Thuật toán ứng dụng

Giảng viên : Lê Quốc Trung, Đỗ Tuấn Anh
TA: Trần Thanh Tùng

Viện Công Nghệ Thông Tin và Truyền Thông
Trường Đại học Bách Khoa Hà Nội

Tháng 5, năm 2020

Mục lục

CONNECTED COMPONENTS

- Cho một đồ thị có n đỉnh và m cạnh 2 chiều
- Tính số lượng thành phần liên thông của đồ thị

Giải thuật duyệt DFS

- Lần lượt duyệt qua các đỉnh của đồ thị
- Mỗi đỉnh duyệt qua tất cả các đỉnh liên thông với đỉnh đó bằng phương pháp đệ quy
- Kiểm tra một đỉnh đã được duyệt qua chưa bằng cách đánh dấu

Code

```
void dfs(int u) {
    visit[u] = 1;
    for (int v : a[u]) {
        if (visit[v]) {
            continue;
        }
        dfs(v);
    }
}
```

Giải thuật duyệt BFS

Giống với giải thuật DFS tuy nhiên thay vì đệ quy thì ta duyệt đỉnh bằng queue

Code

```
void bfs(int root) {
    int head = 0, tail = 1;
    vertex_queue[++head] = root;
    visit[root] = 1;
    while (tail <= head) {
        int u = vertex_queue[tail]; tail++;
        for (int v : a[u]) {
            if (visit[v]) {
                continue;
            }
            vertex_queue[++head] = v;
            visit[v] = 1;
        }
    }
}
```

SHORTEST PATH

- Cho một đồ thị có hướng n đỉnh, m cạnh
- Tìm đường đi ngắn nhất từ đỉnh s tới đỉnh t

Thuật toán 1

- Sử dụng thuật toán Dijkstra.
- Mỗi lần lấy ra đỉnh có đường đi ngắn nhất rồi update độ dài đường đi các đỉnh kề với đỉnh đó.
- Sử dụng mảng đánh dấu để không xét lại một đỉnh 2 lần.
- Độ phức tạp $O(n^2)$.

Code

```
int find_shortest_path(int start, int des,
                       vector < vector < Edge > > a) {
    vector < long long > d(n + 1, 0);
    vector < bool > visit(n + 1, 0);
    for (int i = 0; i <= n; i++) {
        d[i] = MAX;
    }
    d[start] = 0;
    int step = n;
    while (step--) {
        int min_vertex = 0;
        for (int i = 1; i <= n; i++) {
            if (d[min_vertex] > d[i] && visit[i] == 0) {
                min_vertex = i;
            }
        }
        visit[min_vertex] = 1;

        for (Edge e : a[min_vertex]) {
            int v = e.v;
            int w = e.w;
            d[v] = min(d[v], d[min_vertex] + w);
        }
    }
    return d[des];
}
```

Thuật toán 2

- Sử dụng heap để cải tiến từ thuật toán 1.
- Với mỗi khi có đỉnh được update độ dài đường đi thì ta sẽ đưa đỉnh đó vào trong heap
- Độ phức tạp : $O((n+m)\log(m))$

Code

```
int find_shortest_path(int start, int des,
                       vector < vector < Edge > > a) {
    vector < long long > d(n + 1, 0);
    for (int i = 0; i <= n; i++) {
        d[i] = MAX;
    }
    d[start] = 0;
    priority_queue < pair < long long, int > > vertex_queue;
    vertex_queue.push({-0, start});
    while (!vertex_queue.empty()) {
        pair < long long, int > p = vertex_queue.top(); vertex_queue.pop();
        long long distance = -p.first;
        int min_vertex = p.second;
        if (d[min_vertex] < distance) { continue; }
        for (Edge e : a[min_vertex]) {
            int v = e.v;
            int w = e.w;
            if (d[v] > d[min_vertex] + w) {
                d[v] = d[min_vertex] + w;
                vertex_queue.push({-d[v], v});
            }
        }
    }
    return d[des];
}
```

ICBUS

- Cho n thị trấn được đánh số từ 1 tới n.
- Có k con đường hai chiều nối giữa các thị trấn.
- Ở thị trấn thứ i sẽ có một tuyến bus với giá vé là c_i và đi được quãng đường tối đa là d_i.
- Tìm chi phí tối thiểu để đi từ thị trấn 1 tới thị trấn n.

