Санкт-Петербургский политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Кафедра «Мехатроника и роботостроение» при ЦНИИ РТК

Курсовая работа

Дисциплина: Программирование на языках высокого уровня

Тема: Алгоритм Тарьяна

Выполнил

студент гр. 33335/2 Стрекозов А. В.

Преподаватель Ананьевский М. С.

« »_____2018 г.

Санкт-Петербург

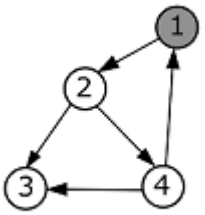
2018 г.

Формулировка задачи, которую решает алгоритм.

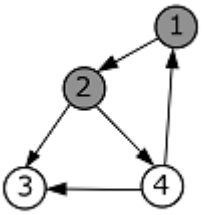
Алгоритм Тарьяна — это процедура систематического обхода всех вершин графа. Он обладает множеством полезных свойств, поэтому на его базе часто строятся алгоритмы решения различных задач на (ориентированных и неориентированных) графах.

Словесное описание алгоритма.

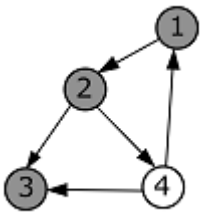
Присваиваем всем вершинам белый цвет. Затем проверяем, что первая вершина окрашена в белый цвет. Заходим в нее и раскрашиваем ее в серый цвет.



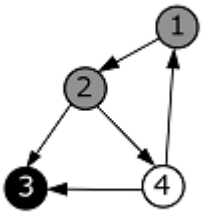
Пробуем пойти в вершину с номером 2. Проверяем, что она белая, и переходим в нее. Окрашиваем ее в серый цвет.



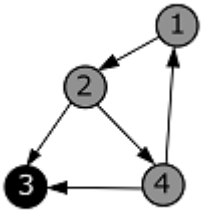
Пробуем пойти в вершину с номером 3. Проверяем, что она белая, и переходим в нее. Окрашиваем ее в серый цвет.



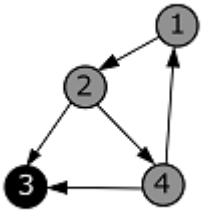
Проверяем, что из вершины с номером 3 не исходит ни одного ребра. Помечаем ее в черный цвет и возвращаемся в вершину с номером 2.



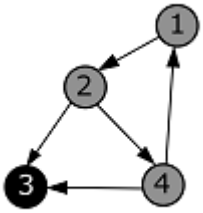
Пробуем пойти в вершину с номером 4. Проверяем, что она белая, и переходим в нее. Окрашиваем ее в серый цвет.



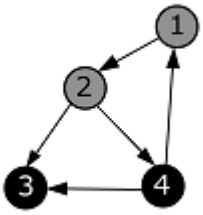
Пробуем пойти в вершину с номером 3. Видим, что она черного цвета, и остаемся на месте.



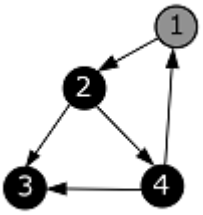
Пробуем пойти в вершину с номером 1. Видим, что она серого цвета, и остаемся на месте.



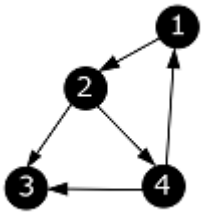
Из вершины с номером 4 больше нет исходящих ребер. Помечаем ее в черный цвет и возвращаемся в вершину с номером 2.



Из вершины с номером 2 больше нет исходящих ребер. Помечаем ее в черный цвет и возвращаемся в вершину с номером 1.



Из вершины с номером 1 больше нет исходящих ребер. Помечаем ее в черный цвет. Алгоритм завершен.



Псевдокод

```
function doTarjan(G[n]: Graph): // функция принимает граф G с количеством вершин n и выполняет обход в глубину во всем графе
   color = array[n, white]
   function Tarjan(u: int):
      color[u] = gray
      for v: (u, v) in G
         if color[v] == white
            Tarjan(v)
      color[u] = black
   for i = 1 to n
      if color[i] == white
```

Tarjan(i)

Реализация алгоритма.

Алгоритм был реализован при помощи языка программирования C++. Созданы 2 класса:

- Класс *node*, который имеет ключ *value*, а также вектор *edge* с набором ссылок на другие узлы. Также в этот класс также добавлено публичное свойство *color*, которое является необходимым для алгоритма Тарьяна.
- Класс *graph*, который содержит массив объектов *node*, а также их количество *amount*. В этом классе реализованы метод dfs, которые и являются основополагающими алгоритма Тарьяна. Также, для тестирования, в этом классе реализован конструктор, который создает ациклический орграф заданного размера со случайными ребрами (но не допуская циклов).

