Thuật toán ứng dụng
Bài thực hành số 5

Giảng viên: TS. Đinh Viết Sang
Trợ giảng: Nguyễn Trung Hiếu

Viện Công nghệ thông tin & truyền thông
Đại học Bách khoa Hà Nội

05/2021

Nội dung

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

```
1  void dfs(int u) {
2      visit[u] = 1;
3      for (int i = 0; i < a[u].size(); i++) {
4          int v = a[u][i];
5          if (!visit[v]) {
6              dfs(v);
7          }
8      }
9  }
```

Giải thuật duyệt BFS

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

```
1  void bfs(int root) {
2      int head = 0, tail = 1;
3      vertex_queue[++head] = root;
4      visit[root] = 1;
5      while (tail <= head) {
6          int u = vertex_queue[tail]; tail++;
7          for (int v : a[u]) {
8              if (visit[v]) {
9                  continue;
10             }
11             vertex_queue[++head] = v;
12             visit[v] = 1;
13         }
14     }
15 }
```

06. BUGLIFE

- Cho n con bọ, các con bọ này được đánh số từ 1 đến n.
- Các con bọ chỉ tương tác với các con bọ có giới tính khác nó.
- Cho danh sách các sự tương tác giữa các con bọ.
- **Yêu cầu:** Liệu có con bọ nào tương tác với cả 2 giới tính không?

Thuật toán

- Áp dụng thuật toán DFS, coi các con bọ là đỉnh, các tương tác là cạnh.
- Với đỉnh u, ta đánh dấu nó là x còn với các đỉnh kề với u, ta đánh dấu là $-x$.
- Sau khi chạy DFS, nếu có 2 đỉnh kề nhau mà có cùng đánh dấu thì tức là tồn tại một con bọ đáng nghi.

Code

```
1  void DFS(int u)
2  {
3    for(int i = 0; i < a[u].size(); i++) {
4      int v = a[u][i];
5      if(!mark[v]) {
6        mark[v] = -mark[u];
7        DFS(v);
8      }
9    }
10 }
```

Code

```
12  for(int i = 1; i <= n; i++) {
13        if(!mark[i]) {
14          mark[i] = 1;
15          DFS(i);
16        }
17      }
18
19      bool ans = true;
20      // true -> No suspicious bugs found
21      // false -> Suspicious bugs found
22      for(int i = 1; i <= n; i++) {
23        for(int j = 0; j < a[i].size(); j++) {
24          if(mark[i] == mark[a[i][j]]) {
25            ans = false;
26          }
27        }
28      }
```

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 < 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

```
10 int find_shortest_path(int start, int des) {
11     vector < long long > d(n + 1, 0);
12     for (int i = 0; i <= n; i++) {
13         d[i] = MAX;
14     }
15     d[start] = 0;
16     priority_queue < pair < long long, int > > vertex_queue;
17     vertex_queue.push({-0, start});
18     while (!vertex_queue.empty()) {
19         pair < long long, int > p = vertex_queue.top();
20         vertex_queue.pop();
21         long long distance = -p.first;
22         int min_vertex = p.second;
23         if (d[min_vertex] < distance) { continue; }
24         for (Edge e : a[min_vertex]) {
25             int v = e.v;
26             int w = e.w;
27             if (d[v] > d[min_vertex] + w) {
28                 d[v] = d[min_vertex] + w;
29                 vertex_queue.push({-d[v], v});
30             }
31         }
32     }
33     return d[des];
34 }
```

06. 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

```
1  void calculate_dist() {
2      ** Calculate dist[u][v] using BFS algorithm **
3  }
4  void find_shortest_path() {
5      for (int i = 0; i <= n; i++) {
6          ans[i] = MAX;
7          visit[i] = 0;
8      }
9      ans[1] = 0;
10     int step = n;
11     while (step--) {
12         int min_vertex = 0;
13         for (int i = 1; i <= n; i++) {
14             if(!visit[i] && ans[min_vertex] > ans[i])
15                 min_vertex = i;
16             }
17         }
```

Code

```
18              visit[min_vertex] = 1;
19              for (int i = 1; i <= n; i++) {
20                  if(dist[min_vertex][i] <= d[min_vertex]){
21                      ans[i] = min(ans[i], ans[min_vertex]
22                      + c[min_vertex]);
23                  }
24              }
25          }
26          cout << ans[n] << endl;
27      }
```