

# LECTURE 08

## DIJKSTRA'S ALGORITHM

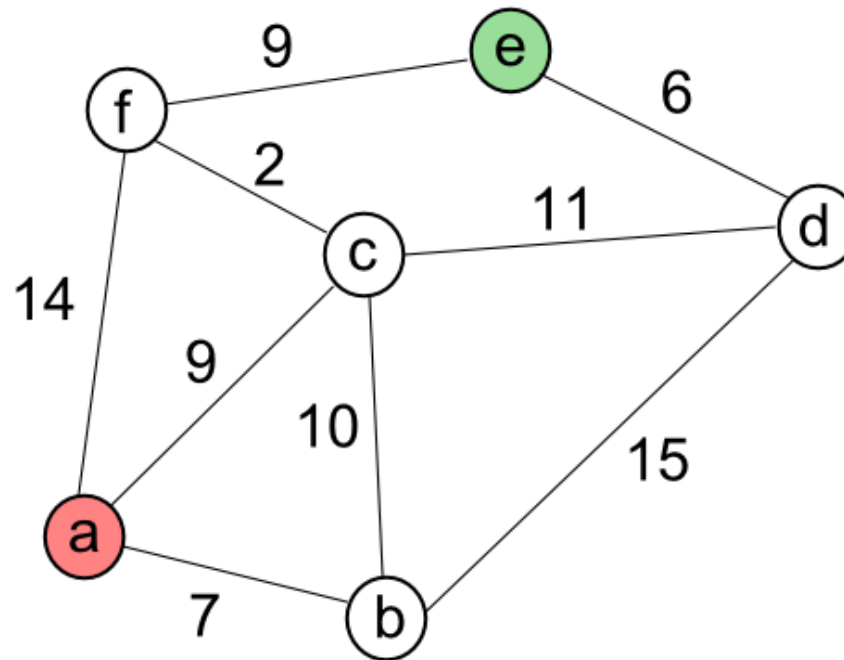


Phạm Nguyễn Sơn Tùng

Email: [sontungtn@gmail.com](mailto:sontungtn@gmail.com)

# Thuật toán Dijkstra

Thuật toán **Dijkstra** là thuật toán tìm đường đi có chi phí nhỏ nhất từ **một đỉnh** đến **tất cả các đỉnh** còn lại trong đồ thị có **trọng số** (trọng số **không âm**).



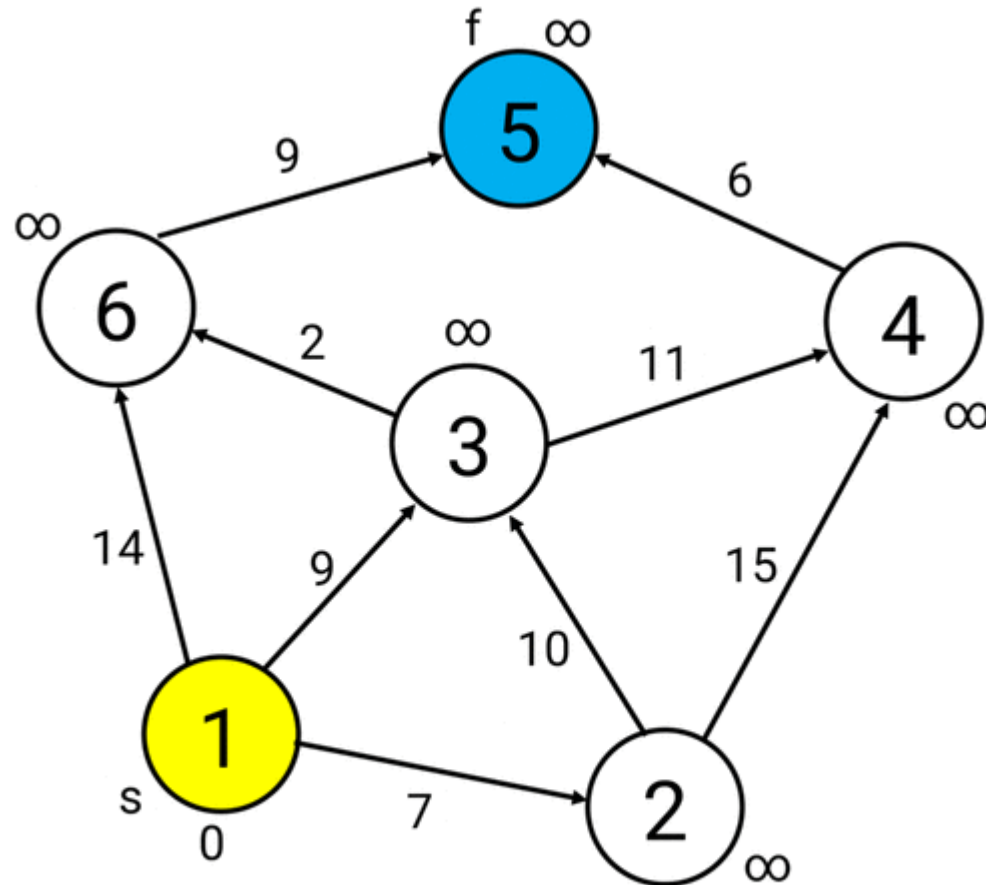
# Độ phức tạp thuật toán Dijkstra

Thuật toán **Dijkstra** bình thường sẽ có độ phức tạp là  $O(V^2)$ .

Tuy nhiên ta có thể sử dụng kết hợp các cấu trúc dữ liệu khác để giảm độ phức tạp:

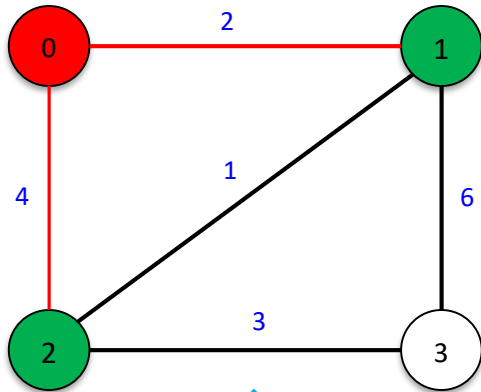
- Dùng Heap (priority queue) có độ phức tạp:  $O(E \log V)$ .
- Dùng kết hợp với cấu trúc dữ liệu cây nhị phân tìm kiếm độ phức tạp là:  $O(E \log V)$ .

# Mô phỏng cách chạy thuật toán



# Ý tưởng của thuật toán

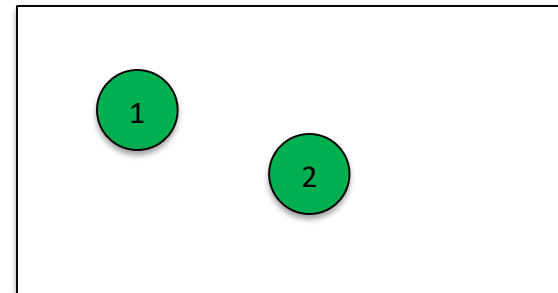
Xuất phát từ một đỉnh bất kỳ. Đi đến các đỉnh kề của đỉnh này.



