Big-O Blue

LECTURE 16

PRIM'S ALGORITHM

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Định nghĩa Prim

Thuật toán **Prim** là thuật toán tham lam dùng để tìm "cây bao trùm nhỏ nhất" (cây khung nhỏ nhất - Minimum Spanning Tree MST). Nghĩa là **Prim** sẽ tìm tập hợp **các cạnh** của đồ thị vô hướng sao cho tạo thành một cây chứa tất cả các đỉnh, với tổng trọng số các cạnh của cây là **nhỏ nhất**.

Độ phức tạp của Prim

Độ phức tạp: Có 2 loại độ phức tạp.

- Nếu cài đặt với Thuật Toán mức cơ bản (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(ElogV).**

Ý tưởng của thuật toán

Xuất phát từ một đỉnh bất kỳ. Đi tới tất cả các đỉnh kề của đỉnh này, nếu đỉnh kề nào là đỉnh chưa được thăm và có chi phí đi đến đó, nhỏ hơn chi phí đường đi hiện tại thì cập nhật lại chi phí và lưu đỉnh này lại.

Tiếp tục đem 1 đỉnh khác (từ tập đỉnh đã được lưu) ra xét và đi cho đến khi không còn đỉnh nào có thể đi. Mỗi bước đi nếu gặp chi phí mới nhỏ hơn chi phí hiện tại thì **cập nhật** lại.

Trong quá trình cập nhật chi phí, tiến hành **lưu đỉnh cha** của đỉnh kề. Kết quả lưu lại cuối cùng là đường đi với chi phí nhỏ nhất đi từ đỉnh xuất phát đến tất cả các đỉnh trong đồ thị.

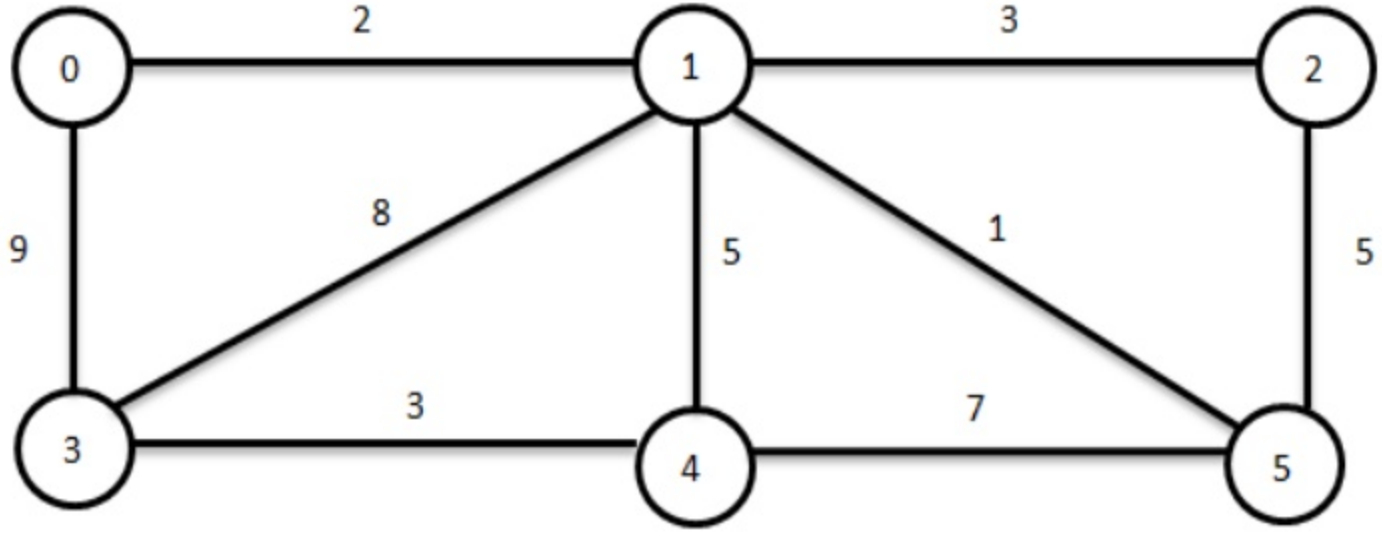
Sự khác nhau Prim và Dijkstra

Dijkstra: Khi bạn đang đứng tại 1 đỉnh, bạn muốn đi đến đỉnh khác, bạn phải xét chi phí **đường đi trước đó + đường đi đến đỉnh khác < chi phí 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 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.



