ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: объектно-ориентированное программирование

Тема: алгоритм Тарьяна для поиска компонент сильной связности

Студент гр.3331506/90401: Ососов Я. Р.

Преподаватель: Ананьевский М.С

Санкт-Петербург
2022

Оглавление

ВВЕДЕНИЕ

Алгоритм Тарьяна - алгоритм в теории графов, наряду с алгоритмом Косарайю, для нахождения компонент сильной связности ориентированного графа. Данный алгоритм имеет линейную временную сложность.

Данный алгоритм применяется для конденсации графов, позволяя уйти от циклов внутри графа. Ориентированный цикличный граф заменятся графом конденсации, где циклы, являясь компонентами сильной связности, заменяются одним компонентом, характеризующим данную совокупность. Это позволяет решить ряд проблем, связанных с работой с ориентированными цикличными графами, например, поиска в глубину (Depth-first search).

Одно полезное свойство алгоритма состоит в том, что ни один сильно связный компонент не будет идентифицирован раньше любого из его потомков. Следовательно, порядок, в котором идентифицируются сильно связанные компоненты, представляет собой обратную топологическую разновидность DAG, образованную сильно связанными компонентами. Как побочный итог поиска компонент сильной связности ориентированного графа - осуществление образной топологической сортировки графа компонент сильной связности (графа конденсации).

Также данный алгоритм применяется для решения задачи 2-SAT (satisfiability - удовлетворяемость). Представим булевы формулы, в виде графа импликации, ориентированного графа, который выражает переменные экземпляра и их отрицания как вершины в графе, а ограничения на пары переменных как направленные ребра. В результате конденсации данного графа можно проверить необходимое и достаточное условие: формула 2-CNF выполнима тогда и только тогда, когда нет переменной, принадлежащей той же компоненте сильной связности, что и ее отрицание.

В свою очередь 2-SAT связана с: криптографический алгоритм RSA (обеспечения безопасности банковских транзакций, а так же для создания безопасного соединения), алгоритмы поиска «рекомендованного» контента, задача о ранце, задачи логистики и т.д.

1 Описание алгоритма

1.1 Некоторые сведения из теории графов

Ориентированный граф (орграф) – граф, ребрам которого присвоено направление.

Орграф сильно связный, или просто сильный, если все его вершины взаимно достижимы; максимальный сильный подграф называется компонентой сильной связности CSS (именно определение и выделение данной компоненты и является главной задачей алгоритма Тарьяна).

Древесное ребро (tree edge) – направлено от леса к деревьям;

Прямое ребро (forward edge) – направлено из вершины к потомку в дереве;

Перекрестное ребро (cross edge) – направлено от узла a к узлу b, где поддеревья с корнями a и b не пересекаются;

Обратное ребро (back edge) - от вершины к предку.

Пример ориентированного графа представлен на рисунке 1.

