

LECTURE 15

PRIM'S ALGORITHM



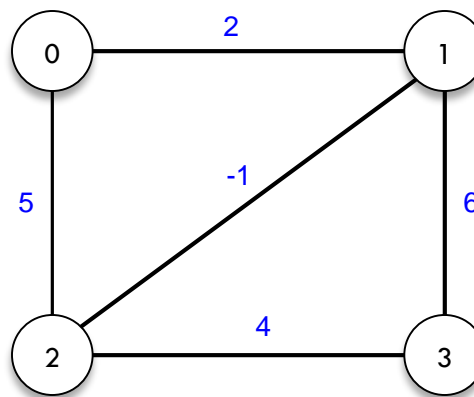
Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Minimum Spanning Tree

Minimum Spanning Tree – MST (Cây khung nhỏ nhất hay cây bao trùm nhỏ nhất) là tập hợp **một số cạnh** của đồ thị vô hướng có trọng số, sao cho tạo thành một cây chứa **tất cả các đỉnh** với tổng trọng số các cạnh là **nhỏ nhất**.

Trong một đồ thị có nhiều cây khung, MST là chúng ta tìm cây khung có trọng số nhỏ nhất.



Tổng trọng số: $-1 + 2 + 4 = 5$

Thuật toán Prim

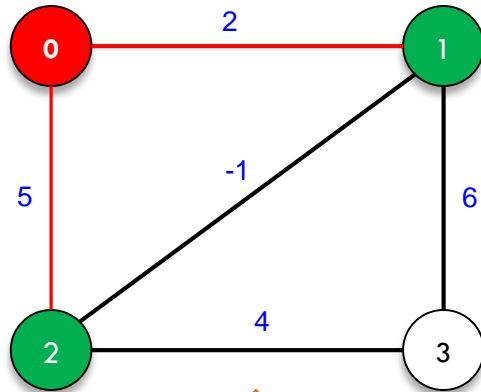
Prim's Algorithm (thuật toán Prim): là thuật toán tham lam dùng để tìm cây khung nhỏ nhất.

Tùy vào cách cài đặt, thuật toán Prim có 2 loại độ phức tạp khác nhau:

- Nếu cài đặt với thuật toán ở mức độ cơ bản (sử dụng ma trận kề) thì độ phức tạp của Prim là $O(V^2)$ hoặc $O(V^2 + E)$.
- Nếu cài đặt với **hàng đợi ưu tiên** thì độ phức tạp của Prim là $O(E \log(V))$

Ý tưởng của thuật toán

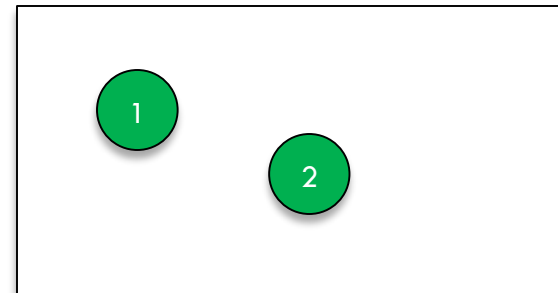
Xuất phát từ một đỉnh bất kỳ. Đi đến các đỉnh kề của đỉnh này.



So sánh chi phí cạnh hiện tại với chi phí cạnh trước đó.

Đỉnh	0	1	2	3
Chi phí	0	∞	∞	∞

Thêm vào kho.

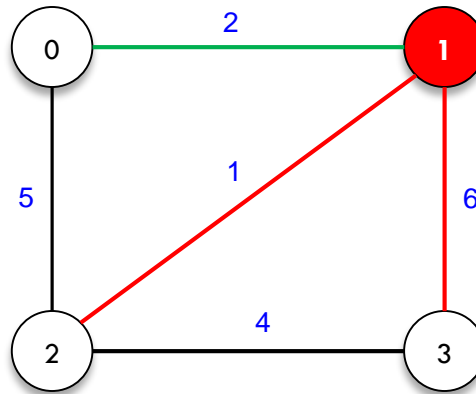


Lưu vết và đem đỉnh mới ra xét.

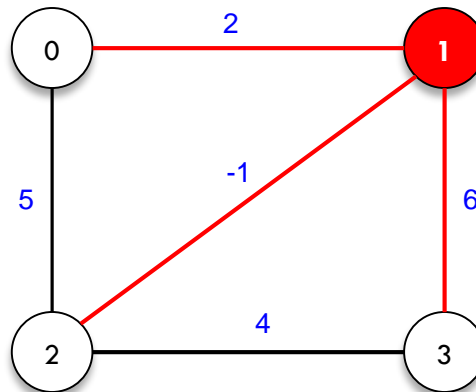
→ Dừng khi kho không còn đỉnh nào. Xuất kết quả bài toán.

Sự khác nhau giữa Prim và Dijkstra

Dijkstra: Khi bạn đang đứng tại 1 đỉnh, bạn muốn đi đến đỉnh khác, bạn phải xét **chi phí đường đi trước đó + chi phí đường đi đến đỉnh khác** < **chi phí đường đi tốt nhất hiện tại** hay không → quyết định đi / không đi.

