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      }
```

06. ADDEDGE

- Cho đồ thị vô hướng n đỉnh, m cạnh.
- Tính số cạnh thêm vào để đồ thị thêm đúng một chu trình đơn giản.
  - Chu trình đơn giản: là một chu trình đi qua mỗi đỉnh đúng một lần.

Nhận xét

- Một cạnh e_1, nếu đang nằm trong một chu trình có sẵn nào đó thì sẽ không thể nằm trong bất kỳ chu trình đơn giản mới nào.
  - Vì nếu cạnh e_1 nằm trong chu trình đơn giản mới đồng thời nằm trong chu trình hiện tại, cạnh e^* thêm vào sẽ tạo ra 2 chu trình đơn giản mới.
- $\rightarrow$ cạnh e_1 là không cần thiết (vì chắc chắn sẽ không nằm trên chu trình mới nào.

Thuật toán

- Xóa toàn bộ các cạnh nằm trên bất kỳ một chu trình nào của đồ thị $\rightarrow$ ta được một rừng các cây.
- Bài toán quy thành tính số lượng cạnh mới thêm vào để được một chu trình đơn trên cây.
- Lời giải cho bài toán trên là: $n * (n-1)/2 - (n-1)$ trong đó n là số đỉnh của cây (mọi cặp đỉnh trừ đi số cạnh của cây).
- Để xóa toàn bộ các cạnh nằm trên bất kỳ một chu trình nào của đồ thị.
  - Các cạnh không bị xóa là cầu của đồ thị gốc $\rightarrow$ tìm tất cả các cầu của đồ thị.

Code - tìm cầu

```
10 void Tarjan(int u, int p) {
11   num[u] = low[u] = ++cnt;
12   for (int i = 0; i < adj[u].size(); i++) {
13     int v = adj[u][i].first;
14     int w = adj[u][i].second;
15     if (v == p) continue;
16     if (num[v]) low[u] = min(low[u], num[v]);
17     else {
18       Tarjan(v, u);
19       low[u] = min(low[u], low[v]);
20       if (low[v] > num[u]) is_bridge[w] = 1;
21     }
22   }
23 }
```

Code - DFS

```
24 int Dfs(int u) {
25   used[u] = 1;
26   int res = 1;
27   for (int i = 0; i < adj[u].size(); i++) {
28     int v = adj[u][i].first;
29     int w = adj[u][i].second;
30     if (!is_bridge[w]) continue;
31     if (!used[v]) res += Dfs(v);
32   }
33   return res;
34 }
```

Code - DFS

```
35 for (int i = 1; i <= m; i++) {
36     int u, v;
37     cin >> u >> v;
38     adj[u].push_back(make_pair(v, i));
39     adj[v].push_back(make_pair(u, i));
40 }
41 for (int i = 1; i <= n; i++)
42     if (!num[i]) Tarjan(i, 0);
43 long long res = 0;
44 for (int i = 1; i <= n; i++) {
45     if (!used[i]) {
46       int c = Dfs(i);
47       res += 1ll * c * (c - 1) / 2 - (c - 1);
48     }
49 }
50 cout << res;
```

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      }
```