Thuật toán

- **Bước 1 :** Tính khoảng cách di chuyển ngắn nhất của tất cả các cặp đỉnh u, v bằng thuật toán BFS. Lưu vào mảng $dist[u][v]$
- **Bước 2 :** Tạo một đồ thị mới một chiều trong đó đỉnh u được nối tới đỉnh v khi $dist[u][v] <= d[u]$ và cạnh này có trọng số là $c[u]$
- **Bước 3 :** Tìm đường đi ngắn nhất từ 1 tới n trên đồ thị mới được tạo ra bằng thuật toán Dijkstra.
- Độ phức tạp thuật toán $O(n^2)$

Code

```
void calculate_dist() {
    ** Calculate dist[u][v] using BFS algorithm **
}
void find_shortest_path() {
    for (int i = 0; i <= n; i++) {
        ans[i] = MAX;
        visit[i] = 0;
    }
    ans[1] = 0;
    int step = n;
    while (step--) {
        int min_vertex = 0;
        for (int i = 1; i <= n; i++) {
            if (visit[i] == 0 && ans[min_vertex] > ans[i]) {
                min_vertex = i;
            }
        }
        visit[min_vertex] = 1;
        for (int i = 1; i <= n; i++) {
            if (dist[min_vertex][i] <= d[min_vertex]) {
                ans[i] = min(ans[i], ans[min_vertex] + c[min_vertex]);
            }
        }
    }
    cout << ans[n] << endl;
}
```

ARTICULATION POINTS AND BRIDGES

- Cho đồ thị vô hướng n đỉnh và m cạnh.
- Tìm số lượng khớp (articulation point) và cầu (bridge) của đồ thị.

Naive approach $O(n * m)$

- Duyệt qua tất cả các đỉnh của đồ thị, với mỗi đỉnh kiểm tra xem có phải là khớp của đồ thị hay không
- Sử dụng thuật BFS hoặc DFS để kiểm tra
- Độ phức tạp $O(n * m)$

Thuật toán Tarjan (https ://codeforces.com/blog/entry/71146)

- Duyệt cây theo thứ tự DFS, ta liệt kê được thứ tự các đỉnh được duyệt, ta gọi là cây DFS.
- Với mỗi đỉnh ta cần tính 2 mảng
  - **num[u]** : Thời gian đếm thăm đỉnh u theo thứ tự duyệt DFS.
  - **low[u]** : Thời gian nhỏ nhất mà đỉnh u và các đỉnh con của u có thể đến thăm được trong cây DFS.
- Khớp là các đỉnh u mà có $num[u] \leq low[v]$ trong đó v là con trực tiếp của u trong cây DFS.
- Đối với đỉnh root của cây DFS thì phải có thêm điều kiện có ít nhất 2 con trong cây DFS.
- Độ phức tạp : $O(n + m)$.

Code

```
int dfs(int u, int par) {
    int num_childs = 0;
    low[u] = num[u] = ++cnt;
    visit[u] = 1;
    for (int v : a[u]) {
        if (v == par) { continue; }
        if (visit[v] == 0) {
            num_childs++;
            dfs(v, u);
            /*if (low[v] > num[u]) {
                bridges++;
            }*/
            ap[u] |= low[v] >= num[u];
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], num[v]);
        }
    }
    return num_childs;
}

for (int i = 1; i <= n; i++) {
    if (visit[i] == 0) {
        ap[i] = dfs(i, i) >= 2;
    }
}
```

Thuật toán tìm cầu

- Ta cũng sử dụng thuật toán Tarjan.
- Cầu là các cạnh u, v sao cho u là cha của v trong cây DFS và $low[v] > num[u]$

MINIMUM SPANNING TREE

- Cho một đồ thị liên thông vô hướng n đỉnh, m cạnh, mỗi cạnh có một trọng số.
- Tìm cây khung nhỏ nhất của đồ thị

Thuật toán

- Sử dụng thuật toán Prim.
- Ta luôn maintain mảng khoảng cách ngắn nhất đến tập đỉnh trong cây khung hiện tại.
- Sử dụng cấu trúc heap để lưu lại phần tử có khoảng cách ngắn nhất.
- Độ phức tạp $O(n + m)log(m)$

Code

```
long long mimimum_spanning_tree(vector < vector < Edge > > a) {
    vector < int > d(n + 1, 0);
    vector < bool > visit(n + 1, 0);
    int root = 1;
    for (int i = 1; i <= n; i++) { d[i] = MAX; }
    d[root] = 0;
    priority_queue < pair < int, int > > vertex_queue;
    vertex_queue.push({-d[root], root});
    long long ans = 0;
    while (!vertex_queue.empty()) {
        pair < int, int > p = vertex_queue.top(); vertex_queue.pop();
        int distance = -p.first;
        int min_vertex = p.second;
        if (visit[min_vertex]) { continue; }
        visit[min_vertex] = 1;
        ans += distance;
        for (Edge e : a[min_vertex]) {
            int v = e.v; int w = e.w;
            if (d[v] > w) {
                d[v] = w;
                vertex_queue.push({-d[v], v});
            }
        }
    }
    return ans;
}
```