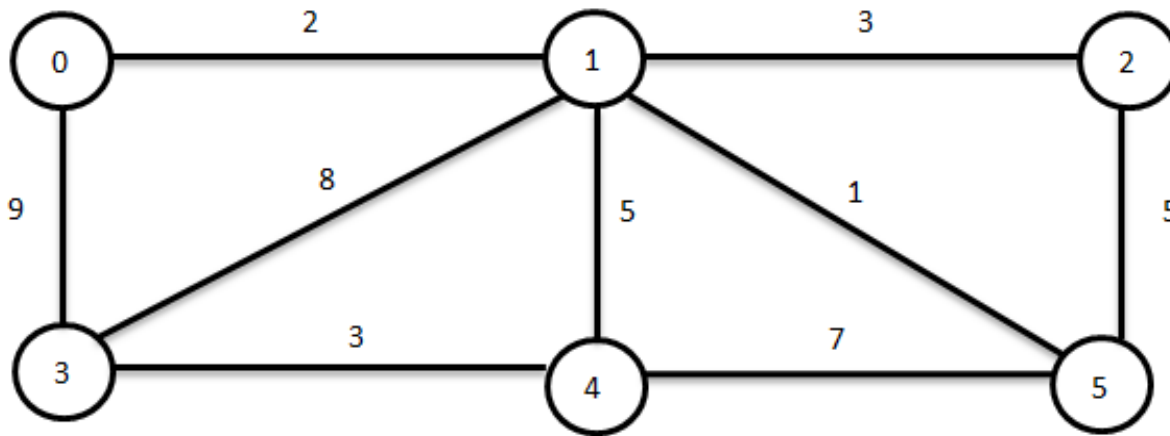


Prim: Khi bạn đang đứng tại 1 đỉnh, bạn muốn đi đến đỉnh khác, bạn cần xét **chi phí đường đi đến đỉnh khác (chi phí cạnh)** < **chi phí tốt nhất hiện tại** hay không → quyết định đi / không đi.



Bài toán minh họa

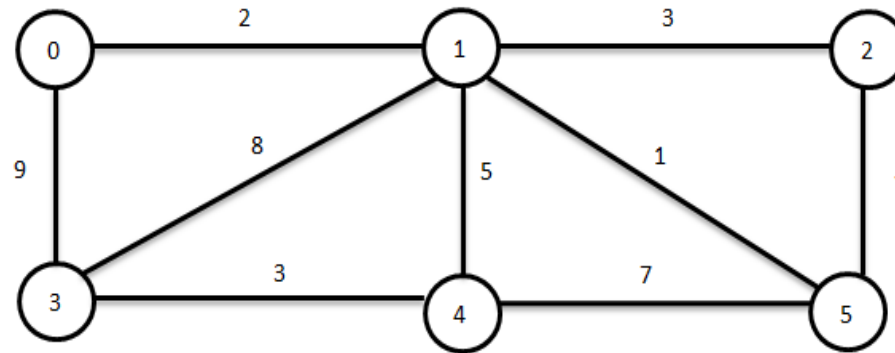
Cho đồ thị **vô hướng** như hình vẽ. Tìm **cây khung nhỏ nhất** của đồ thị bên dưới.



Edge List

6	9
0	1 2
0	3 9
1	2 3
1	3 8
1	4 5
1	5 1
2	5 5
3	4 3
4	5 7

Bước 0: Chuẩn bị dữ liệu (1)



Chuyển danh sách cạnh kề vào **graph**.

Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

Mảng chứa chi phí đường đi **dist**.

Đỉnh	0	1	2	3	4	5
Chi phí	∞	∞	∞	∞	∞	∞

Bước 0: Chuẩn bị dữ liệu (2)

Mảng đánh dấu các đỉnh đã xét **visited**.

Đỉnh	0	1	2	3	4	5
Trạng thái	false	false	false	false	false	false

Mảng lưu vết đường đi **path**.

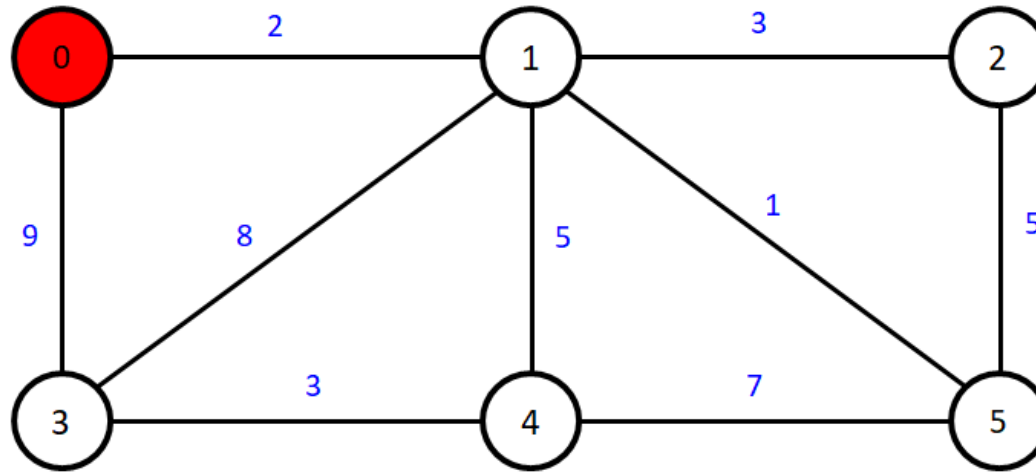
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	-1	-1	-1	-1	-1

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	∞	∞	∞	∞	∞

priority_queue - bỏ cặp (0, 0) vào hàng đợi ưu tiên.

