Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра исследования операций

Большакова Елена Андреевна

Свойство бисвязности ориентированного графа

Дипломная работа

Научный руководитель:
Доктор физ.-мат. наук, профессор Романовский Иосиф Владимирович

Санкт-Петербург
2016

Saint-Petersburg State University
The Faculty of Mathematics and Mechanics
Subdepartment of Operations Research

Bolshakova Elena Andreevna

Property of bicconnectivity of a directed graph

Graduation Thesis

Scientific Supervisor
Doctor of Physics and Mathematics, professor I.V. Romanovskiy

Saint Petersburg
2016

Содержание

Введение

В ряде приложений важно уметь выделить в графе компоненты сильной связности. Мне было поручено сделать обзор имеющихся алгоритмов анализа связности ориентированного графов.

В этом анализе большую роль играют понятия «шарниры» и «компоненты сильной связности». Грубо говоря, шарниры – это вершины, такие, что разделение графа в них делит граф на сильно-связные компоненты.

Мне нужно было выяснить, какие сейчас приняты методы выделения в графе сильно-связных компонент.

Оказалось, есть три известных популярных алгоритма, которые и описываются в этой работе. К сожалению, конкурировать с этими алгоритмами я не смогла.

Основное внимание в работе уделено алгоритму Косарайю.

Графо-теоретические понятия

Граф – совокупность непустого множества вершин и наборов пар вершин (связей между вершинами)

Ориентированный граф – (сокращённо *орграф*) G - это упорядоченная пара

$$G := (V, E)$$

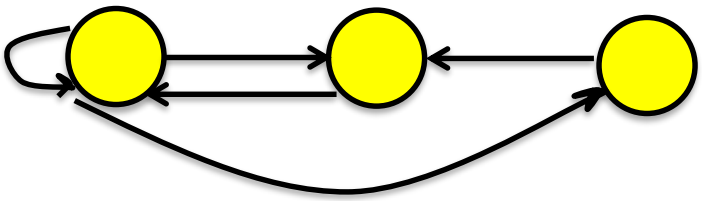
где V —множество вершин

E — множество (упорядоченных) пар различных вершин, называемых дугами или ориентированными рёбрами.

Дуга — это упорядоченная пара вершин (v, w), где вершину v называют началом, а w — концом дуги.

Можно сказать, что дуга $v \to w$ ведёт от вершины v к вершине w.

Другими словами, орграф – это такой граф, все ребра которого имеют направление.



Орграф, полученный из простого графа ориентацией ребер, называется *направленным*.

Связность

Рассмотрим сначала неориентированный граф.

Две вершины *u, v* называются *связными*, если найдется соединяющая их цепь (путь). Каждая вершина связана с собой.

Компонента связности – максимальный связной подграф графа.

Несвязный граф имеет по крайней мере две компоненты связности.

Связный граф – граф, у которого любая пара вершин связана.

Другими словами, связный граф содержит ровно одну компоненту связности.

Каждая компонента связности представляет собой связный граф.

Если в n-вершинном графе количество ребер равно $m > \frac{(n-1)(n-2)}{2}$, то этот граф связен.

Шарнир (точка сочленения/разделяющая вершина) – вершина *v*, если ее удаление увеличивает число компонент связности.

В дереве все неконцевые вершины являются точками сочленения.

В ориентированных графах различают несколько понятий связности.

Ориентированный граф называется *слабо-связным*, если является связным неориентированный граф, полученный из него заменой ориентированных рёбер неориентированными.

Ориентированный граф называется *сильно-связным*, если в нём существует (ориентированный) путь из любой вершины в любую другую, или, что эквивалентно, граф содержит ровно одну сильно связную компоненту.

Две вершины *s* и *t* любого графа сильно связны, если существует ориентированный путь из *s* в *t* и ориентированный путь из *t* в *s*.

Компонентами сильной связности орграфа называются его максимальные по включению сильно связные подграфы.