Основная часть кода приведена ниже:

```
bool graph::Tarjan(unsigned int curNode, uiVector &stack)
{
    //проверяю цвета текущей вершины
    if (this->nodes[curNode].color == COLOR_OF_NODE_GREY)
    {
        return true;
    }
    if (this->nodes[curNode].color == COLOR_OF_NODE_BLACK)
    {
        return false;
    }
    this->nodes[curNode].color = COLOR_OF_NODE_GREY;
    //запускаю поиск в глубину для всех вершин, на которые ссылается
текущая
    unsigned int edgeCount = this->nodes[curNode].edgeSize();
    for (unsigned int i = 0; i < edgeCount; i++)
    {
        unsigned int link = this->nodes[curNode].edge(i);
        if (this->Tarjan(link, stack))
        {
            return true;
        }
    }
```

```
    //если больше идти некуда - заношу в стек и крашу в черный
    stack.push_back(curNode);
    this->nodes[curNode].color = COLOR_OF_NODE_BLACK;
    return false;
}
```

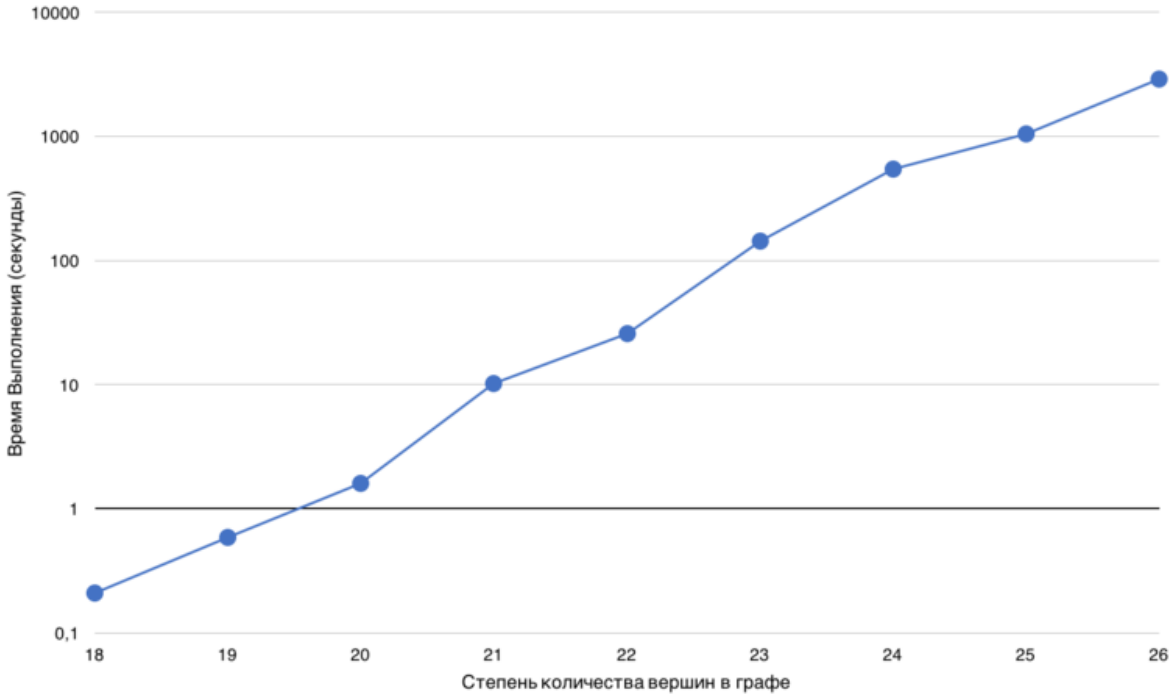
Анализ алгоритма

1. Временная сложность алгоритма

Сложность такого алгоритма соответствует сложности алгоритма поиска в глубину, то есть $O(m+n)$, где n – число вершин, m – число ребер. Это доказывается тем, что при проходе в глубину алгоритм лишь единожды проходит через единственную вершину, окрашивая ее в черный цвет. Таким образом, от каждой серой вершины, возможно только некоторое количество проверок на то, является ли смежная ей белой, а общее количество таких проверок от всех вершин равно количеству ребер в графе.

2. Время выполнения алгоритма:

Размер графа от 2^{18} до 2^{26}



Применение алгоритма

Алгоритм Тарьяна служит основой для топологической сортировки, которая в свою очередь является одной из основных алгоритмов на графах, который применяется для решения множества более сложных задач, например: при распараллеливании алгоритмов (логические схемы), когда по некоторому описанию алгоритма нужно составить граф зависимостей его операций и, отсортировав его топологически, определить, какие из операций являются независимыми и могут выполняться параллельно (одновременно), как пример - при создание карты сайта (компьютерные сети), где имеет место древовидная система разделов. Ко всему прочему, это может применятся при составлении: сети автомобильных дорог, схем метро и схем лабиринтов.

Список литературы

1. *Роберт Седжвик.* Глава 5. Метод уменьшения размера задачи: Топологическая сортировка // Алгоритмы на графах = Graph algorithms. — 3-е изд. — Россия, Санкт-Петербург: «ДиаСофтЮП», 2002. — С. 496. — ISBN 5-93772-054-7.
2. *Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.* Глава 23.1.3. Поиск в глубину // Алгоритмы: построение и анализ / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — С. 632-635. — ISBN 5-8459-0857-4.