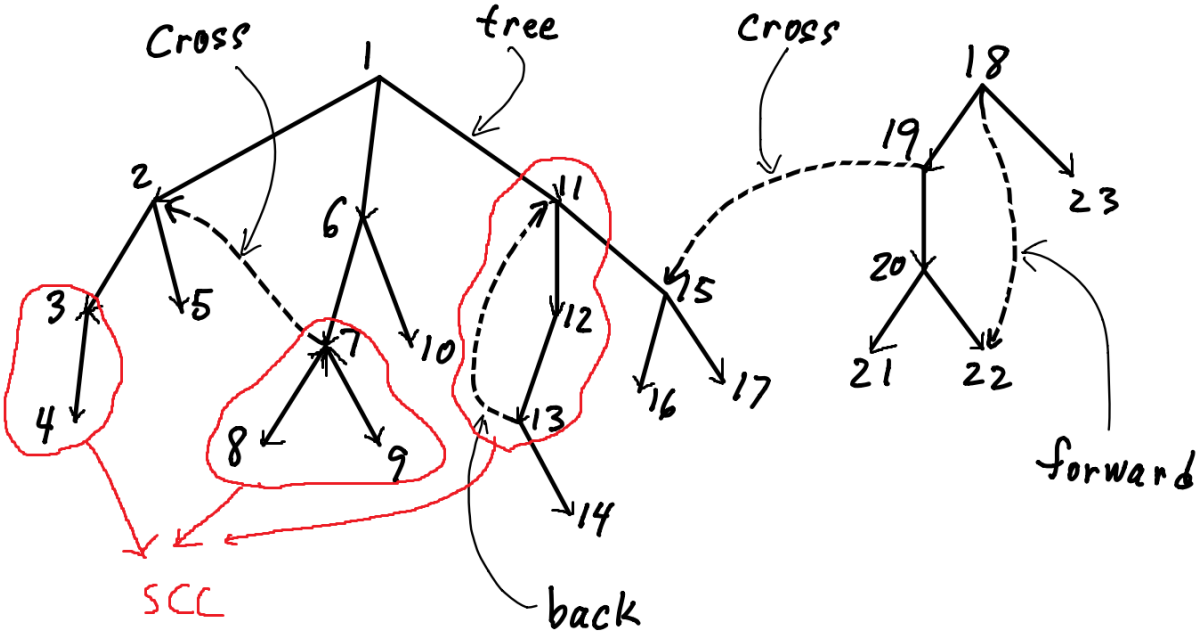


Рисунок 1 – Ориентированный граф

Поиск в глубину DFS (depth-first search) – один из методов обхода графа, лежащий в основе данного алгоритма. Стратегия поиска в глубину, как и следует из названия, состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из

рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину.

1.2 Идея алгоритма

Перед тем как обозначить основную идею алгоритма Тарьяна обозначим утверждения, лежащие в основе алгоритма.

1 Утверждение: SCC есть нечто непрерывное и связное. Не бывает такого, что две вершины компоненты сильной связности находятся в разных деревьях. Как следствие, если две вершины находятся в компоненте сильной связности, то все вершины на пути между ними являются частью компоненты сильной связности.

2 Утверждение: Все компоненты сильной связности связаны одним куском дерева обхода, покинув которое мы покидаем нашу компоненту сильной связности и не сможем вернуться обратно. Также верно и обратное.

Идея алгоритма такова: представляем граф в виде дерева с произвольно выбранного узла производится поиск в глубину (и последующие поиски в глубину ведутся на любых еще не найденных узлах). Переходя к новому, еще не посещенному узлу v, мы обозначаем его время вхождения (его номер в порядке обхода $num(v) = time$), добавляем его в стек и присваиваем значение переменной $low\_link(v) = num(v)$, которая характеризует принадлежность к компоненте сильной связности (номер элемента сильной связности с минимальным порядковым номером вхождения). Далее делаем вызов DFS к элементу w. При откате из DFS обновляем значение следующим образом:

$$low\_link(v) \rightarrow \min(low\_link(v), low\_link(w))$$

Если вызов DFS привел нас в уже посещенный узел, который находится в стеке, обновляем значение следующим образом:

$$low\_link(v) \rightarrow \min(\mathrm{num}(w), low\_link(v))$$

Попробуем привести небольшое доказательство:

Tree edge:

Предположим, что w не находится в том же SCC, что и v. Это противоречит утверждению 2, следовательно это недостижимо.

Предположим, что w находится в том же SCC, что и v. Тогда, как раз приведенная выше формула, показывает, что значение low_link потомка, который имеет связь к предку, есть num предка.

Back edge:

Предположим, что w находится в том же SCC, что и v. Тогда, будучи предком w должен быть в стеке, как и v. Тогда, как раз приведенная выше формула, показывает, что значение low_link потомка, который имеет связь к предку, есть num предка.

Предположим, что w не находится в том же SCC, что и v. Это противоречит утверждению 2, следовательно это недостижимо.

Cross edge:

Предположим, что w находится в том же SCC, что и v. Тогда он либо должен находиться стеке, и тогда вышеописанная формула работает корректно, если же нет, то это будет противоречить условию 1.

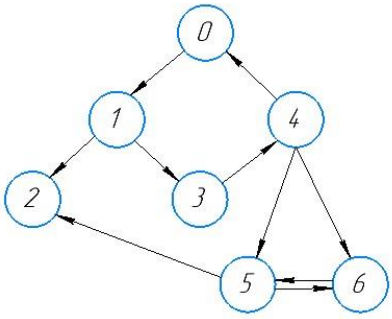
Предположим, что w не находится в том же SCC, что и v. Тогда ее не будет в стеке.

Что подтверждает, правильность вышеописанной формулы обновления значения low_link.

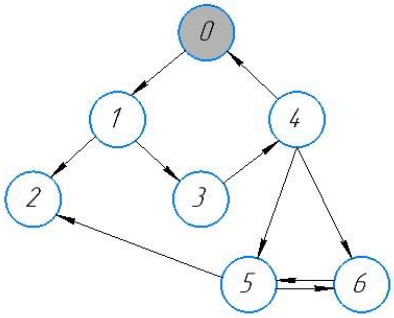
Когда мы завершаем функцию DFS(v), мы проверяем, если $low\_link(v) = num(v)$. Если это так, то v является базовой вершиной, так что мы делаем достаем вершины из стека, пока v не будет достигнута, тогда мы останавливаемся. Каждая такая группа вершин является SCC.