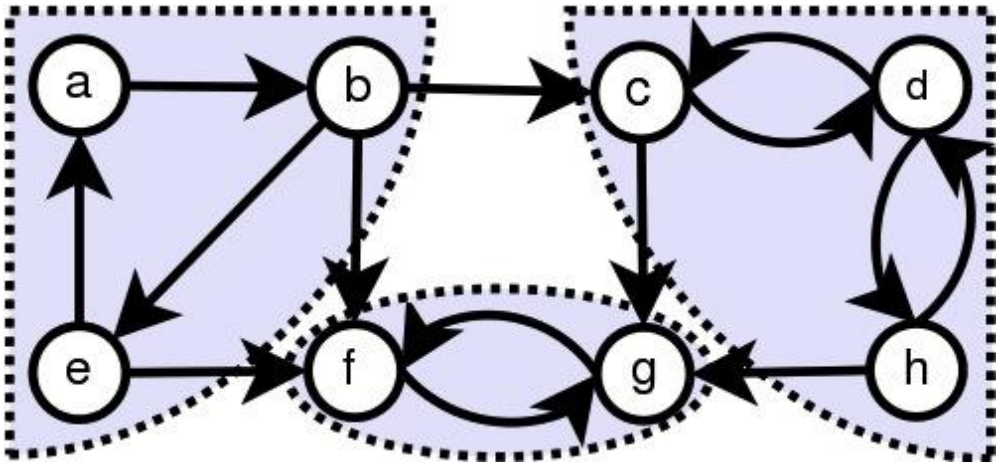
Любая вершина орграфа сильно связна сама с собой.

Простейший алгоритм решения задачи о поиске сильно связных компонент в орграфе:

1. Проверяем, достижима ли t из s, и s из t, для всех пар s и t выполняя транзитивное замыкание.
2. Определяем такой неориентированный граф, в котором для каждой такой пары содержится ребро.
3. Поиск компонент связности такого неориентированного графа даст компоненты сильной связности нашего орграфа.

Сейчас известны три алгоритма, решающих данную задачу за линейное время, то есть в *V* раз быстрее, чем приведенный выше алгоритм. Это алгоритмы Косарайю, Габова и Тарьяна.

Пример орграфа, для которого найдены все три компоненты сильной связности



Если *v* и *w* являются некоторыми вершинами , для которых существует определенная ориентированная цепь *P(v,w)*, идущая от *v* к *w,* то *вершина w достижима из v, v>w.*

Если *w* достижима из *v* и *s* достижима из *w*, то *s* достижима из *v .*

Обозначим через *D(v)* множество всех вершин, как достижимых, так и обратно достижимых из *v*.

Две вершины *v* и *w* называются *эквивалентными по достижимости*, когда *D(v)=D(w),* то есть если *v>w, w>v* или же *v ~w.*

Для выполнения этого необходимо и достаточно, чтобы вершины v и w были в G взаимно связаны (сильно связаны, бисвязны) , другими словами, чтобы в G были определенные цепи P(v,w) и Q(w,v).

Две вершины *эквивалентны по достижимости* тогда и только тогда, когда они взаимно связаны.

Поиск в глубину

Поиск в глубину (DFS – depth first search) – метод обхода графа.

Поиск в глубину представляет особый метод, который является одним из основных алгоритмов на графах. Этот метод служит для решения многих задач, таких как задача связанности, а также множества других задач, связанных с обработкой графов. Данный метод очень значим, так как именно он послужил основой для разработки ряда других быстрых алгоритмов.

Суть данного метода, заключается в том, чтобы двигаться “вглубь” до тех пор, пока это возможно, то есть до тех пор, пока у нас есть непройденные исходящие ребра. Обход вершин графа происходит по определенному закону, принципу: если из текущей вершины есть ребра, ведущие в непройденные вершины, то идем туда, иначе возвращаемся назад.

Поиск в глубину имеет определенный принцип действий, алгоритм.