0

(0, 0)

Bước 1: Chạy thuật toán lần 1

0

priority_queue

(0, 0)

- Lấy cặp **đỉnh 0, chi phí 0** ra khỏi hàng đợi và đánh dấu **true** tại **visited[0]**.
- Xét các đỉnh có kết nối với đỉnh 0.

graph

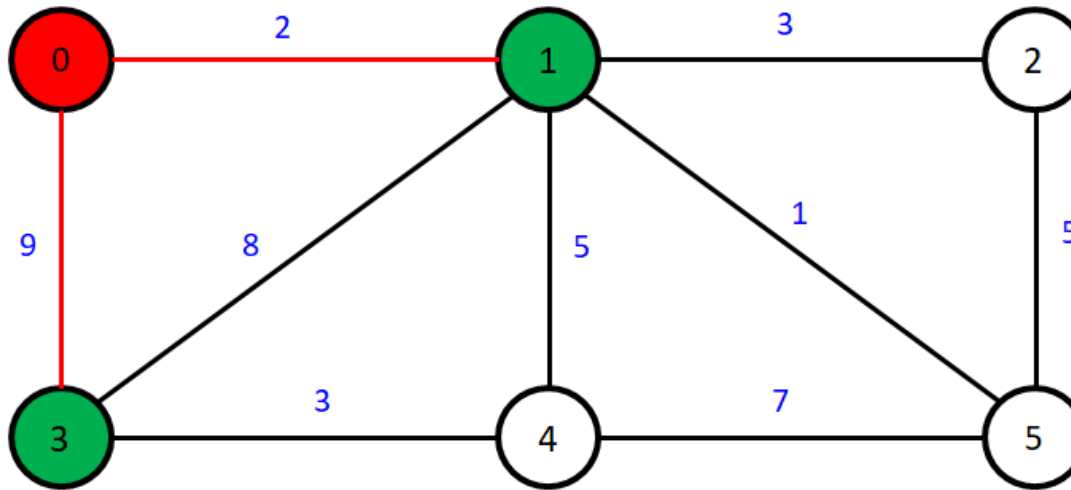
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (1, 2): $\text{dist}[1] = \infty > 2 \rightarrow$ Cập nhật $\text{dist}[1] = 2$.
- (3, 9): $\text{dist}[3] = \infty > 9 \rightarrow$ Cập nhật $\text{dist}[3] = 9$.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	$\infty \rightarrow 2$	∞	$\infty \rightarrow 9$	∞	∞

Bước 1: Chạy thuật toán lần 1



Lưu cặp giá trị (1, 2) và (3, 9) vào hàng đợi ưu tiên.

priority_queue

	0	1
	(1, 2)	(3, 9)

Cập nhật giá trị của đỉnh 1 và đỉnh 3.

path

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

Bước 2: Chạy thuật toán lần 2

priority_queue

	0	1
	(1, 2)	(3, 9)

- Lấy cặp **đỉnh 1, chi phí 2** ra khỏi hàng đợi và đánh dấu **true** tại **visited[1]**.
- Xét các đỉnh có kết nối với đỉnh 1.

graph

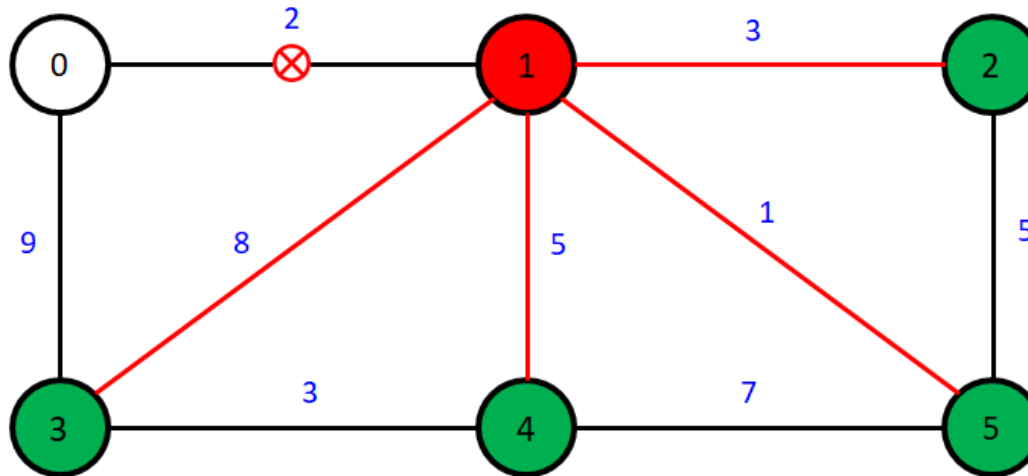
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (0, 2): đỉnh 0 đã viếng thăm → **KHÔNG** cập nhật.
- (2, 3): $\text{dist}[2] = \infty > 3 \rightarrow$ Cập nhật $\text{dist}[2] = 3$.
- (3, 8): $\text{dist}[3] = 9 > 8 \rightarrow$ Cập nhật $\text{dist}[3] = 8$.
- (4, 5): $\text{dist}[4] = \infty > 5 \rightarrow$ Cập nhật $\text{dist}[4] = 5$.
- (5, 1): $\text{dist}[5] = \infty > 1 \rightarrow$ Cập nhật $\text{dist}[5] = 1$.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	$\infty \rightarrow 3$	$9 \rightarrow 8$	$\infty \rightarrow 5$	$\infty \rightarrow 1$

Bước 2: Chạy thuật toán lần 2



Lưu các cặp (2, 3), (3, 8), (4, 5) và (5, 1) vào hàng đợi ưu tiên.

