4), I(55), K(26), G(72) | 55 | 112 | 167 | {D(290), K(234), F(359), G(239)} | {I, C} |
| 3 | **K** (f=234) | I(5), C(26), B(84) | 81 | 229 | 310 | {D(290), G(239), F(359), B(469)} | {I, C, K} |
| 4 | **G** (f=239) | A(78), C(72), K(92) | 127 | 142 | 269 | {D(290), F(359), A(205), B(469)} | {I, C, K, G} |
| 5 | **A** (f=205 – nhỏ nhất, là đích)** | — | 205 | 0 | 205 | — | {I, C, K, G, A} |

# Coding

```python
import heapq

graph = {
    'A': [('M', 30), ('G', 78)],
    'M': [('A', 30), ('E', 28), ('F', 90)],
    'G': [('A', 78), ('C', 72), ('K', 92)],
    'F': [('M', 90), ('C', 4), ('A', 65)],
    'E': [('M', 28), ('D', 4)],
    'D': [('E', 4), ('I', 76), ('M', 12)],
    'C': [('F', 4), ('I', 55), ('K', 26), ('G', 72)],
    'I': [('D', 76), ('C', 55), ('K', 5)],
    'K': [('I', 5), ('C', 26), ('B', 84)],
    'B': [('K', 84)]
}

h = {
    'A': 0,
    'G': 142,
    'M': 414,
    'E': 445,
    'F': 301,
    'D': 214,
    'C': 112,
    'I': 279,
    'K': 229,
    'B': 388
}
```

```python
start = 'I'
goal = 'A'

def a_star(graph, h, start, goal):
    open_heap = []
    counter = 0
    g = {start: 0}
    f = {start: g[start] + h.get(start, float('inf'))}
    came_from = {}
    heapq.heappush(open_heap, (f[start], counter, start))
    closed_set = set()
    trace = []
    step = 0

    while open_heap:
        _, _, current = heapq.heappop(open_heap)
        if current in closed_set:
            continue
        step += 1
        trace.append((step, current, g[current], h[current], g[current] + h[current]))
        if current == goal:
            # reconstruct path
            path = []
            node = goal
            while node in came_from:
                path.append(node)
                node = came_from[node]
            path.append(start)
            path.reverse()
            return {'path': path, 'cost': g[goal], 'trace': trace}
```

# Kết quả

'path': ['I', 'C', 'G', 'A']

'cost': 205

'trace': (1, 'I', 0, 279, 279)
(2, 'C', 55, 112, 167)
(3, 'K', 5, 229, 234)
(4, 'G', 127, 142, 269)
(5, 'A', 205, 0, 205)

# So sánh

| Tiêu chí | BFS | GBFS | A* |
|---|---|---|---|
| Xét trọng số | Không | Không (chỉ h) | Có (g + h) |
| Đảm bảo tối ưu | Có (nếu không có trọng số) | Không | Có |
| Tốc độ | Khá nhanh so với đồ thị nhỏ nhưng chậm hơn với đồ thị lớn | Nhanh nhất | Trung bình |
| Node mở rộng | Nhiều | Ít nhất | Trung bình |
| Thích hợp khi | Chi phí bằng nhau | Cần tốc độ | Cần kết quả tối ưu |

# Ý nghĩa

- Ba thuật toán BFS (Breadth-First Search), GBFS (Greedy Best-First Search) và A* đều thuộc nhóm các thuật toán tìm kiếm trên đồ thị, được sử dụng rộng rãi trong trí tuệ nhân tạo và lý thuyết đồ thị để tìm đường đi hoặc lời giải tối ưu. Có thể xem BFS là nền tảng cơ bản, GBFS là hướng tối ưu hóa theo heuristic, và A* là sự dung hòa thông minh giữa hai phương pháp này.
- Về ý nghĩa thực tế, ba thuật toán này đều đóng vai trò quan trọng trong các hệ thống ra quyết định và tìm đường trong thế giới thực.
- Tổng thể, BFS, GBFS và A* không chỉ là những thuật toán lý thuyết trong lĩnh vực trí tuệ nhân tạo mà còn thể hiện ba cấp độ tư duy: từ việc duyệt toàn diện và chắc chắn (BFS), sang tìm kiếm nhanh theo hướng hứa hẹn (GBFS), đến tìm kiếm có định hướng và tối ưu hóa thông minh (A*). Chính sự tiến hóa đó đã tạo nền móng cho các hệ thống AI và robot hiện đại ngày nay.


Thank
you!