Для начала должна быть выбрана начальная вершина *v* графа G, которая, в свою очередь, автоматически помечается и определяется как пройденная.

На втором шаге рекурсивно вызывается поиск в глубину для каждой непомеченной вершины, которая смежна с *v*.

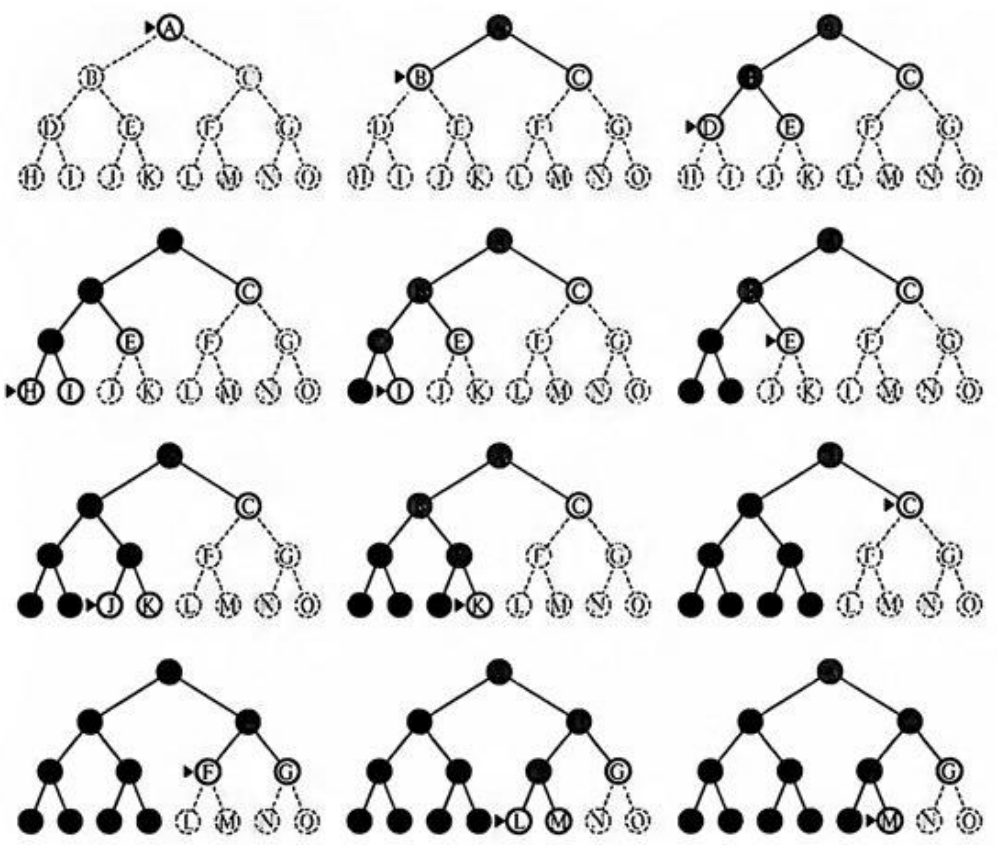
И продолжается этот поиск до тех пор, пока все вершины, достижимые из *v*, не будут помечены.

В случае, если на каком-либо (не начальном) шаге обхода поиск закончился, но, тем не менее, некоторые из вершин так и остались непомеченными, то должна быть выбрана произвольная непомеченная вершина и поиск должен быть повторен.

Процесс поиска должен продолжаться и быть закончен только тогда, когда все вершины графа будут помечены.

При данном методе целесообразно использовать цвет вершины.

Каждая непомеченная вершина изначально имеет белый цвет. Когда вершину обнаруживают в первый раз, ее цвет меняется на серый. Вершина является обработанной тогда и только тогда, когда просмотрены все ее смежные вершины. Когда вершина обработана, она меняет цвет на черный.