	0	1	2	3	4
priority_queue	(5, 1)	(2, 3)	(4, 5)	(3, 8)	(3, 9)

Cập nhật giá trị của các đỉnh 2, 3, 4, 5.

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

Bước 3: Chạy thuật toán lần 3



priority_queue

0	1	2	3	4
(5, 1)	(2, 3)	(4, 5)	(3, 8)	(3, 9)

- Lấy cặp **đỉnh 5, chi phí 1** ra khỏi hàng đợi và đánh dấu **true** tại **visited[5]**.
- Xét các đỉnh có kết nối với đỉnh 5.

graph

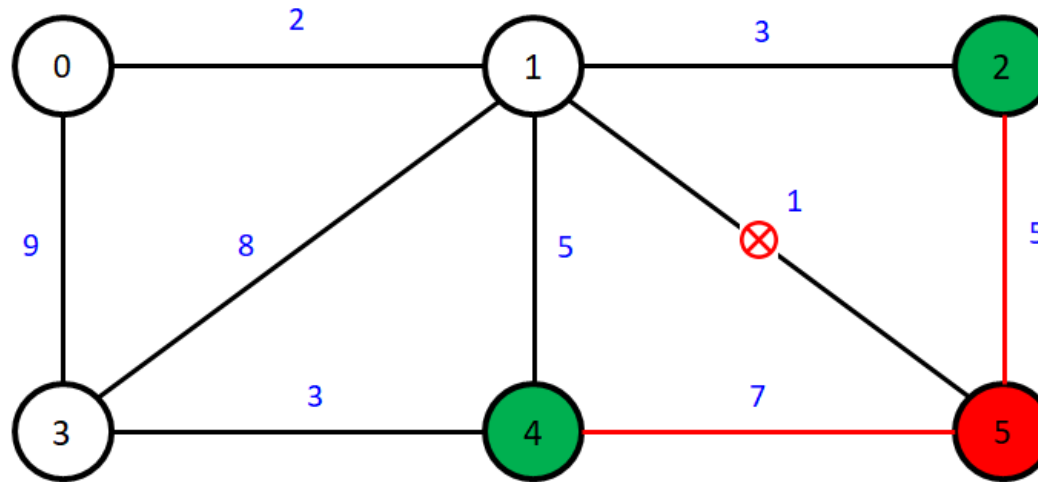
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (1, 1): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (2, 5): $\text{dist}[2] = 3 < 5$ → **KHÔNG** cập nhật.
- (4, 7): $\text{dist}[4] = 5 < 7$ → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8	5	1

Bước 3: Chạy thuật toán lần 3



Không có giá trị mới nào cần lưu vào hàng đợi ưu tiên.

	0	1	2	3
priority_queue	(2, 3)	(4, 5)	(3, 8)	(3, 9)

Không có giá trị đỉnh nào cần cập nhật.

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	1	1	1

Bước 4: Chạy thuật toán lần 4

priority_queue

0	1	2	3
(2, 3)	(4, 5)	(3, 8)	(3, 9)

- Lấy cặp **đỉnh 2, chi phí 3** ra khỏi hàng đợi và đánh dấu **true** tại **visited[2]**.
- Xét các đỉnh có kết nối với đỉnh 2.

graph

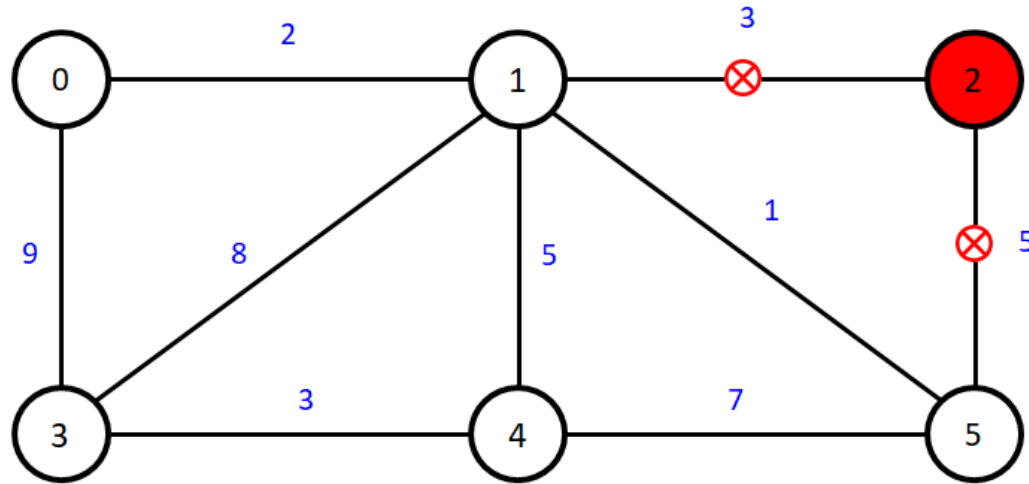
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (1, 3): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (5, 5): đỉnh 5 đã viếng thăm → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8	5	1

Bước 4: Chạy thuật toán lần 4



Không có giá trị mới nào cần lưu vào hàng đợi ưu tiên.