Небольшое замечание: если для всех вершин кроме нулевой в результате выполнения алгоритма Тарьяна выполняется следующее условие $low\_link < num$, то данный граф – сильный.

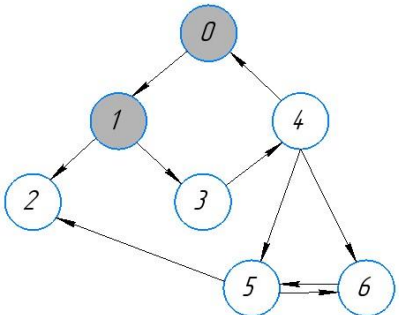
1.3 Пример реализации алгоритма



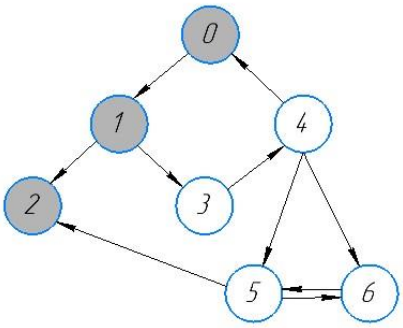
time = 0	0	1	2	3	4	5	6
queue_num	-1	-1	-1	-1	-1	-1	-1
low_link	-1	-1	-1	-1	-1	-1	-1
presents_in_Stack	F	F	F	F	F	F	F
stack							



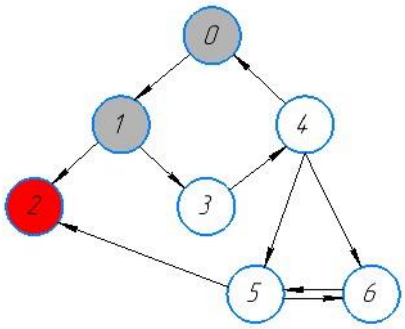
time = 0	0	1	2	3	4	5	6
queue_num	0	-1	-1	-1	-1	-1	-1
low_link	0	-1	-1	-1	-1	-1	-1
presents_in_Stack	T	F	F	F	F	F	F
stack	0						



time = 1	0	1	2	3	4	5	6
queue_num	0	1	-1	-1	-1	-1	-1
low_link	0	1	-1	-1	-1	-1	-1
presents_in_Stack	T	T	F	F	F	F	F
stack	0	1					

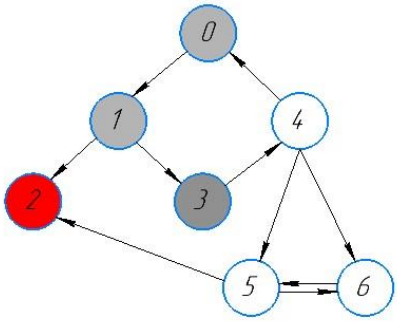


time = 2	0	1	2	3	4	5	6
queue_num	0	1	2	-1	-1	-1	-1
low_link	0	1	2	-1	-1	-1	-1
presents_in_Stack	T	T	T	F	F	F	F
stack	0	1	2				



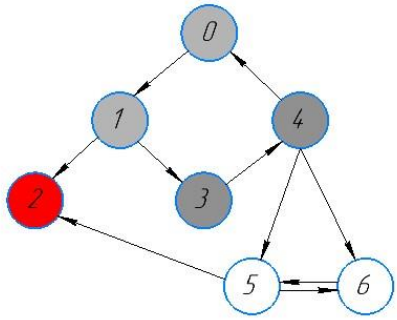
time = 2	0	1	2	3	4	5	6
queue_num	0	1	2	-1	-1	-1	-1
low_link	0	1	2	-1	-1	-1	-1
presents_in_Stack	T	T	F	F	F	F	F
stack	0	1					

SCC: 2



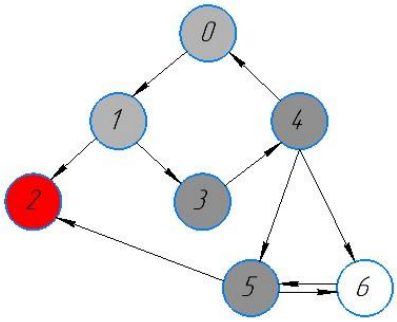
time = 3	0	1	2	3	4	5	6
queue_num	0	1	2	3	-1	-1	-1
low_link	0	1	2	3	-1	-1	-1
presents_in_Stack	T	T	F	T	F	F	F
stack	0	1	3				