Каждой вершине будем устанавливать две метки времени:

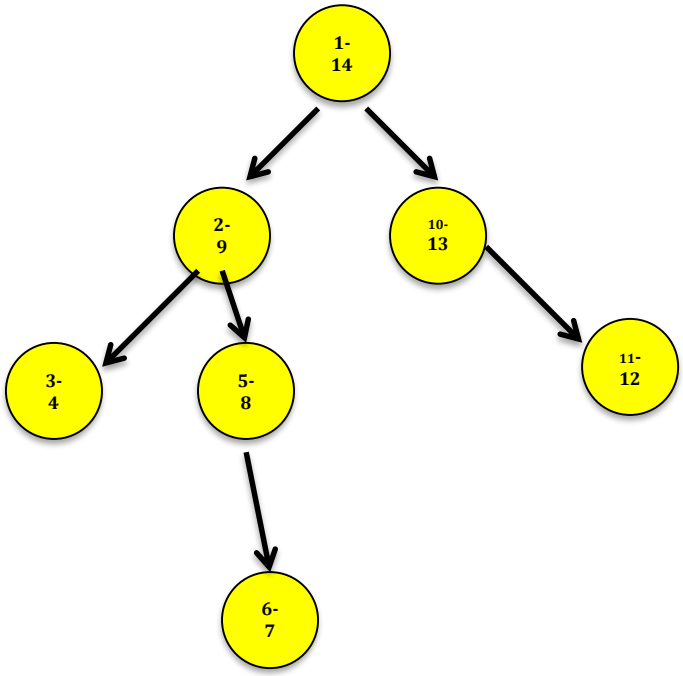
$In(v)$– время, когда вершина обнаружена и цвет изменен на серый;

$Out(v)$ – время, когда вершина обработана и цвет изменен на черный.

Эти метки в дальнейшем будут использоваться в алгоритмах для поиска и выявления компонент сильной связности.

Каждая вершина v соответствует неравенству : $In(v) < Out(v)$

Вершина v будет иметь белый цвет до момента времени $In(v)$, серый цвет с $In(v)$ до $Out(v)$ и станет черной после $Out(v)$.



Классификация ребер

Поиск в глубину используется для классификации ребер графа. Существуют 4 типа ребер:

1. *Дуги дерева* - ребра, которые ведут к вершинам и раньше не посещались. Они осуществляют функцию формирования глубинного остовного леса для заданного графа.
2. *Обратные дуги* -дуги, которые в остовном лесу идут от потомков к предкам. Дуга, которая идет из вершины в саму себя, считается обратной.
3. *Прямыми дуги*- дуги, которые идут от предков к собственным потомкам, но, тем не менее, не являются дугами дерева.
4. *Поперечные дуги*- ребра, которые соединяют вершины, которые не являются предками или потомками.

Ребро (u, v) является

а) *ребром дерева* или *прямым ребром* тогда и только тогда, когда выполняется неравенство