	0	1	2
priority_queue	(4, 5)	(3, 8)	(3, 9)

Không có giá trị đỉnh nào cần cập nhật.

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

Bước 5: Chạy thuật toán lần 5

priority_queue

0	1	2
(4, 5)	(3, 8)	(3, 9)

- Lấy cặp **đỉnh 4, chi phí 5** ra khỏi hàng đợi và đánh dấu **true** tại **visited[4]**.
- Xét các đỉnh có kết nối với đỉnh 4.

graph

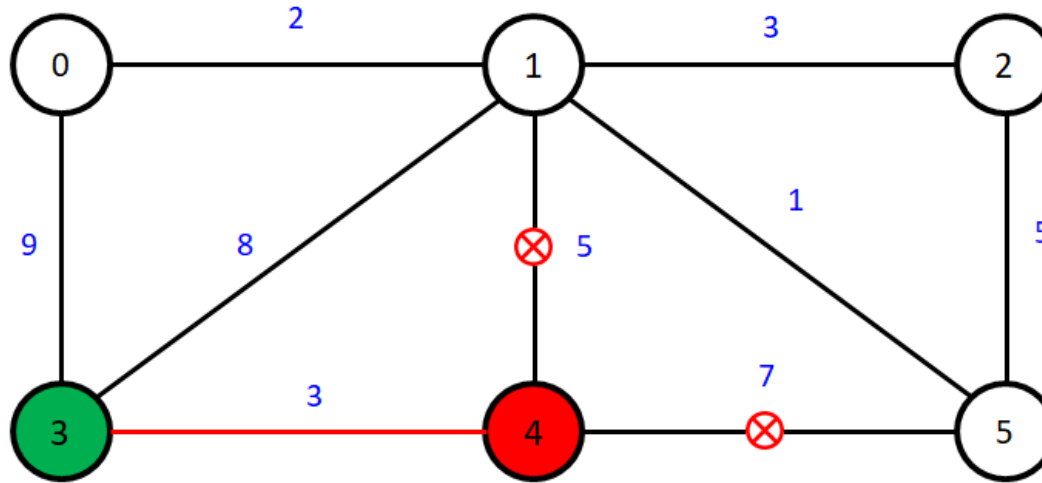
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (1, 5): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (3, 3): $\text{dist}[3] = 8 > 3 \rightarrow$ Cập nhật $\text{dist}[3] = 3$.
- (5, 7): đỉnh 5 đã viếng thăm → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8 → 3	5	1

Bước 5: Chạy thuật toán lần 5




Lưu cặp giá trị (3, 3) vào hàng đợi ưu tiên.

	0	1	2
priority_queue	(3, 3)	(3, 8)	(3, 9)

Cập nhật giá trị của đỉnh 3.

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

Bước 6: Chạy thuật toán lần 6

 **priority_queue**

0	1	2
(3, 3)	(3, 8)	(3, 9)

- Lấy cặp **đỉnh 3, chi phí 3** ra khỏi hàng đợi và đánh dấu **true** tại **visited[3]**.
- Xét các đỉnh có kết nối với đỉnh 3.

graph

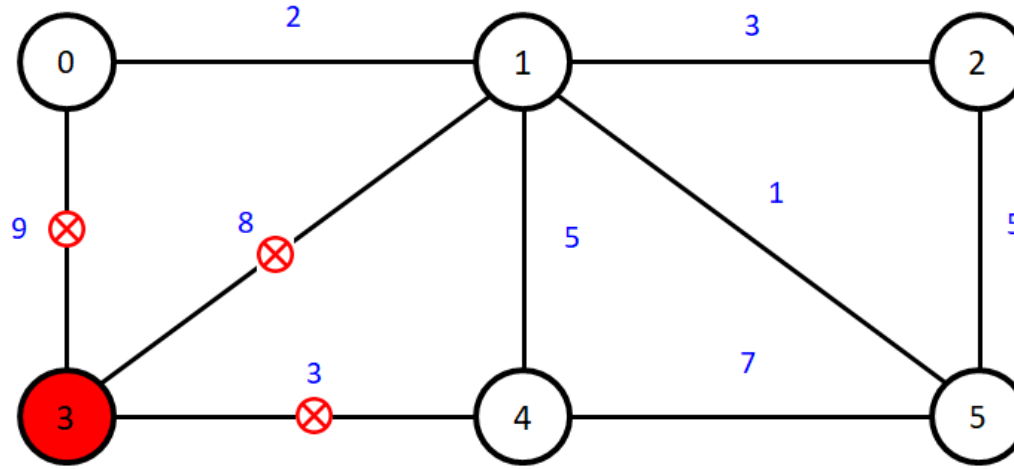
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (0, 9): đỉnh 0 đã viếng thăm → **KHÔNG** cập nhật.
- (1, 8): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (4, 3): đỉnh 4 đã viếng thăm → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

Bước 6: Chạy thuật toán lần 6



Không có giá trị mới nào cần lưu vào hàng đợi ưu tiên.