Bước 0: Chuẩn bị dữ liệu

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

```
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

Chuyển danh sách kề vào **graph**.

Đỉnh	0	1	2	3	4	5
Pair	(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

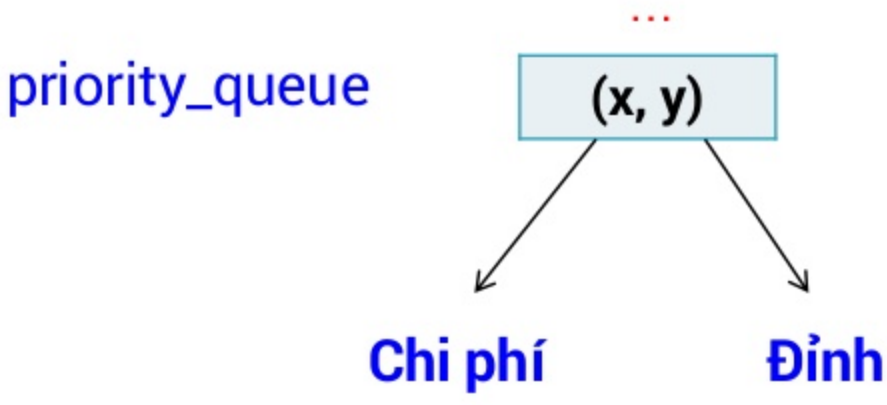
Mảng đánh dấu các đỉnh đã xét.

	0	1	2	3	4	5
visited	false	false	false	false	false	false

Mảng lưu vết đường đi.

	0	1	2	3	4	5
path	-1	-1	-1	-1	-1	-1

Hàng đợi ưu tiên lưu cặp giá trị.



Bước 1: Chạy thuật toán lần 1

Lấy đỉnh bắt đầu đi là **đỉnh 0**. Gán chi phí cho đỉnh 0 là 0.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	∞	∞	∞	∞	∞

Bỏ cặp (0, 0) vào hàng đợi.

priority_queue

0
(0, 0)

Mảng đánh dấu các đỉnh đã xét.

visited

0	1	2	3	4	5
true	false	false	false	false	false

Bước 1: Chạy thuật toán lần 1

Lấy cặp **giá trị 0**, **đỉnh 0** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 0.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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): dist[1] = ∞ > 2 → Cập nhật dist[1] = 2.
- (3, 9): dist[3] = ∞ > 9 → Cập nhật dist[3] = 9.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	∞ → 2	∞	∞ → 9	∞	∞

Bước 1: Chạy thuật toán lần 1

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	∞	9	∞	∞

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

priority_queue

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

Lưu giá trị đỉnh cha của đỉnh 1 và 3 lại.

path

0	1	2	3	4	5
-1	0	-1	0	-1	-1

Bước 2: Chạy thuật toán lần 2

Lấy cặp **giá trị 2**, **đỉnh 1** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 1.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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): dist[2] = ∞ > 3 → Cập nhật dist[2] = 3.
- (3, 8): dist[3] = 9 > 8 → Cập nhật dist[3] = 8.
- (4, 5): dist[4] = ∞ > 5 → Cập nhật dist[4] = 5.
- (5, 1): dist[5] = ∞ > 1 → Cập nhật dist[5] = 1.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	∞ → 3	9 → 8	∞ → 5	∞ → 1

Bước 2: Chạy thuật toán lần 2

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8	5	1

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

priority_queue

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

Lưu giá trị đỉnh cha của các đỉnh 2, 3, 4, 5 lại.

path

0	1	2	3	4	5
-1	0	1	1	1	1

Bước 3: Chạy thuật toán lần 3

Lấy cặp **giá trị 1**, **đỉnh 5** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 5.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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): dist[2] = 3 < 5 → KHÔNG cập nhật.
- (4, 7): 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

