Bài 10.6: Thuật toán Floyd-Warshall

- ✓ Khái niệm và đặc điểm
- ✓ Mã giả và triển khai
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Khái niệm và đặc điểm

- Thuật toán Floyed-Warshall dùng để tìm đường đi ngắn nhất trong đồ thị có hướng với cạnh có trọng số âm hoặc dương. Đồ thị phải đảm bảo không chứa chu trình âm.
- Mỗi lần thực hiện thuật toán sẽ tìm độ dài của các đường đi ngắn nhất giữa tất cả các cặp đỉnh.
- Floyd-Warshall là thuật toán thuộc loại quy hoạch động được công bố vào năm 1962.
- Thuật toán có độ phức tạp O($|V|^3$). Với V là số đỉnh của đồ thị.
- Thuật toán này được sử dụng trong trường hợp tổng quát để tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị.

Ứng dụng

- Tìm đường đi ngắn nhất trên đồ thị.
- Tìm sự giống nhau giữa các đồ thị.
- Định tuyến tối ưu.
- Nghịch đảo ma trận thực.

Mã giả

➢ Sau đây là mã giả thuật toán Floyd-Warshall:

```
// thuật toán Floyd-Warshall
function FloydWarshall(G):
    // B1: khởi tạo ma trận khoảng cách, ma trận next
    dist[][] = ma trận cấp |V| x |V|
    next[][] = ma trận cấp |V| x |V| khởi tạo các phần tử null
    for(i từ 0 đến |V| - 1):
        for(j từ 0 đến |V| - 1):
            if(i != j):
                dist[i][j] = dương vô cực
            else:
                dist[i][j] = 0
                next[i][j] = i
    for(từng cạnh (u, v) trong tập cạnh với trọng số w):
        dist[u][v] = w
        next[u][v] = v
    // B2: lặp
    for(k từ 0 đến |V| - 1):
        for(i từ 0 đến |V| - 1):
            for(j từ 0 đến |V| - 1):
                alt = dist[i][k] + dist[k][j]
                if(dist[i][j] > alt):
                    dist[i][j] = alt
                    next[i][j] = next[i][k]
    return dist[][], next[][]
```

Triển khai

➢ Triển khai thuật toán Floyd-Warshall:

```
public static long[][] floydWarshall(Vertex[] vertices, List<Edge> edges, Vertex[][] next) {
    // b1: khởi tạo ma trận khoảng cách và ma trận lưu vết kết quả
    long[][] dist = new long[vertices.length][vertices.length];
    for (var i = 0; i < vertices.length; i++) {
        for (int j = 0; j < vertices.length; j++) {
            if (i != j) {
                dist[i][j] = Integer.MAX_VALUE;
            } else { // i == j
                next[i][j] = vertices[i];
            }
        }
    }
    for (Edge edge : edges) {
        dist[edge.start][edge.end] = edge.weight;
        next[edge.start][edge.end] = vertices[edge.end];
    }
    // b2: lặp tính khoảng cách nhỏ nhất giữa các cặp đỉnh
    for (int k = 0; k < vertices.length; k++) {
        for (int i = 0; i < vertices.length; i++) {
            for (int j = 0; j < vertices.length; j++) {
                var alt = dist[i][k] + dist[k][j];
                if (dist[i][j] > alt) {
                    dist[i][j] = alt;
                    next[i][j] = next[i][k];
                }
            }
        }
    }
    return dist;
}
```

Mã giả

- Sau đây là mã giả tìm vết đường đi:

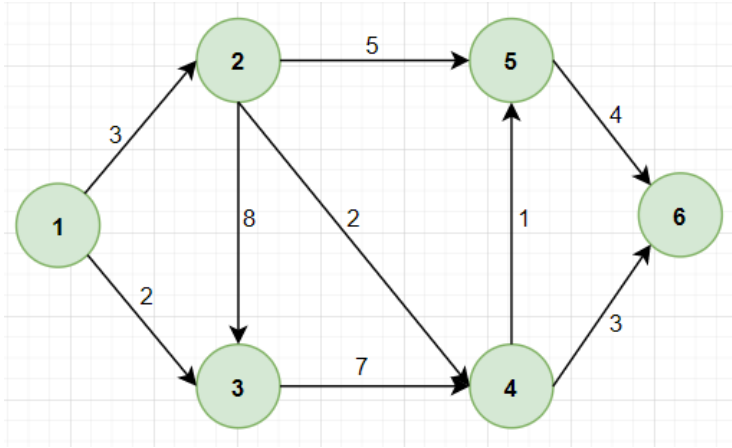
```
// thủ tục lấy đường đi từ đỉnh u tới đỉnh v
function minPath(next[][], u, v):
    if(next[u][v] = null):
        return []
    path = []: danh sách các đỉnh trên đường đi u->v
    path.append(u)
    while(u != v):
        u = next[u][v]
        path.append(u)
    return path
```

Triển khai

➢ Triển khai tìm vết đường đi:

```
/**
 * Phương thức tìm đường đi từ đỉnh u đến đỉnh v trong đồ thị.
 *
 * @param next ma trận đường đi
 * @param u    đỉnh bắt đầu
 * @param v    đỉnh đích
 * @return danh sách chứa đường đi từ u->v
 */
private static List<Vertex> minPath(Vertex[][] next, int u, int v) {
    if (next[u][v] == null) {
        return null;
    }
    ArrayList<Vertex> path = new ArrayList<>();
    path.add(next[u][u]);
    while (u != v) {
        u = next[u][v].index;
        path.add(next[u][u]);
    }
    return path;
}
```

Ví dụ



- Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6: 1 -> 2 -> 4 -> 6.
- Độ dài hành trình: 8.

The End

Cảm ơn các bạn đã đồng hành cùng Branium Academy!