priority_queue

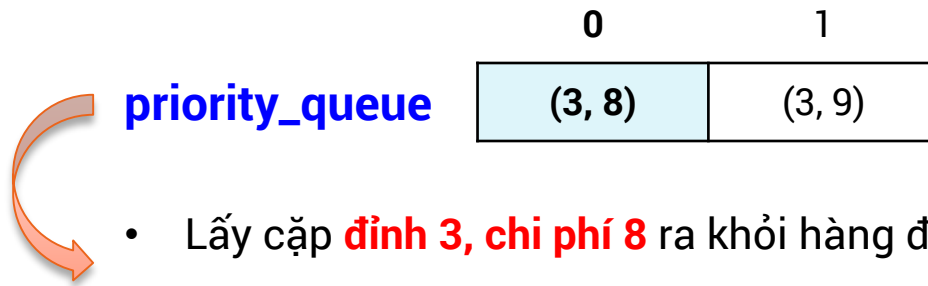
	0	1
	(3, 8)	(3, 9)

Không có giá trị đỉnh nào cần cập nhật.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	4	1	1

Bước 7: Chạy thuật toán lần 7



- Lấy cặp **đỉnh 3, chi phí 8** ra khỏi hàng đợi và đánh dấu **true** mảng **visited**.
- Xét các đỉnh có kết nối với đỉnh 3.

graph

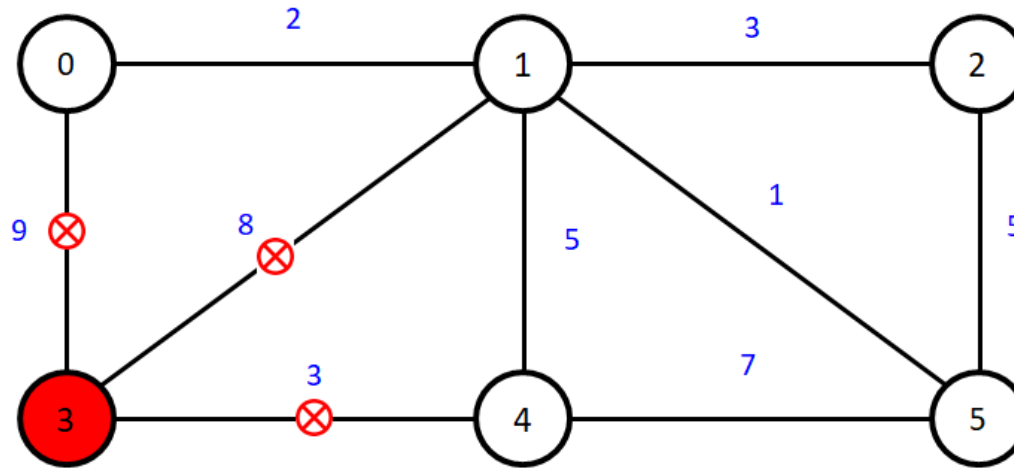
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (0, 9): đỉnh 0 đã viếng thăm → **KHÔNG** cập nhật.
- (1, 8): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (4, 3): đỉnh 4 đã viếng thăm → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

Bước 7: Chạy thuật toán lần 7



Không có giá trị mới nào cần lưu vào hàng đợi ưu tiên.

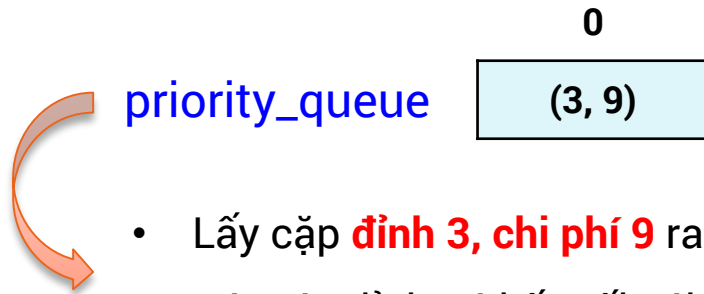
priority_queue 0
(3, 9)

Không có giá trị đỉnh nào cần cập nhật.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	4	1	1

Bước 8: Chạy thuật toán lần 8



- Lấy cặp **đỉnh 3, chi phí 9** ra khỏi hàng đợi và đánh dấu **true** mảng **visited**.
- Xét các đỉnh có kết nối với đỉnh 3.

graph

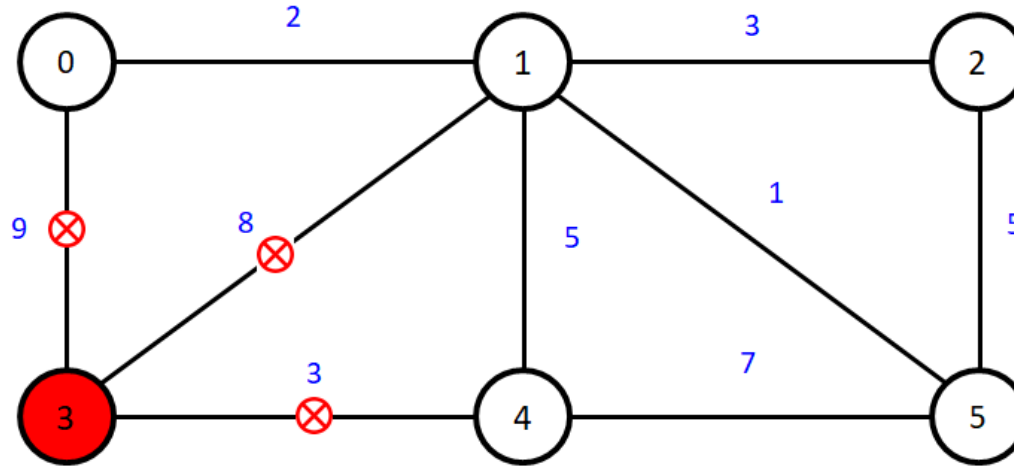
Đỉnh	0	1	2	3	4	5
(Đỉnh kề, trọng số)	(1, 2) (3, 9)	(0, 2) (2, 3) (3, 8) (4, 5) (5, 1)	(1, 3) (5, 5)	(0, 9) (1, 8) (4, 3)	(1, 5) (3, 3) (5, 7)	(1, 1) (2, 5) (4, 7)