$$In(u) < In(v) < Out(v) < Out(u)$$

б) *обратным ребром* тогда и только тогда, когда выполняется неравенство

$$In(v) < In(u) < Out(u) < Out(v)$$

в) *поперечным ребром* тогда и только тогда, когда выполняется неравенство

$$In(v) < Out(v) < In(u) < Out(u)$$

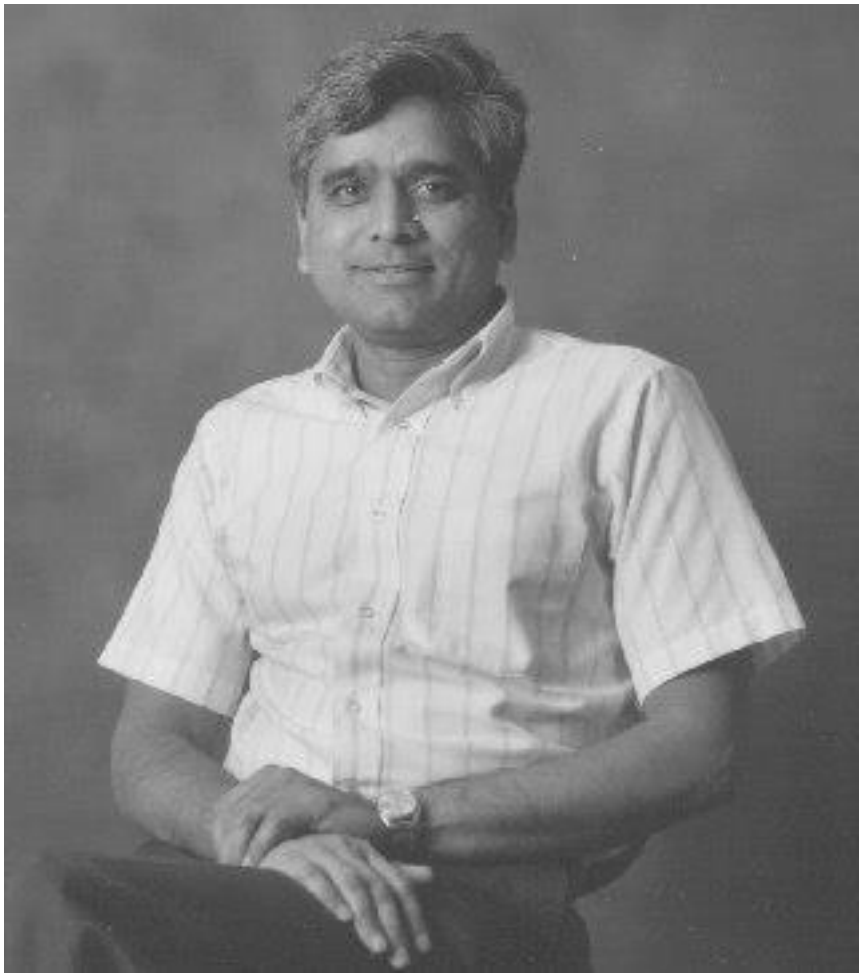
Алгоритм работает за $O(|V| + |E|)$

Ниже приведено описание функции алгоритма поиска в глубину.

```
void Depth_First_Search(int n, int **Graph, bool *Visited,
                        int Node){
  Visited[Node] = true;
  cout << Node + 1 << endl;
  for (int i = 0 ; i < n ; i++)
    if (Graph[Node][i] && !Visited[i])
      Depth_First_Search(n,Graph,Visited,i);
}
```

Алгоритм Косарайю.

Самбасив Рао Косарайю – американский ученый с индийскими корнями, профессор информатики в Университете Джона Хопкинса. В 1978 написал работу в которой описал такой метод, как метод эффективного вычисления компонент сильной связности в орграфе. Позднее данный метод стал известен как «Алгоритм Косарайю», название закрепилось. В 1983 году алгоритм был опубликован в статье А. Ахо, Д. Хопкрофта «Структуры данных и алгоритмы» (Data Structures and Algorithms).



Данный алгоритм заключается в том, что он выявляет сильные компоненты графа $G = (V, E)$, за определенное время $O(|V| + |E|)$.

Его сложность связана со сложностью алгоритма поиска в глубину, который, в свою очередь, должен быть использован дважды, а также со сложностью нахождения обратного графа. Оба эти алгоритма имеют линейную сложность и именно поэтому Алгоритм Косарайю работает за линейное время.

Данный алгоритм опирается на метод поиска в глубину.

Для начала выполняется определенный поиск в глубину для обратного графа, или другими словами, на обращении исходного графа (то есть графа, который может получиться при инвертировании ребер исходного, начального графа.
Далее идет вычисление вектора обратного порядка обхода.

На втором шаге происходит поиск в глубину на исходном графе, причем надо заметить, что вершины берутся в том порядке, который является обратным тому, который, в свою очередь, получился в следствие нумерации вершин при обратном проходе при первом запуске поиска в глубину (по полученному вектору обхода). Когда выполнятся метод поиска в глубину, то используются непосещенные вершины, которые имеют максимальный номер.

