Bài 11.4: Thuật toán Dijkstra

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

Khái niệm và đặc điểm

- Dijkstra là thuật toán tìm đường đi ngắn nhất giữa hai đỉnh của đồ thị có cạnh chứa trọng số dương.
- Thuật toán được phát minh bởi nhà toán học Dijkstra vào năm 1956 và xuất bản năm 1959.
- Thuật toán gốc tìm đường đi ngắn nhất giữa hai đỉnh cho trước. Các biến thể phổ biến của nó cố định đỉnh đầu và tìm đường đi ngắn nhất tới các đỉnh khác trong đồ thị.
- Thuật toán có độ phức tạp O((|V| + |E|)log(V)) khi sử dụng min heap.
- Độ phức tạp khi sử dụng Fibonacci heap là O(|E| + |V|log(V)) .
- Khi sử dụng mảng thì thuật toán có độ phức tạp O(n^2).
- Thuật toán này và các biến thể của nó được sử dụng rộng rãi trong một số lĩnh vực, đặc biệt là trí tuệ nhân tạo.

Mô tả thuật toán

Gọi node khởi đầu là s. Node đích cần tới là e. Khoảng cách giữa node s và e kí hiệu d(e).

Thuật toán Dijkstra bắt đầu với giá trị khoảng cách khởi tạo là vô cực và cập nhật dần qua từng bước:

- B1: đánh dấu tất cả các node chưa được thăm. Tạo một danh sách chứa các node chưa được thăm này gọi là unvisited.
- B2: gán giá trị khoảng cách bằng 0 cho node s. Các node khác gán bằng vô cực. Gán node khởi tạo cho node current.
- B3: với node current, tính khoảng cách đến các node lân cận chưa được thăm của nó và chọn node cho giá trị nhỏ nhất để cập nhật.
- B4: Đánh dấu current đã được thăm và loại khỏi danh sách unvisited. Một node đã thăm sẽ không được thăm lại.

Mô tả thuật toán

- B5: khi node đích e được dánh dấu là đã thăm hoặc khoảng cách giữa node current và các node chưa thăm là vô cực, kêt thúc.
- B6: ngược lại, chọn node có khoảng cách nhỏ nhất trong tập các node chưa thăm và quay lại bước 3.

Mã giả

- Sau đây là mã giả thuật toán Dijkstra:

```
// thuật toán Dijkstra
// weightMatrix: ma trận trọng số
// source: đỉnh bắt đầu
// target: đỉnh mục tiêu
// dist: mảng lưu khoảng cách từ source đến đỉnh các đỉnh vi
// prev: mảng lưu vết chứa các đỉnh trên đường đi ngắn nhất
function Dijkstra(weightMatrix[], prev[], size, source, int target):
    tạo tập đỉnh chưa thăm Q
    tạo mảng chứa trọng số dist
    // khởi tạo giá trị khoảng cách và danh sách đỉnh trên đường đi
    for(từng đỉnh v trong đồ thị G):
        dist[v] = vô cực
        prev[v] = 0
        thêm v vào Q
    dist[source] = 0
    // tiến hành lặp
    while(Q không rỗng):
        u = đỉnh trong Q có dist[u] nhỏ nhất
        if(u là đỉnh đích):
            trả về trọng số đường đi từ source->target
        xóa u khỏi Q
        for(từng đỉnh v trong Q sao cho v kề u):
            alt = dist[u] + length(u, v)
            if(alt < dist[v]):
                dist[v] = alt
                prev[v] = u;
    return -1
```

Triển khai

```
// thuật toán dijkstra
double dijkstra(int** weightMatrix, int* prev, int size, int source, int target) {
    bool* unvisited = new bool[size]();
    int* dist = new int[size]();
    for (int i = 0; i < size; i++)
    {
        dist[i] = INT_MAX;
        unvisited[i] = true;
    }
    dist[source] = 0;
    while (!empty(unvisited, size)) {
        int u = findMinWeightVertex(dist, unvisited, size);
        if (u == target) {
            return dist[u];
        }
        unvisited[u] = false;
        for (int v = 0; v < size; v++)
        {
            int weight = weightMatrix[u][v];
            if (weight != 0 && (unvisited[v] == true)) {
                int alt = dist[u] + weightMatrix[u][v];
                if (alt < dist[v]) {
                    dist[v] = alt;
                    prev[v] = u;
                }
            }
        }
    }
    return -1;
}
```

Sử dụng hàng đợi ưu tiên

➢ Mã giả thuật toán Dijkstra sử dụng hàng đợi ưu tiên:

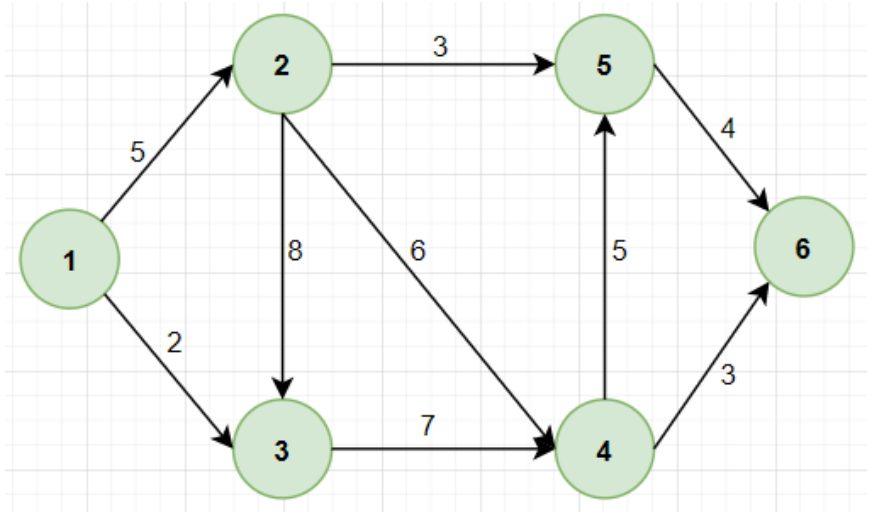
```
// thuật toán Dijkstra sử dụng hàng đợi ưu tiên
// G: đồ thị đang xét
// source: đỉnh bắt đầu
// dist: mảng lưu khoảng cách từ source đến đỉnh các đỉnh vi
// prev: mảng lưu vết chứa các đỉnh trên đường đi ngắn nhất
function Dijkstra(G, source):
    dist[source] = 0 // gán khoảng cách tại đỉnh source = 0
    tạo hàng đợi ưu tiên Q chứa các đỉnh
    // khởi tạo giá trị khoảng cách và danh sách đỉnh trên đường đi
    for(từng đỉnh v trong đồ thị G):
        if(v != source):
            dist[v] = vô cực
            prev[v] = null
        Q.add(v, dist[v]) // thêm đỉnh v và khoảng cách của nó vào Q
    // tiến hành lặp
    while(Q không rỗng):
        u = Q.extractMin() // lấy ra và xóa đỉnh có giá trị nhỏ nhất
        for(từng đỉnh v trong Q sao cho v kề u):
            alt = dist[u] + length(u, v)
            if(alt < dist[v]):
                dist[v] = alt
                prev[v] = u;
                Q.decreasePriority(v, alt)
    return dist[], prev[]
```

Sử dụng hàng đợi ưu tiên

➢ Mã thật thuật toán Dijkstra sử dụng hàng đợi ưu tiên:

```
public static Vertex[] dijkstra(Vertex[] vertices,
                                int[][] weightMatrix, int source, char target) {
    // tạo hàng đợi ưu tiên chứa tập đỉnh chưa thăm
    PriorityQueue<Vertex> unvisited = new PriorityQueue<>(vertices.length);
    Vertex[] prev = new Vertex[vertices.length]; // mảng chứa lưu vết đường đi
    vertices[source].weight = 0; // đỉnh bắt đầu duyệt sẽ có trọng số bằng 0
    for (var i = 0; i < vertices.length; i++) { // xét từng đỉnh
        if (i != source) {
            vertices[i].weight = Integer.MAX_VALUE; // khởi tạo giá trị trọng số cho đỉnh
        }
        unvisited.add(vertices[i], vertices[i].weight);
    }
    while (!unvisited.isEmpty()) { // lặp đến khi danh sách unvisited rỗng
        var u : Lesson105.Vertex = unvisited.pop().getValue(); // tìm đỉnh có trọng số nhỏ nhất
        if (u.label == target) { // nếu u là đỉnh đích
            break; // kết thúc thuật toán
        }
        for (int v = 0; v < vertices.length; v++) { // duyệt từng đỉnh của đồ thị
            // nếu cạnh (u, v) khác 0 và đỉnh v còn chưa được thăm
            if (weightMatrix[u.index][v] != 0 && unvisited.contains(vertices[v])) {
                var alt = u.weight + weightMatrix[u.index][v]; // tính trọng số thành phần
                if (alt < vertices[v].weight) { // nếu trọng số tính được nhỏ hơn thì
                    vertices[v].weight = alt; // cập nhật trọng số của đỉnh v
                    prev[v] = u; // cập nhật đỉnh trước v là u
                    unvisited.update(vertices[v], alt);
                }
            }
        }
    }
    return prev; // trả về danh sách chứa đường đi của thuật toán
}
```

Ví dụ



$$\begin{vmatrix} 0 & 5 & 2 & 0 & 0 & 0 \\ 0 & 0 & 8 & 6 & 3 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 3 \\ 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

- Đường đi ngắn nhất từ đỉnh 1 đến đỉnh 6: 1 -> 2 -> 5 -> 6.
- Độ dài hành trình: 12.

Nội dung tiếp theo

Thuật toán Bellman-Ford