- (0, 9): đỉnh 0 đã viếng thăm → **KHÔNG** cập nhật.
- (1, 8): đỉnh 1 đã viếng thăm → **KHÔNG** cập nhật.
- (4, 3): đỉnh 4 đã viếng thăm → **KHÔNG** cập nhật.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

Bước 8: Chạy thuật toán lần 8



Không có giá trị mới nào cần lưu vào hàng đợi ưu tiên.

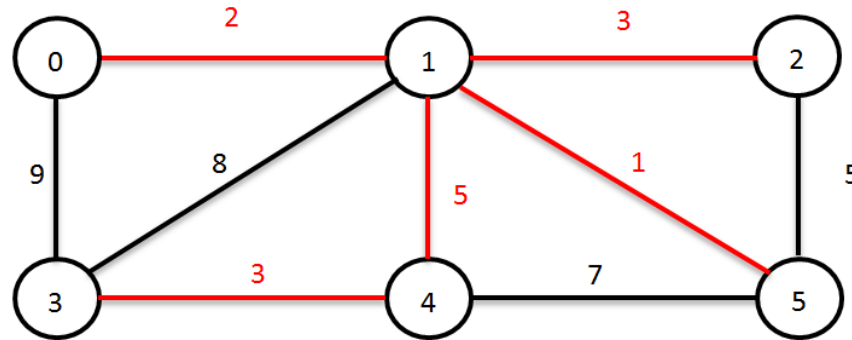
priority_queue ...  **Dừng thuật toán.**

Không có giá trị đỉnh nào cần cập nhật.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	4	1	1

Kết quả chạy Prim



path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	4	1	1

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

0 – 1: 2

1 – 2: 3

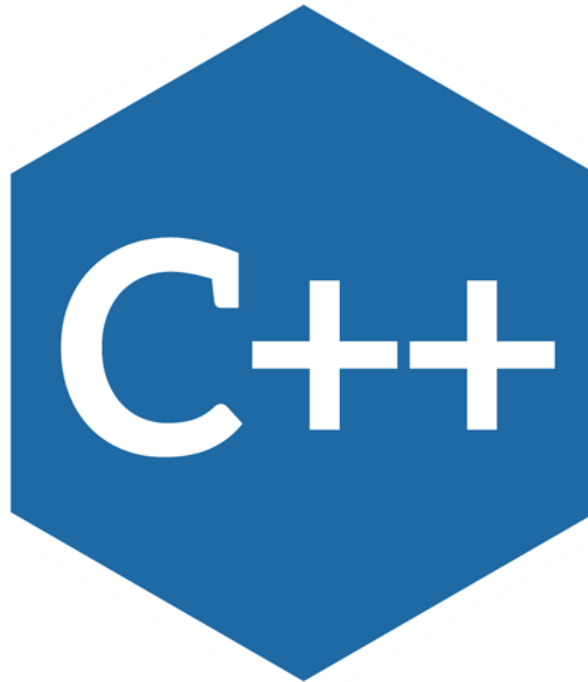
4 – 3: 3

1 – 4: 5

1 – 5: 1

Weight of MST: 14

MÃ NGUỒN MINH HỌA BẰNG C++



Source Code Prim

Khai báo thư viện và các biến toàn cục:

```
1. #include <algorithm>
2. #include <iostream>
3. #include <string>
4. #include <vector>
5. #include <queue>
6. #include <functional>
7. using namespace std;
8. #define MAX 100
9. const int INF = 1e9;
10. vector<pair<int, int> > graph[MAX];
11. vector<int> dist(MAX, INF);
12. int path[MAX];
13. bool visited[MAX];
14. int N, M;
```



Source Code Prim



```
15. void printMST()
16. {
17.     int ans = 0;
18.     for (int i = 0; i<N; i++)
19.     {
20.         if (path[i] == -1)
21.             continue;
22.         ans += dist[i];
23.         cout << path[i] << " - " << i << ": " << dist[i]<<endl;
24.     }
25.     cout<<"Weight of MST: "<<ans<<endl;
26. }
```

```
27. struct option
28. {
29.     bool operator() (const pair<int, int> &a, const pair<int, int> &b) const
30.     {
31.         return a.second > b.second;
32.     }
33. };
```

Source Code Prim



```
34. void prims(int src)
35. {
36.     priority_queue<pair<int, int>, vector<pair<int, int> >,
37.                   option > pq;
38.     pq.push(make_pair(src, 0));
39.     dist[src] = 0;
40.     while (!pq.empty())
41.     {
42.         int u = pq.top().first;
43.         pq.pop();
44.         visited[u] = true;
// to be continued
```