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8	5	1

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

priority_queue

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

Không có giá trị đỉnh nào cần lưu đỉnh cha lại.

path

0	1	2	3	4	5
-1	0	1	1	1	1

Bước 4: Chạy thuật toán lần 4

Lấy cặp **giá trị 3**, **đỉnh 2** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 2.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	8	5	1

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

priority_queue

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

Không có giá trị đỉnh nào cần lưu đỉnh cha lại.

path

0	1	2	3	4	5
-1	0	1	1	1	1

Bước 5: Chạy thuật toán lần 5

Lấy cặp **giá trị 5**, **đỉnh 4** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 4.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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): dist[3] = 8 > 3 → Cập nhật 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

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

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

priority_queue

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

Lưu giá trị đỉnh cha của đỉnh 3 lại.

path

0	1	2	3	4	5
-1	0	1	4	1	1

Bước 6: Chạy thuật toán lần 6

Lấy cặp **giá trị 3**, **đỉnh 3** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh đỉnh 3.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

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

priority_queue

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

Không có giá trị đỉnh nào cần lưu đỉnh cha lại.

path

0	1	2	3	4	5
-1	0	1	4	1	1

Bước 7: Chạy thuật toán lần 7

Lấy cặp **giá trị 8**, **đỉnh 3** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 3.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

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

priority_queue

0
(9, 3)

Không có giá trị đỉnh nào cần lưu đỉnh cha lại.

path

0	1	2	3	4	5
-1	0	1	4	1	1

Bước 8: Chạy thuật toán lần 8

Lấy cặp **giá trị 9**, **đỉnh 3** ra khỏi hàng đợi và xem xét các đỉnh có kết nối với đỉnh 3.

graph

Đỉnh	0	1	2	3	4	5
Pair	(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.

dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

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

priority_queue

0
...

→ dừng thuật toán.

Không có giá trị đỉnh nào cần lưu đỉnh cha lại.

path

0	1	2	3	4	5
-1	0	1	4	1	1

Kết quả chạy Prim

Cây khung bao trùm nhỏ nhất của đồ thị.

path

0	1	2	3	4	5
-1	0	1	4	1	1

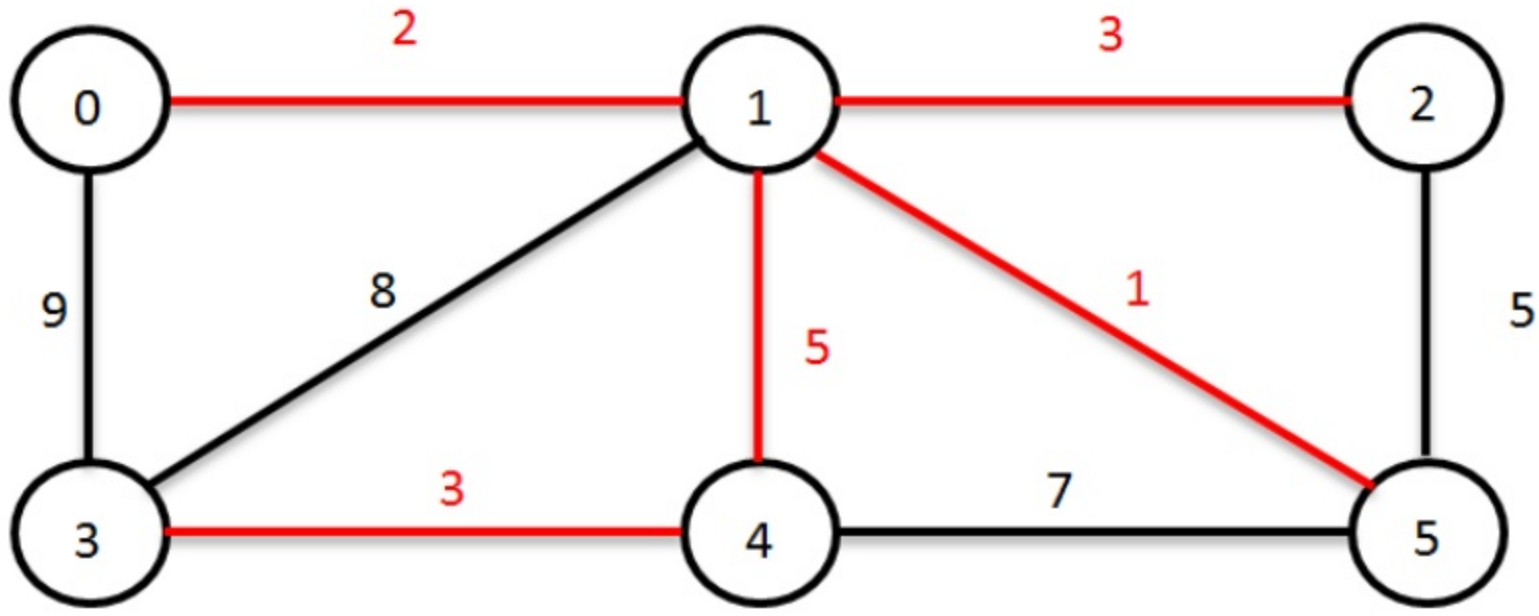
dist

Đỉnh	0	1	2	3	4	5
Chi phí	0	2	3	3	5	1

Kết quả