So sánh chi phí đường đi đến đỉnh kề đang xét với đường đi hiện tại.

Đỉnh	0	1	2	3
Chi phí	0	$\infty$	$\infty$	$\infty$

(nếu nhỏ hơn)  
thêm vào kho.

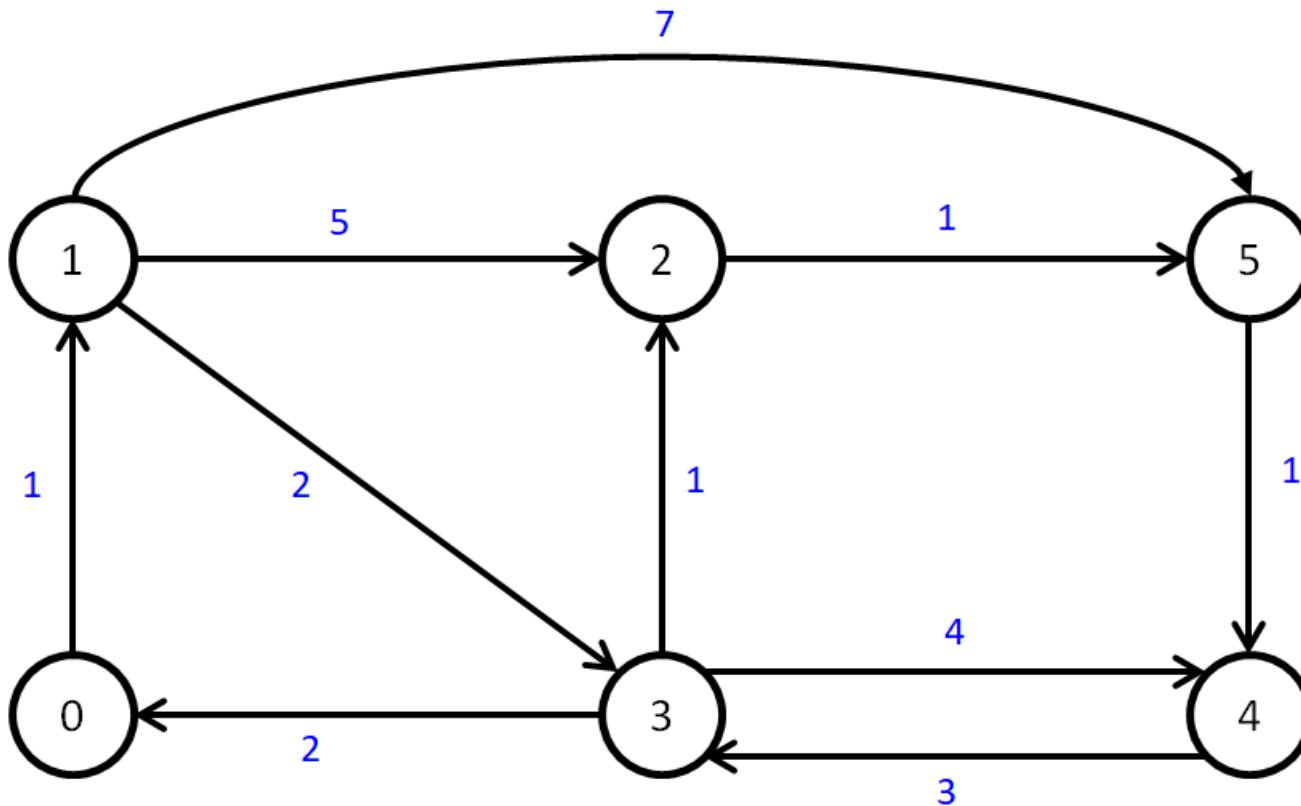


Lưu vết và đem đỉnh mới ra xét.

→ Dừng khi kho không còn đỉnh nào. Xuất kết quả bài toán.

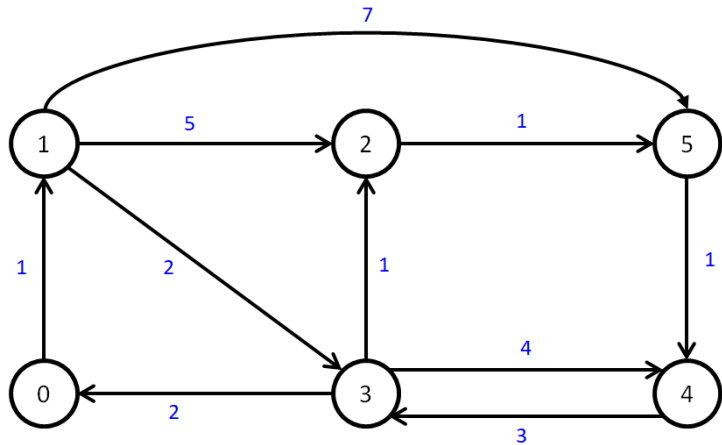
# Bài toán minh họa

Cho đồ thị có hướng, có trọng số như hình vẽ. Tìm **đường đi ngắn nhất (chi phí thấp nhất)** từ đỉnh 0 đến tất cả các đỉnh còn lại.



# Bước 0: Chuẩn bị dữ liệu (1)

Dữ liệu đầu vào là ma trận kề, danh sách cạnh kề hoặc định dạng dữ liệu khác.



*Adjacency Matrix*

6					
0	1	0	0	0	0
0	0	5	2	0	7
0	0	0	0	0	1
2	0	1	0	4	0
0	0	0	3	0	0
0	0	0	0	1	0

*Edge List*

```

6 10
0 1 1
1 2 5
1 3 2
1 5 7
2 5 1
3 0 2
3 2 1
3 4 4
4 3 3
5 4 1

```

# Bước 0: Chuẩn bị dữ liệu (2)

Chuyển ma trận kề / danh sách cạnh kề vào **graph**.

Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	(3, 3)	(4, 1)

Mảng chứa chi phí đường đi **dist**.

Đỉnh	0	1	2	3	4	5
Chi phí	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

Mảng lưu vết đường đi **path**.

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	-1	-1	-1	-1	-1

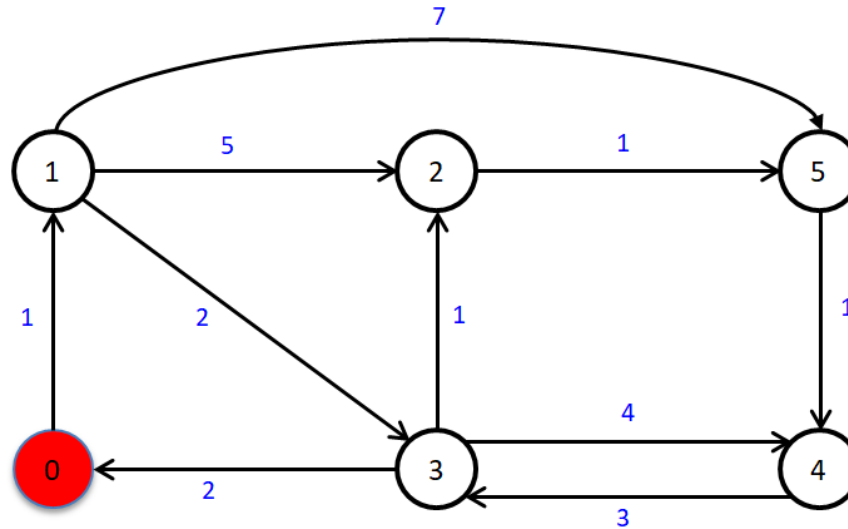
Hàng đợi ưu tiên **priority queue**, lưu cặp giá trị (**đỉnh, chi phí**).