Если выполнить весь алгоритм, то в конечном итоге получится лес, у которого деревья будут представлять сильные компоненты связности графа.

Ниже приведен пример из книги Роберта Седжвика «Фундаментальные алгоритмы на C++ - Часть 5 – Алгоритмы на графах».

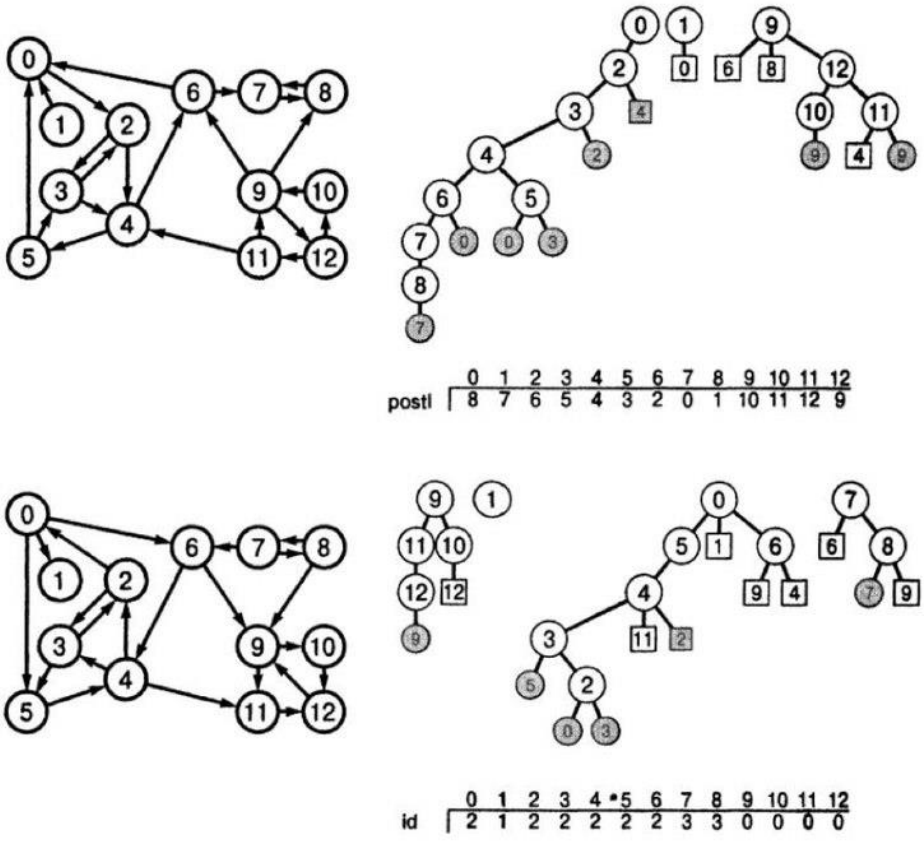


РИСУНОК 19.28. ВЫЧИСЛЕНИЕ СИЛЬНЫХ КОМПОНЕНТ (АЛГОРИТМ КОСАРАЙЮ)

Чтобы вычислить сильные компоненты орграфа, изображенного внизу слева, мы сначала выполняем поиск в глубину на его обращении (вверху слева), вычисляя вектор обратного порядка обхода, который присваивает вершинам индексы в порядке, в каком завершаются рекурсивные DFS (сверху). Этот порядок эквивалентен обратному порядку обхода леса DFS (вверху справа). Затем мы используем обращение этого порядка, чтобы выполнить поиск в глубину на исходном графе (внизу). Сначала мы проверяем все узлы, доступные из вершины 9, затем осуществляем просмотр этого вектора справа налево, обнаруживая при этом, что 1 есть крайняя справа непосещенная вершина, поэтому мы выполняем рекурсивный вызов для вершины 1 и т.д. Деревья в лесе DFS, которые выбираются в результате этого процесса, представляют собой сильные компоненты: все вершины каждого дерева имеют те же значения, что и вектор id, индексированный именами вершин (внизу).