```
0 – 1: 2
2 – 1: 3
3 – 4: 3
1 – 4: 5
1 – 5: 1
Weight of MST: 14
```

Kết quả chạy Prim



MÃ NGUỒN MINH HỌA BẰNG C++

Source Code Prim C++

Khai báo thư viện và các biến toàn cục:

```
#include <algorithm>
#include <iostream>
#include <string>
#include <vector>
#include <queue>
#include <functional>
using namespace std;
#define MAX 100
const int INF = 1e9;
vector<pair<int, int> > graph[MAX];
vector<int> dist(MAX, INF);
int path[MAX];
bool visited[MAX];
int N;
```

Source Code Prim C++

In ra cây khung nhỏ nhất tìm được:

```
void printMST() {
    int ans = 0;
    for (int i = 0; i<N; i++) {
        if (path[i] == -1)
            continue;
        ans += dist[i];
        cout << path[i] << " - " << i << ": " << dist[i]<<endl;
    }
    cout<<"Weight of MST: "<<ans<<endl;
}
```

Source Code Prim C++

Thuật toán Prim (part 1)

```
void Prims(int src) {
    priority_queue<pair<int, int>, vector<pair<int, int> >,
                                        greater<pair<int, int> > >pq;
    pq.push(make_pair(0, src));
    dist[src] = 0;
    while (!pq.empty()) {
          int u = pq.top().second;
          pq.pop();
          visited[u] = true;
// to be continued
```

Source Code Prim C++

Thuật toán Prim (part 2)

```
        for (int i = 0; i<graph[u].size(); i++) {
             int v = graph[u][i].first;
             int w = graph[u][i].second;
             if (!visited[v] && dist[v] > w) {
                  dist[v] = w;
                  pq.push(make_pair(w, v));
                  path[v] = u;
             }
         } //end for
    } //end while
}
```

Source Code Prim C++

Hàm main chương trình

```
int main() {
	int M, u, v, w;
	cin >> N >> M;
	memset(path, -1, sizeof(path));
	for (int i = 0; i<M; i++) {
		cin >> u >> v >> w;
		graph[u].push_back(make_pair(v, w));
		graph[v].push_back(make_pair(u, w));
	}
	int s = 0;
	Prims(s);
	printMST();
	return 0;
}
```

MÃ NGUỒN MINH HỌA BẰNG PYTHON

Source Code Prim python

Khai báo thư viện và các biến toàn cục:

```
import queue


INF = 1e9


class Node:
    dist = 0
    id = 0
    def __init__(self, id, dist):
        self.dist = dist
        self.id = id
    def __lt__(self, other):
        return self.dist <= other.dist
```

Source Code Prim python

In ra cây khung nhỏ nhất tìm được:

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

Source Code Prim python

```
def Prim(src):
    pq = queue.PriorityQueue()
    pq.put(Node(src, 0))
    dist[src] = 0
    while pq.empty() == False:
        top = pq.get()
        u = top.id
        d = top.dist
        visited[u] = True
        for neighbor in graph[u]:
            v = neighbor.id
            w = neighbor.dist
            if visited[v] == False and w < dist[v]:
                dist[v] = w
                pq.put(Node(v, w))
                path[v] = u
```

Source Code Prim python

Hàm main chương trình

```
if __name__ == '__main__':
    n, m = map(int, input().split())
    graph = [[] for i in range(n + 1)]
    dist = [INF for i in range(n + 1)]
    path = [-1 for i in range(n + 1)]
    visited = [False for i in range(n + 1)]
    for i in range(m):
        u, v, w = map(int, input().split())
        graph[u].append(Node(v, w))
        graph[v].append(Node(u, w))
    Prim(0)
    PrintMST()
```

MÃ NGUỒN MINH HỌA BẰNG JAVA

Source Code Prim Java

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.

```
class Node implements Comparable<Node> {
    public Integer id;
    public Integer dist;
    public Node(Integer id, Integer dist) {
        this.id = id;
        this.dist = dist;
    }
    @Override
    public int compareTo(Node other) {
        return this.dist.compareTo(other.dist);
    }
}
```

Source Code Prim Java

In ra cây khung nhỏ nhất tìm được:

```
private static void printMST() {
    int n = dist.length;
    int ans = 0;
    for (int i = 0; i < n; i++) {
        if (path[i] == -1) {
            continue;
        }
        ans += dist[i];
        System.out.printf("%d - %d: %d\n", path[i], i, dist[i]);
    }
    System.out.printf("Weight of MST: %d", ans);
}
```

Source Code Prim Java

Thuật toán Prim (part 1)

```
public static void prim(int source, ArrayList<ArrayList<Node>> graph)
{
    PriorityQueue<Node> pq = new PriorityQueue<Node>();
    int n = graph.size();
    dist = new int[n];
    path = new int[n];
    visited = new boolean[n];
    Arrays.fill(dist, Integer.MAX_VALUE);
    Arrays.fill(path, -1);
    Arrays.fill(visited, false);

    pq.add(new Node(source, 0));
    dist[source] = 0;
```

Source Code Prim Java

Thuật toán Prim (part 2)

```
    while (!pq.isEmpty()) {
        Node top = pq.poll();
        int u = top.id;
        visited[u] = true;
        for (int i = 0; i < graph.get(u).size(); i++) {
            Node neighbor = graph.get(u).get(i);
            int v = neighbor.id, w = neighbor.dist;
            if (!visited[v] && w < dist[v]) {
                dist[v] = w;
                pq.add(new Node(v, w));
                path[v] = u;
            }
        }
    }
}
```

Source Code Prim Java

Big-O Blue

Hàm main để chạy chương trình

```
// khai báo đầu class
private static int[] dist;
private static int[] path;
private static boolean[] visited;
// main
public static void main (String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt(), m = sc.nextInt();
    ArrayList<ArrayList<Node>> graph = new ArrayList<ArrayList<Node>>();
    for (int i = 0; i < n; i++) {
        graph.add(new ArrayList<Node>());
    }
    for (int i = 0; i < m; i++) {
        int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
        graph.get(u).add(new Node(v, w));
        graph.get(v).add(new Node(u, w));
    }
    prim(0, graph);
    printMST();
}
```

Hỏi đáp