...

(x, y)



# Bước 0: Chuẩn bị dữ liệu (3)



**dist** - lấy đỉnh bắt đầu đi là **đỉnh 0**. Gán chi phí cho đỉnh 0 là 0.

Đỉnh	0	1	2	3	4	5
Chi phí	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

**priority queue** - bỏ cặp (0, 0) vào hàng đợi ưu tiên.

0

(0, 0)

# Bước 1: Chạy thuật toán lần 1

0

(0, 0)

priority queue



Lấy cặp **đỉnh 0, chi phí 0** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 0.

graph

Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	(3, 3)	(4, 1)

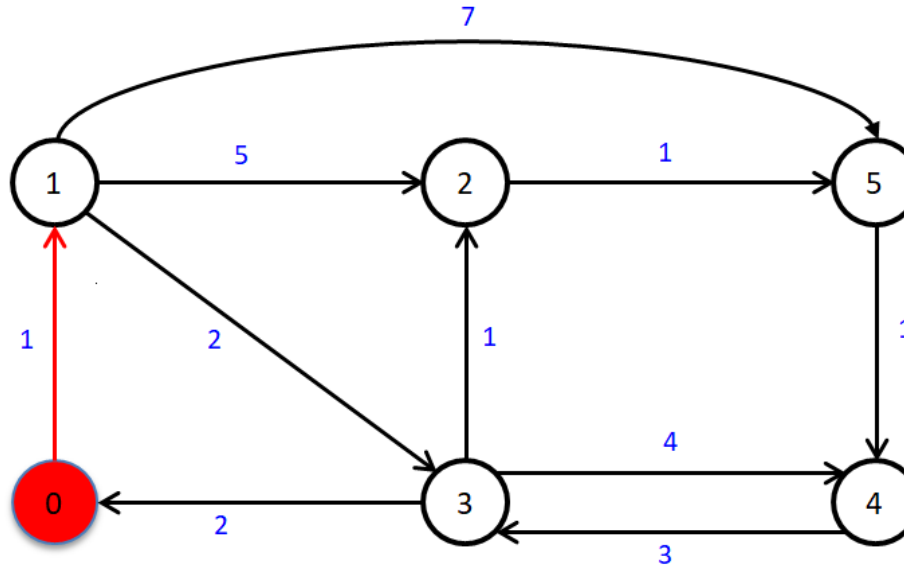
Chi phí trong bảng dist thay đổi (đang đứng ở đỉnh 0, chi phí 0).

- (1, 1):  $0 + 1 < \infty \rightarrow$  Cập nhật chi phí đỉnh 1 của mảng dist.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	$\infty \rightarrow 1$	$\infty$	$\infty$	$\infty$	$\infty$

# Bước 1: Chạy thuật toán lần 1



Lưu cặp giá trị (1, 1) vào hàng đợi ưu tiên.

priority queue

0

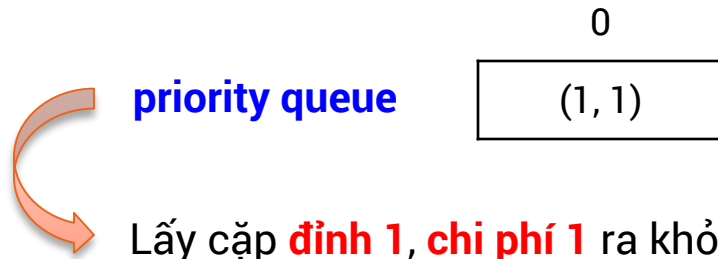
(1, 1)

Lưu vết đường đi của đỉnh 1 lại.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	-1	-1	-1	-1

# Bước 2: Chạy thuật toán lần 2



Lấy cặp **đỉnh 1, chi phí 1** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 1.

graph

Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	(3, 3)	(4, 1)

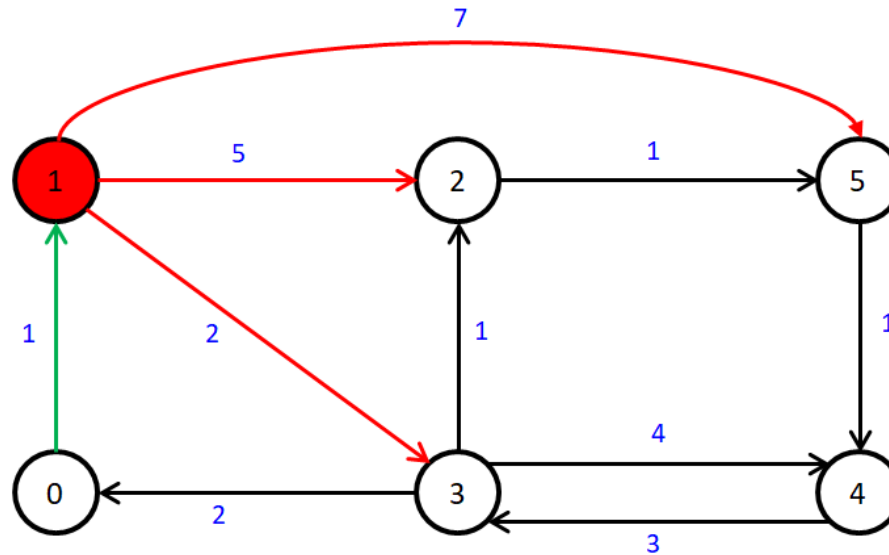
Chi phí trong bảng dist thay đổi (đang đứng ở đỉnh 1, chi phí 1).

- (2, 5):  $1 + 5 < \infty \rightarrow$  Cập nhật chi phí đỉnh 2 của mảng dist.
- (3, 2):  $1 + 2 < \infty \rightarrow$  Cập nhật chi phí đỉnh 3 của mảng dist.
- (5, 7):  $1 + 7 < \infty \rightarrow$  Cập nhật chi phí đỉnh 5 của mảng dist.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	1	$\infty \rightarrow 6$	$\infty \rightarrow 3$	$\infty$	$\infty \rightarrow 8$

# Bước 2: Chạy thuật toán lần 2




Lưu cặp giá trị (2, 6), (3, 3), (5, 8) vào hàng đợi ưu tiên, thứ tự trong hàng đợi ưu tiên thay đổi.

	0	1	2
priority queue	(3, 3)	(2, 6)	(5, 8)

Lưu vết đường đi của đỉnh 2, 3, 5 lại.