Source Code Prim

Thuật toán Prim (part 2)

```
45.         for (int i = 0; i < graph[u].size(); i++)
46.         {
47.             int v = graph[u][i].first;
48.             int w = graph[u][i].second;
49.             if (!visited[v] && dist[v] > w)
50.             {
51.                 dist[v] = w;
52.                 pq.push(make_pair(v, w));
53.                 path[v] = u;
54.             }
55.         } //end for
56.     } //end while
57. }
```



Source Code Prim



```
58. int main()
59. {
60.     int u, v, w;
61.     cin >> N >> M;
62.     memset(path, -1, sizeof(path));
63.     for (int i = 0; i < M; i++)
64.     {
65.         cin >> u >> v >> w;
66.         graph[u].push_back(make_pair(v, w));
67.         graph[v].push_back(make_pair(u, w));
68.     }
69.     int s = 0;
70.     prims(s);
71.     printMST();
72.     return 0;
73. }
```


MÃ NGUỒN MINH HỌA BẰNG PYTHON



Source Code Prim

Khai báo thư viện và các biến toàn cục:

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



Source Code Prim

In ra cây khung nhỏ nhất tìm được:

```
9. def printMST():
10.     ans = 0
11.     for i in range(n):
12.         if path[i] == -1:
13.             continue
14.         ans += dist[i]
15.         print("{0} - {1}: {2}".format(path[i], i, dist[i]))
16.     print("Weight of MST: {0}".format(ans))
```



Source Code Prim



```
17. def prims(src):
18.     pq = queue.PriorityQueue()
19.     pq.put(Node(src, 0))
20.     dist[src] = 0
21.     while pq.empty() == False:
22.         top = pq.get()
23.         u = top.id
24.         visited[u] = True
25.         for neighbor in graph[u]:
26.             v = neighbor.id
27.             w = neighbor.dist
28.             if visited[v] == False and w < dist[v]:
29.                 dist[v] = w
30.                 pq.put(Node(v, w))
31.                 path[v] = u
```

Source Code Prim

Hàm main chương trình

```
32. if __name__ == '__main__':
33.     n, m = map(int, input().split())
34.     graph = [[] for i in range(n)]
35.     dist = [INF for i in range(n)]
36.     path = [-1 for i in range(n)]
37.     visited = [False for i in range(n)]
38.     for i in range(m):
39.         u, v, w = map(int, input().split())
40.         graph[u].append(Node(v, w))
41.         graph[v].append(Node(u, w))
42.     prims(0)
43.     printMST()
```



MÃ NGUỒN MINH HỌA BẰNG JAVA



Source Code Prim



```
1. import java.util.ArrayList;
2. import java.util.Arrays;
3. import java.util.PriorityQueue;
4. import java.util.Scanner;
5. class Node implements Comparable<Node> {
6.     public Integer id;
7.     public Integer dist;
8.     public Node(Integer id, Integer dist) {
9.         this.id = id;
10.        this.dist = dist;
11.    }
12.    @Override
13.    public int compareTo(Node other) {
14.        return this.dist.compareTo(other.dist);
15.    }
16. }
```

Source Code Prim



```
17. class Main {
18.     private static int[] dist;
19.     private static int[] path;
20.     private static boolean[] visited;
21.     private static void printMST() {
22.         int n = dist.length;
23.         int ans = 0;
24.         for (int i = 0; i < n; i++) {
25.             if (path[i] == -1) {
26.                 continue;
27.             }
28.             ans += dist[i];
29.             System.out.printf("%d -
                                   %d: %d\n", path[i], i, dist[i]);
30.         }
31.         System.out.printf("Weight of MST: %d", ans);
32.     }
```


Source Code Prim

```
33. public static void prims(int src, ArrayList<ArrayList<Node>> graph) {
34.     PriorityQueue<Node> pq = new PriorityQueue<Node>();
35.     int n = graph.size();
36.     dist = new int[n];
37.     path = new int[n];
38.     visited = new boolean[n];
39.     Arrays.fill(dist, Integer.MAX_VALUE);
40.     Arrays.fill(path, -1);
41.     Arrays.fill(visited, false);
42.     pq.add(new Node(src, 0));
43.     dist[src] = 0;
```



Source Code Prim



```
44.     while (!pq.isEmpty()) {
45.         Node top = pq.poll();
46.         int u = top.id;
47.         visited[u] = true;
48.         for (int i = 0; i < graph.get(u).size(); i++) {
49.             Node neighbor = graph.get(u).get(i);
50.             int v = neighbor.id, w = neighbor.dist;
51.             if (!visited[v] && w < dist[v]) {
52.                 dist[v] = w;
53.                 pq.add(new Node(v, w));
54.                 path[v] = u;
55.             }
56.         }
57.     }
58. }
```

Source Code Prim



```
59. public static void main (String[] args) {
60.     Scanner sc = new Scanner(System.in);
61.     int n = sc.nextInt(), m = sc.nextInt();
62.     ArrayList<ArrayList<Node>> graph = new ArrayList<ArrayList<Node>>();
63.     for (int i = 0; i < n; i++) {
64.         graph.add(new ArrayList<Node>());
65.     }
66.     for (int i = 0; i < m; i++) {
67.         int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
68.         graph.get(u).add(new Node(v, w));
69.         graph.get(v).add(new Node(u, w));
70.     }
71.     prims(0, graph);
72.     printMST();
73. }
```

Hỏi đáp