Программа реализация алгоритма Косарайю на C++

```
template <class Graph>
class SC
  { const Graph &G;
    int cnt, scnt;
    vector<int> postI, postR, id;
    void dfsR(const Graph &G, int w)
      { id[w] = scnt;
        typename Graph::adjIterator A(G, w);
        for (int t = A.beg(); !A.end(); t = A.nxt())
          if (id[t] == -1) dfsR(G, t);
        postI[cnt++] = w;
      }
  public:
    SC(const Graph &G) : G(G), cnt(0), scnt(0),
      postI(G.V()), postR(G.V()), id(G.V(), -1)
      { Graph R(G.V(), true);
        reverse(G, R);
        for (int v = 0; v < R.V(); v++)
          if (id[v] == -1) dfsR(R, v);
        postR = postI; cnt = scnt = 0;
        id.assign(G.V(), -1);
        for (int v = G.V()-1; v >= 0; v--)
          if (id[postR[v]] == -1)
            { dfsR(G, postR[v]); scnt++; }
      }
    int count() const { return scnt; }
    bool stronglyreachable(int v, int w) const
      { return id[v] == id[w]; }
  };
```

Реализация алгоритмов Габова и Тарьяна

Алгоритмы та же находят сильные компоненты орграфа за линейное время.

```
#include "STACK.cc"
template <class Graph>
class SC
  { const Graph &G;
    STACK<int> S;
    int cnt, scnt;
    vector<int> pre, low, id;
    void scR(int w)
      { int t;
        int min = low[w] = pre[w] = cnt+ + ;
        S.push(w);
        typename Graph::adjIterator A(G, w);
        for (t = A.beg(); !A.end(); t = A.nxt())
          { if (pre[t] == -1) scR(t);
            if (low[t] < min) min = low[t];
          }
        if (min < low[w]) { low[w] = min; return; }
        do
          { id[t = S.pop()] = scnt; low[t] = G.V(); }
        while ( t ! = w) ;
        scnt++;
      }
  public:
    SC(const Graph &G) : G(G), cnt(0), scnt(0),
      pre(G.V(), -1), low(G.V()), id(G.V())
      { for (int v = 0; v < G.V(); v++)
          if (pre[v] == -1) scR(v);
      }
    int count() const { return scnt; }
    bool stronglyreachable(int v, int w) const
      { return id[v] == id[w]; }
  };
```

Алгоритм Габова

```
void scR(int w)
  { int v;
    pre[w] = cnt++;
    S.push(w); path.push(w);
    typename Graph::adjIterator A(G, w);
    for (int t = A.beg(); !A.end(); t = A.nxt())
      if (pre[t] == -1)
        scR(t);
      else if (id[t] == -1)
        while (pre[path.top()] > pre[t]) path.pop();
    if (path.top() == w) path.pop(); else return;
    do { id[v = S.pop()] = scnt; } while (v != w);
    scnt++;
  }
```

Заключение

Таким образом рассмотрели алгоритмы нахождения компонент сильной связности орграфа.

В литературе есть программы этих методов, построенных на популярном методе в глубину.

Алгоритмы настолько маленькие, что их оказалось возможным привести полностью.

Список литературы

[1] Роберт Седжвик. Алгоритмы на графах = Graph algorithms. — 3-е изд. — Россия, Санкт-Петербург: «ДиаСофтЮП», 2002. — 496 с

[2] С. Дасгупта, Х. Пападимитриу, У. Вазирани. Алгоритмы — М.: МЦНМО, 2014. — 320 с

[3] Богомолов А.М. Алгебраические основы теории дискретных систем.— М.: Наука, 1997. — 368 с