	Đỉnh	0	1	2	3	4	5
path	Lưu vết	-1	0	1	1	-1	1

# Bước 3: Chạy thuật toán lần 3

 **priority queue**

0	1	2
(3, 3)	(2, 6)	(5, 8)

Lấy cặp **đỉnh 3, chi phí 3** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 3.

**graph**

Đỉnh	0	1	2	<b>3</b>	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	<b>(0, 2)</b> <b>(2, 1)</b> <b>(4, 4)</b>	(3, 3)	(4, 1)

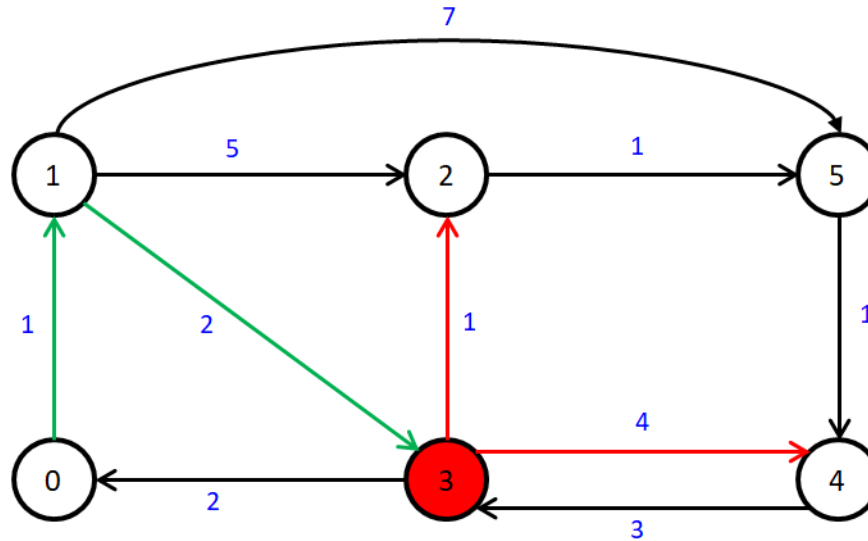
Chi phí trong bảng dist thay đổi (đang đứng ở đỉnh 3, chi phí 3).

- (0, 2):  $3 + 2 > 0 \rightarrow$  **KHÔNG** cập nhật mảng dist.
- (2, 1):  $3 + 1 < 6 \rightarrow$  Cập nhật chi phí đỉnh 2 của mảng dist.
- (4, 4):  $3 + 4 < \infty \rightarrow$  Cập nhật chi phí đỉnh 4 của mảng dist.

**dist**

Đỉnh	0	1	<b>2</b>	3	<b>4</b>	5
Chi phí	0	1	<b>6 <math>\rightarrow</math> 4</b>	3	<b><math>\infty \rightarrow 7</math></b>	8

# Bước 3: Chạy thuật toán lần 3



Lưu cặp giá trị (2, 4), (4, 7) vào hàng đợi ưu tiên, thứ tự trong hàng đợi ưu tiên thay đổi.

	0	1	2	3
priority queue	(2, 4)	(2, 6)	(4, 7)	(5, 8)

Lưu vết đường đi của đỉnh 2, 4 lại.

	Đỉnh	0	1	2	3	4	5
path	Lưu vết	-1	0	3	1	3	1

# Bước 4: Chạy thuật toán lần 4



priority queue

0	1	2	3
(2, 4)	(2, 6)	(4, 7)	(5, 8)

Lấy cặp **đỉnh 2, chi phí 4** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 2.

graph

Đỉnh	0	1	<b>2</b>	3	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	<b>(5, 1)</b>	(0, 2) (2, 1) (4, 4)	(3, 3)	(4, 1)

Chi phí trong bảng dist thay đổi (đang đứng ở đỉnh 2, chi phí 4).

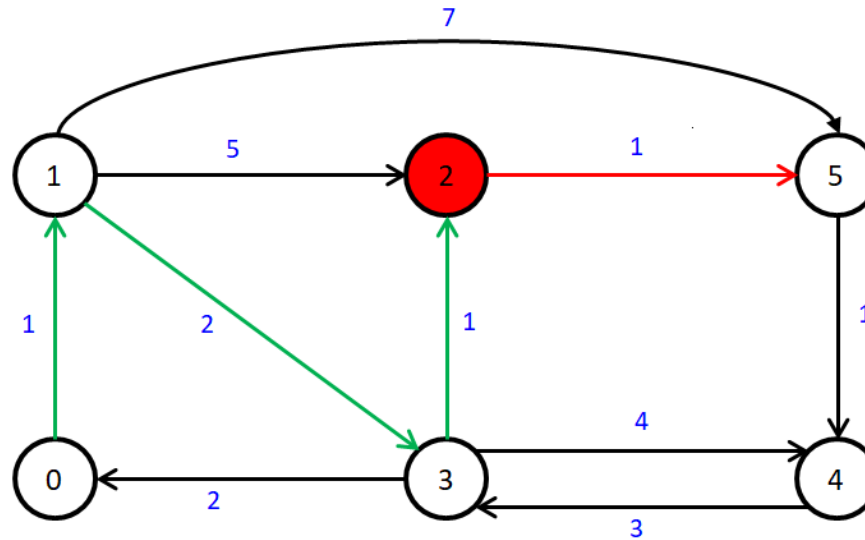
- (5, 1):  $4 + 1 < 8 \rightarrow$  Cập nhật chi phí đỉnh 5 của mảng dist.

dist

Đỉnh	<b>0</b>	1	2	3	4	<b>5</b>
Chi phí	<b>0</b>	1	4	3	7	<b>8 <math>\rightarrow</math> 5</b>



# Bước 4: Chạy thuật toán lần 4



Lưu cặp giá trị (5, 5) vào hàng đợi ưu tiên, thứ tự trong hàng đợi ưu tiên thay đổi.

	0	1	2	3
priority queue	(5, 5)	(2, 6)	(4, 7)	(5, 8)

Lưu vết đường đi của đỉnh 5 lại.

