Компоненты сильной связности: алгоритм и доказательство

1. Определения

Дан ориентированный граф (орграф) из n вершин и m рёбер. К орграфу не применимо понятие "связности", но применимо понятие "сильной связности":

Def: *Компонента сильный связности* – максимальное по включению множество вершин орграфа, попарно достижимых друг из друга.

Сокращение: *scc* – strong connectivity components.

Следствие: чтобы найти компоненту сильной связности, содержащую вершину v, нужно взять достижимые из v по прямым рёбрам вершины A, достижимые из v по обратным рёбрам вершины B, и вернуть $A \cap B$.

Алгоритм за $\mathcal{O}(nm)$: перебираем вершины в произвольном порядке, видим вершину v, не лежащую в уже найденных компонентах сильной связности, находим за $\mathcal{O}(m)$ компоненту вершины v, продолжаем.

2. Алгоритм за $\mathcal{O}(m)$

Будем перебирать вершины графа в порядке топологической сортировки. Из каждой не помеченной вершины будем запускать dfs по обратном рёбрам, свеженайденные вершины – новая компонента сильной связности.

```
const int N = 1e6; // максимальный размер графа
struct Graph {
  int n; // количество вершин в графе
  vector<int> c[N], rc[N]; // прямые и обратные рёбра

  vector<int> marked;
  deque<int> topsort;
  void get_topsort( int v ) {
    marked[v] = 1;
    for (int x : c[v]) // прямые рёбра!
      if (!marked[x])
        get_topsort(x);
    topsort.push_front(v); // подсовываем v в начало topsort
  }
  void get_component( int v, vector<int> &component ) {
    marked[v] = 1;
    component.push_back(v);
    for (int x : rc[v]) // обратные рёбра!
      if (!marked[x])
        get_component(x, component);
  }

  void add_edge( int a, int b ) { // все рёбра нужно добавлять так
    c[a].push_back(b);
    rc[b].push_back(a);
  }
  vector<vector<int>> solve() {
    topsort.clear();
    marked = vector<int>(n, 0); // все вершины не помечены
    for (int v = 0; v < n; v++)
```

```
31        if (!marked[v])
32          get_topsort(v);
33      marked = vector<int>(n, 0);
34      vector<vector<int>> components;
35      for (int v : topsort)
36        if (!marked[v]) {
37          vector<int> component;
38          get_component(v, component);
39          components.push_back(component);
40        }
41      return components;
42    }
43  };
```

Алгоритм состоит из двух поисков в глубину – по прямым и по обратным рёбрам. Поэтому так же, как и поиск в глубину, работает за $\mathcal{O}(n+m)$.

3. Корректность

Обозначим за $A(v)$ компоненту сильной связности вершины v.
Функция `get_component(v, component)` включает в `component` все вершины, достижимые по обратным рёбрам, поэтому $A(v) \subseteq$ `component`. Пусть есть путь из v в x по обратным рёбрам и $x \notin A(v)$. Докажем, что компоненту $A(x)$ мы к этому моменту уже нашли, поэтому x уже помечена и не войдёт в `component`. Для этого в `topsort` хотя бы одна вершина из $A(x)$ должна лежать раньше всего множества $A(v)$. Рассмотрим первый момент, когда мы в `get_topsort` вошли в $A(x)$, пусть это вершина $y \in A(x)$. В этот момент $A(v)$ или целиком помечена, или целиком не помечена. Если не помечена, мы сперва обойдём всю $A(v)$, и только затем выйдем из y, положим её в `topsort`. Поэтому y в `topsort` точно раньше всего $A(v)$. ■

4. Ссылки

Существует алгоритм Тарьяна, использующий лишь один dfs для поиска компонент сильной связности. И идейно и реализационно он очень похож на алгоритмы поиска мостов, точек сочленения. Тема сильной связности раскрыта

1. В ИТМО-конспектах.
2. На emax.
3. В английской википедии.