	Đỉnh	0	1	2	3	4	5
path	Lưu vết	-1	0	3	1	3	2

# Bước 5: Chạy thuật toán lần 5



priority queue

0	1	2	3
(5, 5)	(2, 6)	(4, 7)	(5, 8)

Lấy cặp **đỉnh 5, chi phí 5** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 5.

graph

Đỉnh	0	1	2	3	4	<b>5</b>
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	(3, 3)	<b>(4, 1)</b>

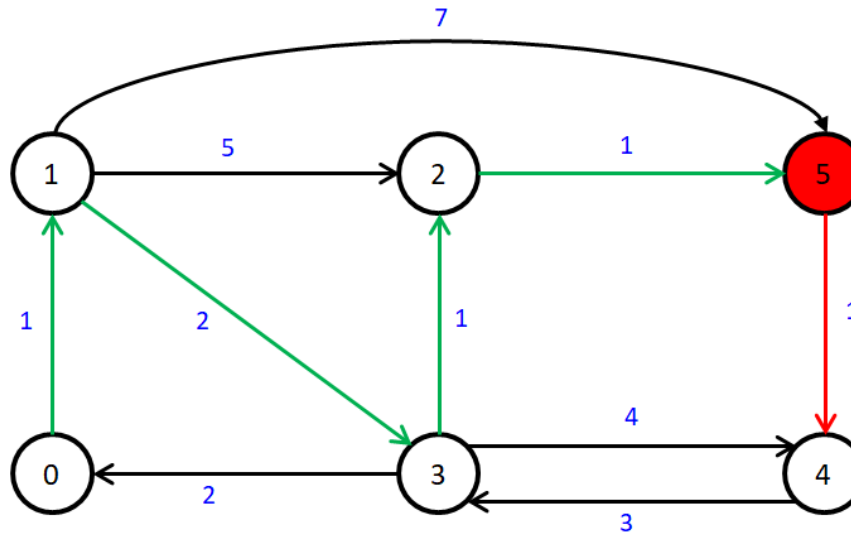
Chi phí trong bảng dist thay đổi (đang đứng ở đỉnh 5, chi phí 5).

- (4, 1):  $5 + 1 < 7 \rightarrow$  Cập nhật chi phí đỉnh 4 của mảng dist.

dist

Đỉnh	<b>0</b>	1	2	3	<b>4</b>	5
Chi phí	<b>0</b>	1	4	3	<b>7 <math>\rightarrow</math> 6</b>	5

# Bước 5: Chạy thuật toán lần 5



Lưu cặp giá trị (4, 6) vào hàng đợi ưu tiên, thứ tự trong hàng đợi ưu tiên thay đổi.

	0	1	2	3
priority queue	(2, 6)	(4, 6)	(4, 7)	(5, 8)

Lưu vết đường đi của đỉnh 4 lại.

	Đỉnh	0	1	2	3	4	5
path	Lưu vết	-1	0	3	1	5	2

# Bước 6: Chạy thuật toán lần 6



priority queue

0	1	2	3
(2, 6)	(4, 6)	(4, 7)	(5, 8)

Lấy cặp **đỉnh 2, chi phí 6** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 2.

graph

Đỉnh	0	1	<b>2</b>	3	4	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	<b>(5, 1)</b>	(0, 2) (2, 1) (4, 4)	(3, 3)	(4, 1)

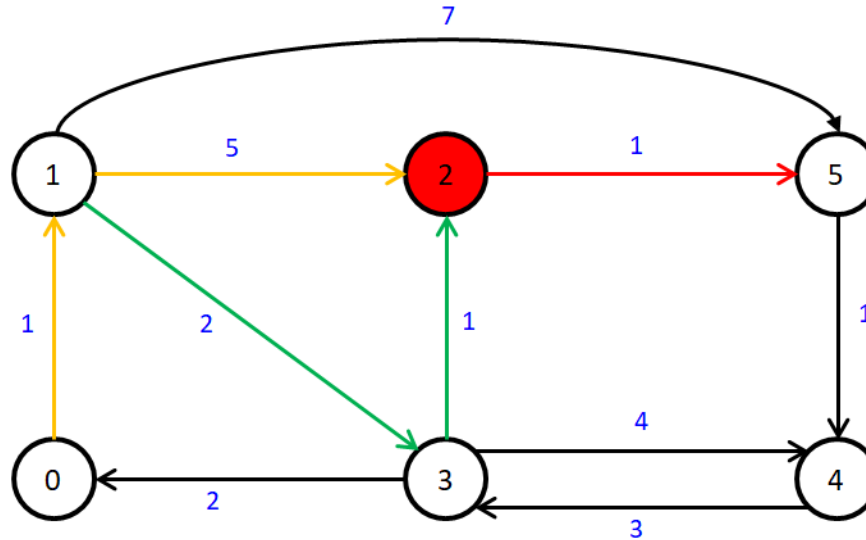
Chi phí trong bảng dist không thay đổi (đang đứng ở đỉnh 2, chi phí 6).

- (5, 1):  $6 + 1 > 5 \rightarrow$  **KHÔNG** cập nhật mảng dist.

dist

Đỉnh	<b>0</b>	1	2	3	4	5
Chi phí	<b>0</b>	1	4	3	6	5

# Bước 6: Chạy thuật toán lần 6



Không có cặp giá trị nào mới được lưu vào, trong hàng đợi ưu tiên còn 3 cặp.

priority queue

0	1	2
(4, 6)	(4, 7)	(5, 8)

Các giá trị mảng path không có sự thay đổi.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	3	1	5	2

# Bước 7: Chạy thuật toán lần 7



priority queue

0	1	2
(4, 6)	(4, 7)	(5, 8)

Lấy cặp **đỉnh 4, chi phí 6** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 4.

graph

Đỉnh	0	1	2	3	<b>4</b>	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	<b>(3, 3)</b>	(4, 1)

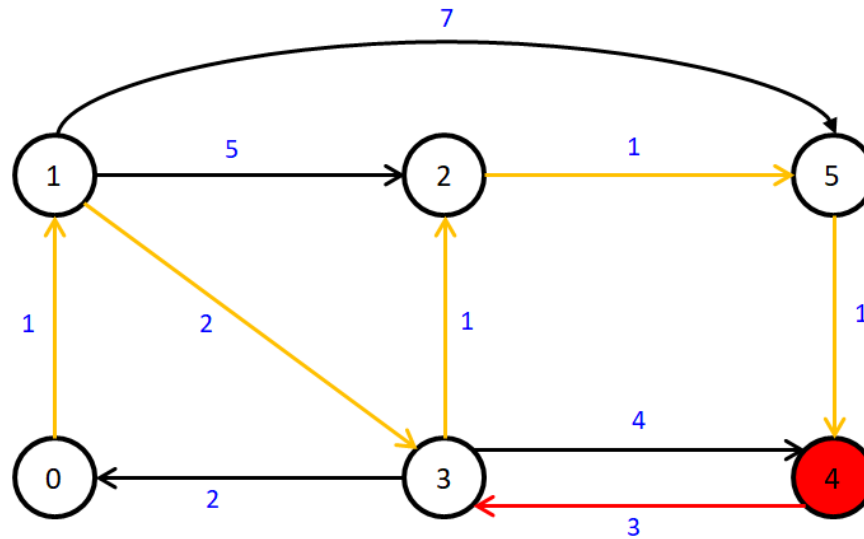
Chi phí trong bảng dist không thay đổi (đang đứng ở đỉnh 4, chi phí 6).

- (3, 3):  $6 + 3 > 3 \rightarrow$  **KHÔNG** cập nhật mảng dist.

dist

Đỉnh	<b>0</b>	1	2	3	4	5
Chi phí	<b>0</b>	1	4	3	6	5

# Bước 7: Chạy thuật toán lần 7



Không có cặp giá trị nào mới được lưu vào, trong hàng đợi ưu tiên còn 2 cặp.

**priority queue**

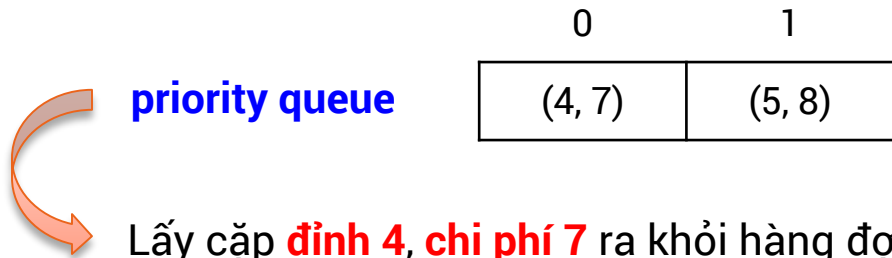
0	1
(4, 7)	(5, 8)

Các giá trị mảng path không có sự thay đổi.

**path**

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	3	1	5	2

# Bước 8: Chạy thuật toán lần 8



Lấy cặp **đỉnh 4, chi phí 7** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 4.

**graph**

Đỉnh	0	1	2	3	<b>4</b>	5
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	<b>(3, 3)</b>	(4, 1)

Chi phí trong bảng dist không thay đổi (đang đứng ở đỉnh 4, chi phí 7).

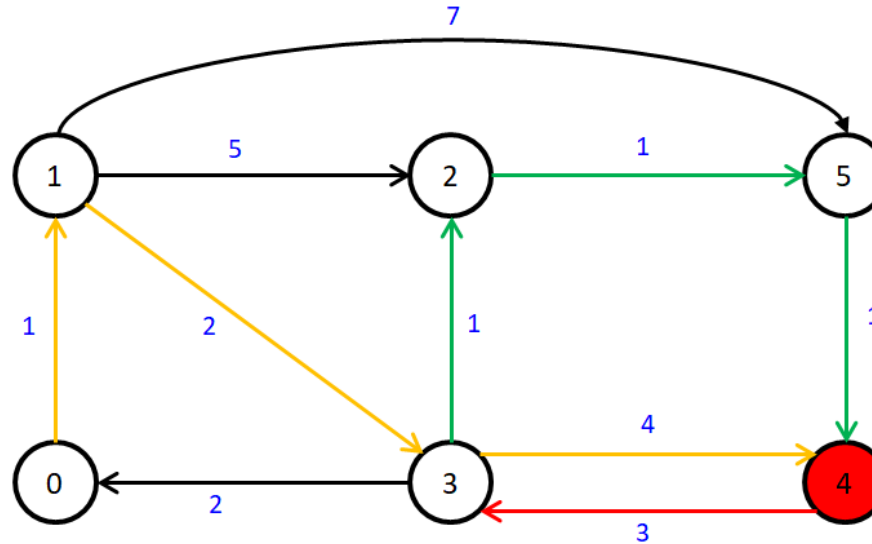
- (3, 3):  $7 + 3 > 3 \rightarrow$  **KHÔNG** cập nhật mảng dist.

**dist**

Đỉnh	<b>0</b>	1	2	3	4	5
Chi phí	<b>0</b>	1	4	3	6	5



# Bước 8: Chạy thuật toán lần 8



Không có cặp giá trị nào mới được lưu vào, trong hàng đợi ưu tiên còn 1 cặp.

priority queue

0  
(5, 8)

Các giá trị mảng path không có sự thay đổi.

	Đỉnh	0	1	2	3	4	5
path	Lưu vết	-1	0	3	1	5	2

# Bước 9: Chạy thuật toán lần 9

0

(5, 8)

priority queue



Lấy cặp **đỉnh 5, chi phí 8** ra khỏi hàng đợi và xét các đỉnh có kết nối với đỉnh 5.

graph

Đỉnh	0	1	2	3	4	<b>5</b>
(Đỉnh kề, trọng số)	(1, 1)	(2, 5) (3, 2) (5, 7)	(5, 1)	(0, 2) (2, 1) (4, 4)	(3, 3)	<b>(4, 1)</b>

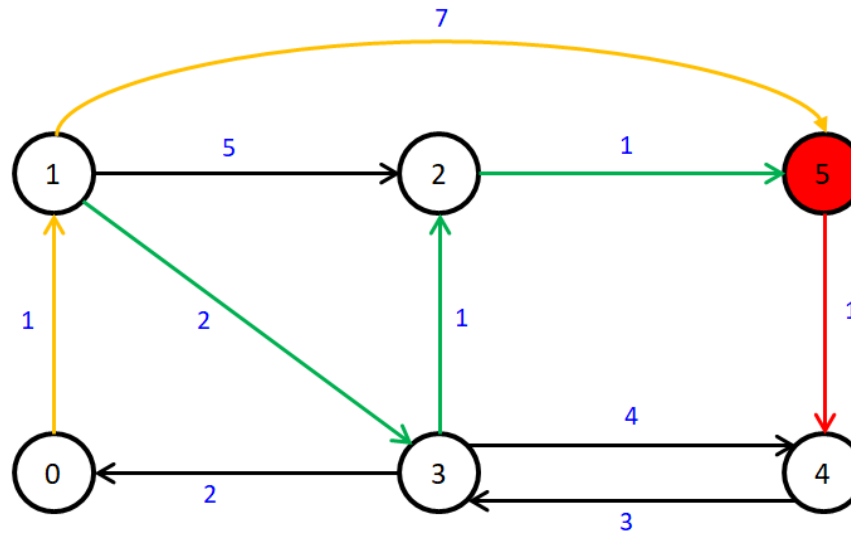
Chi phí trong bảng dist không thay đổi (đang đứng ở đỉnh 5, chi phí 8).

- (4, 1):  $8 + 1 > 6 \rightarrow$  **KHÔNG** cập nhật mảng dist.

dist

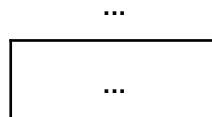
Đỉnh	<b>0</b>	1	2	3	4	5
Chi phí	<b>0</b>	1	4	3	6	5

# Bước 9: Chạy thuật toán lần 9



Không có cặp giá trị nào mới được lưu vào, hàng đợi ưu tiên rỗng → Dừng thuật toán.

priority queue



Các giá trị mảng path không có sự thay đổi.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	3	1	5	2

# Kết quả chạy Dijkstra

Tìm đường đi ngắn nhất từ 0 đến 4.



path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	3	1	5	2

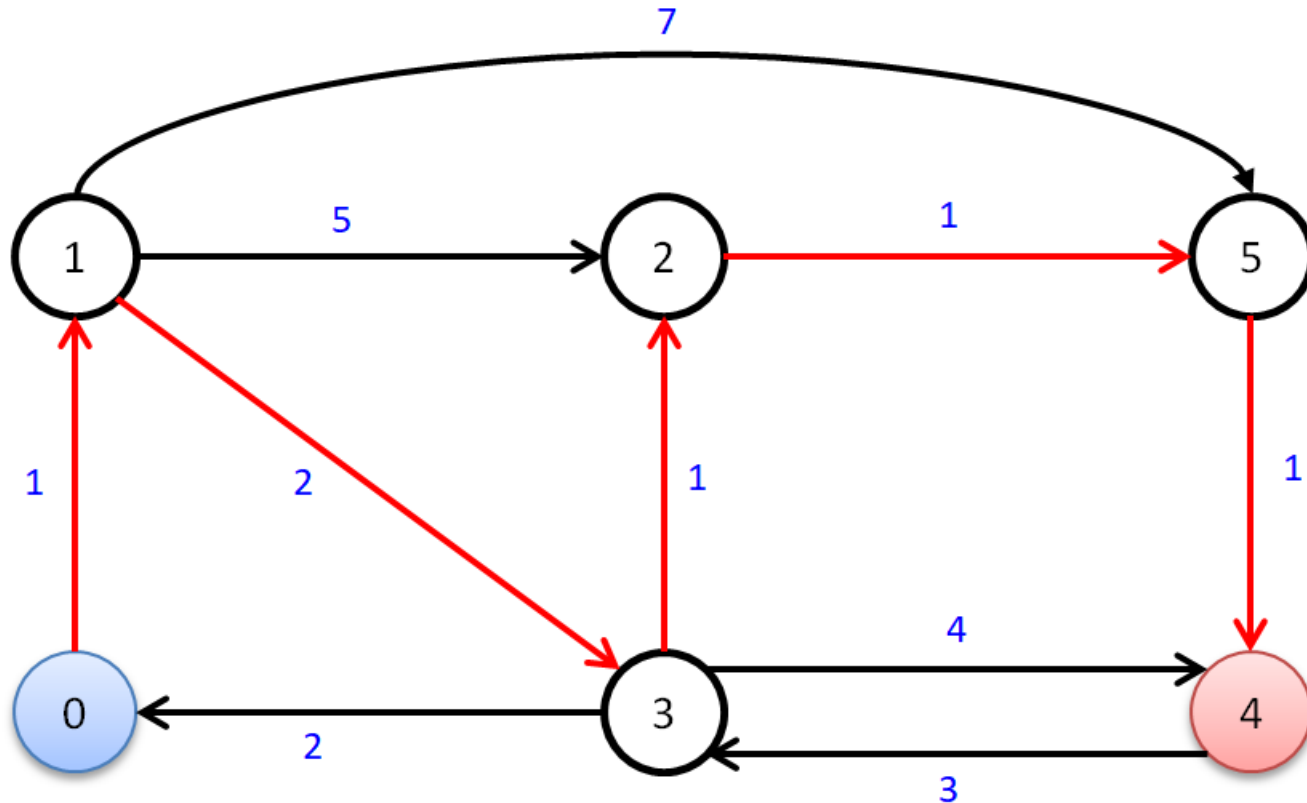
dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	1	4	3	6	5

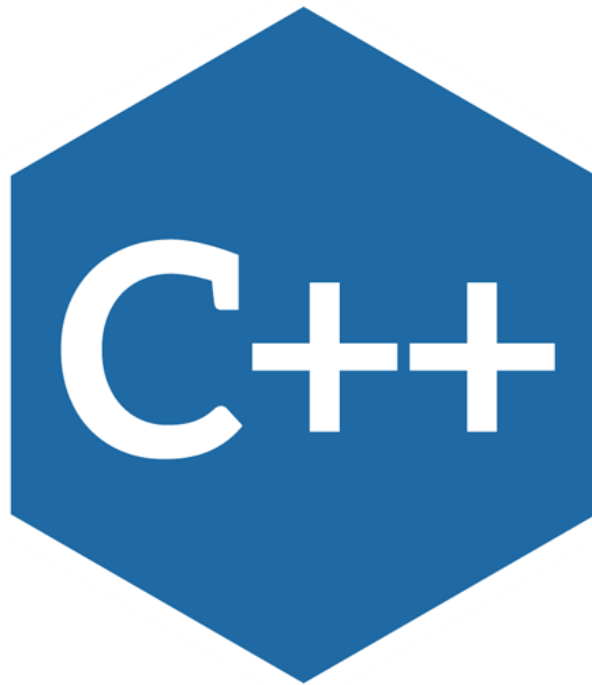
**0 → 1 → 3 → 2 → 5 → 4**  
**Cost: 6**

# Kết quả chạy Dijkstra

$0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4$   
Cost: 6



# MÃ NGUỒN MINH HỌA BẰNG C++



# Source Code Dijkstra

Khai báo thư viện và các biến toàn cục:

```
1.  #include <iostream>
2.  #include <vector>
3.  #include <queue>
4.  using namespace std;
5.  #define MAX 100
6.  const int INF = 1e9;
7.  vector<vector<pair<int, int> > > graph;
8.  vector<int> dist(MAX, INF);
9.  int path[MAX];
```



```
10. struct option
11. {
12.     bool operator() (const pair<int, int> &a, const pair<int, int> &b) const
13.     {
14.         return a.second > b.second;
15.     }
16. };
```

# Source Code Dijkstra

```
17. void Dijkstra(int s)
18. {
19.     priority_queue<pair<int, int>, vector<pair<int, int> >, option > pq;
20.     pq.push(make_pair(s, 0));
21.     dist[s] = 0;
22.     while (!pq.empty())
23.     {
24.         pair<int, int> top = pq.top();
25.         pq.pop();
26.         int u = top.first;
27.         int w = top.second;
        // to be continued
```





# Source Code Dijkstra



```
28.         for (int i = 0; i < graph[u].size(); ++i)
29.         {
30.             pair<int, int> neighbor = graph[u][i];
31.             if (w + neighbor.second < dist[neighbor.first])
32.             {
33.                 dist[neighbor.first] = w + neighbor.second;
34.                 pq.push(pair<int, int>(neighbor.first, dist[neighbor.first]));
35.                 path[neighbor.first] = u;
36.             }
37.         }
38.     }
39. }
```

# Source Code Dijkstra



```
40. int main()
41. {
42.     int n, s, t;
43.     cin >> n;
44.     s = 0, t = 4;
45.     graph = vector<vector<pair<int, int> > >(MAX + 5, vector<pair<int, int> >());
46.     int d = 0;
47.     for (int i = 0; i < n; i++)
48.         for (int j = 0; j < n; j++)
49.             {
50.                 cin >> d;
51.                 if(d > 0)
52.                     graph[i].push_back(pair<int, int>(j, d));
53.             }
54.     Dijkstra(s);
55.     int ans = dist[t];
56.     cout << ans;
57.     return 0;
58. }
```

# MÃ NGUỒN MINH HỌA BẰNG PYTHON



# Source Code Dijkstra

Khai báo class Node với id là tên đỉnh, dist là chi phí.

define operator `__lt__` (less than) cho class, so sánh theo dist nhỏ hơn, dùng trong PriorityQueue.

```
1. import queue
2. MAX = 100
3. INF = int(1e9)
4. class Node:
5.     def __init__(self, id, dist):
6.         self.dist = dist
7.         self.id = id
8.     def __lt__(self, other):
9.         return self.dist <= other.dist
```



# Source Code Dijkstra

## Thuật toán Dijkstra.

```
10. def Dijkstra(s):
11.     pq = queue.PriorityQueue()
12.     pq.put(Node(s, 0))
13.     dist[s] = 0
14.     while pq.empty() == False:
15.         top = pq.get()
16.         u = top.id
17.         w = top.dist
18.         for neighbor in graph[u]:
19.             if w + neighbor.dist < dist[neighbor.id]:
20.                 dist[neighbor.id] = w + neighbor.dist
21.                 pq.put(Node(neighbor.id, dist[neighbor.id]))
22.                 path[neighbor.id] = u
```



# Source Code Dijkstra

Hàm main để chạy chương trình.

```
23. if __name__ == '__main__':  
24.     n = int(input())  
25.     s, t = 0, 4  
26.     graph = [[] for i in range(n + 5)]  
27.     dist = [INF for i in range(n + 5)]  
28.     path = [-1 for i in range(n + 5)]  
29.     for i in range(n):  
30.         d = list(map(int, input().split()))  
31.         for j in range(n):  
32.             if d[j] > 0:  
33.                 graph[i].append(Node(j, d[j]))  
34.     Dijkstra(s)  
35.     ans = dist[t]  
36.     print(ans)
```



# MÃ NGUỒN MINH HỌA BẰNG JAVA



# Source Code Dijkstra

Khai báo class Node với id là tên đỉnh, dist là chi phí, kế thừa abstract Comparable. Vừa dùng để lưu đồ thị vừa dùng để lưu trữ trong PriorityQueue.

```
1. class Node implements Comparable<Node> {  
2.     public Integer id;  
3.     public Integer dist;  
4.     public Node(Integer id, Integer dist) {  
5.         this.id = id;  
6.         this.dist = dist;  
7.     }  
8.     @Override  
9.     public int compareTo(Node other) {  
10.         return this.dist.compareTo(other.dist);  
11.     }  
12. }
```






# Source Code Dijkstra

## Thuật toán Dijkstra (part 1)

```
13. public static void Dijkstra(int s) {  
14.     PriorityQueue<Node> pq = new PriorityQueue<Node>();  
15.     int n = graph.size();  
16.     dist = new int[n];  
17.     path = new int[n];  
18.     for (int i=0; i<n; i++) {  
19.         dist[i] = Integer.MAX_VALUE;  
20.         path[i] = -1;  
21.     }  
22.     pq.add(new Node(s, 0));  
23.     dist[s] = 0;  
    // to be continued
```



# Source Code Dijkstra

## Thuật toán Dijkstra (part 2)

```
24. while (!pq.isEmpty()) {  
25.     Node top = pq.poll();  
26.     int u = top.id;  
27.     int w = top.dist;  
28.     for (int i = 0; i < graph.get(u).size(); i++) {  
29.         Node neighbor = graph.get(u).get(i);  
30.         if (w + neighbor.dist < dist[neighbor.id]) {  
31.             dist[neighbor.id] = w + neighbor.dist;  
32.             pq.add(new Node(neighbor.id, dist[neighbor.id]));  
33.             path[neighbor.id] = u;  
34.         }  
35.     }  
36. }  
37. }
```

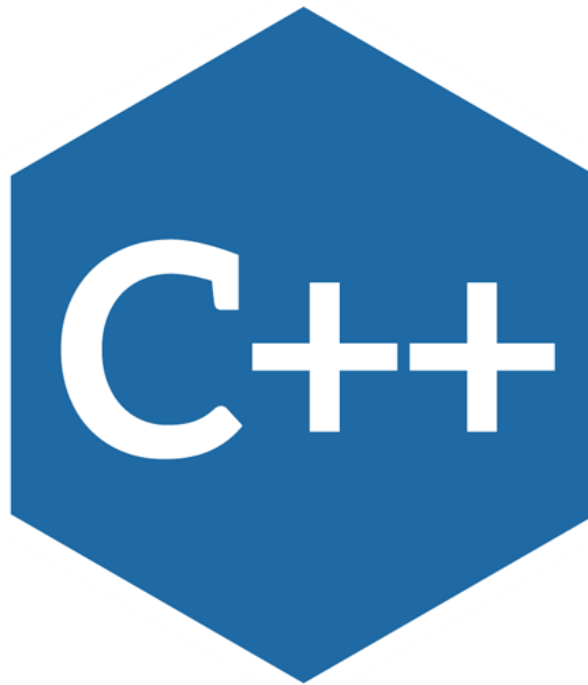


# Source Code Dijkstra



```
38. private static int[] dist;
39. private static int[] path;
40. private static ArrayList<ArrayList<Node>> graph;
41. public static void main(String[] args) {
42.     Scanner sc = new Scanner(System.in);
43.     int n = sc.nextInt();
44.     int s = 0, t = 4;
45.     graph = new ArrayList<ArrayList<Node>>();
46.     for (int i = 0; i < n; i++) {
47.         graph.add(new ArrayList<Node>());
48.         for (int j = 0; j < n; j++) {
49.             int d = sc.nextInt();
50.             if (d > 0)
51.                 graph.get(i).add(new Node(j, d));
52.         }
53.     }
54.     Dijkstra(s);
55.     System.out.println(dist[t]);
56. }
```

# BONUS CẤU TRÚC PAIR CHO C++



# Bonus: Cấu trúc pair (1)

**pair** là một cấu trúc gồm 2 thuộc tính, 2 thuộc tính này có thể cùng hoặc khác kiểu dữ liệu với nhau.

Thư viện:

```
<Thư viện bất kỳ nào đó>  
using namespace std;
```

Khai báo dạng mặc định:

```
pair<data_type1, data_type2> variable_name;
```

```
pair<int, string> p;
```

p

<b>first</b>	...
<b>second</b>	...

# Bonus: Cấu trúc pair (2)

Khai báo dạng có tham số đầu vào:

```
pair<data_type1, data_type2> variable_name (value1, value2);
```

```
pair<int, string> p (10, "abc");
```

p

first	10
second	"abc"

# Bonus: Cấu trúc pair (3)

**first, second:** Truy cập và gán giá trị cho thuộc tính của pair.

```
pair<int, string> p;  
p.first = 2;  
p.second = "abc";  
cout << p.first << "-" << p.second;
```

**2-abc**

# Bonus: Cấu trúc pair (4)

Kết hợp với vector: Tạo thành mảng động chứa các cặp giá trị.

```
vector<pair<int, string> > v;  
pair<int, string> p;  
p = make_pair(2, "abc");  
v.push_back(p);  
p = make_pair(3, "def");  
v.push_back(p);
```

0	1
2, "abc"	3, "def"



# Bonus: Cấu trúc pair (5)

Kết hợp với mảng tĩnh: Tạo thành mảng tĩnh chứa các cặp giá trị.

```
typedef pair<int, int> Student;
```

```
int main() {  
    Student a[100];  
    return 0;  
}
```

# Hỏi đáp