SCC: 2



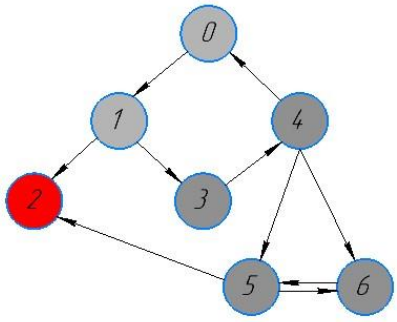
time = 4	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	-1	-1
low_link	0	1	2	3	4	-1	-1
presents_in_Stack	T	T	F	T	T	F	F
stack	0	1	3	4			

SCC: 2



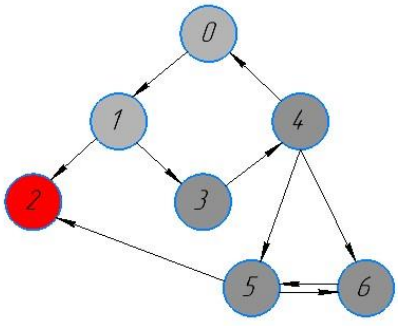
time = 5	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	-1
low_link	0	1	2	3	4	5	-1
presents_in_Stack	T	T	F	T	T	T	F
stack	0	1	3	4	5		

SCC: 2



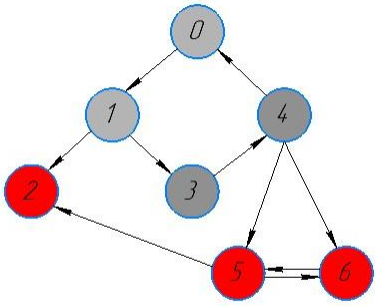
time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	1	2	3	4	5	6
presents_in_Stack	T	T	F	T	T	T	T
stack	0	1	3	4	5	6	

SCC: 2



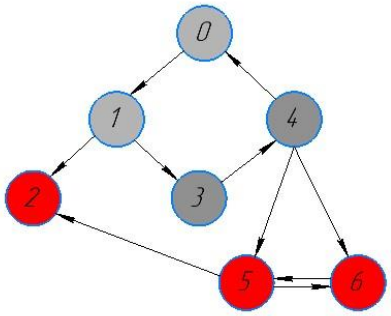
time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	1	2	3	4	5	5
presents_in_Stack	T	T	F	T	T	T	T
stack	0	1	3	4	5	6	

SCC: 2



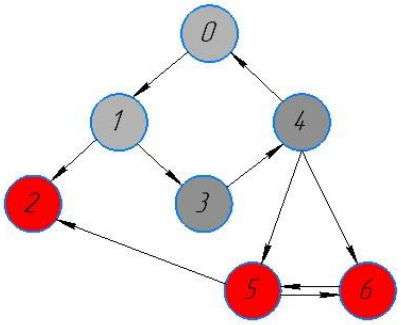
time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	1	2	3	4	5	5
presents_in_Stack	T	T	F	T	T	F	F
stack	0	1	3	4			

SCC: 2 SCC: 5,6



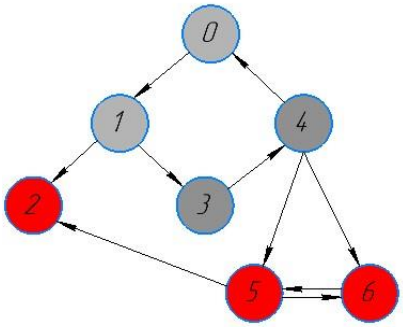
time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	1	2	3	0	5	5
presents_in_Stack	T	T	F	T	T	F	F
stack	0	1	3	4			

SCC: 2 SCC: 5,6



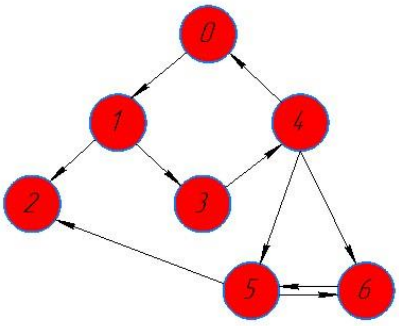
time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	1	2	0	0	5	5
presents_in_Stack	T	T	F	T	T	F	F
stack	0	1	3	4			

SCC: 2 SCC: 5,6



time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	0	2	0	0	5	5
presents_in_Stack	T	T	F	T	T	F	F
stack	0	1	3	4			

SCC: 2 SCC: 5,6



time = 6	0	1	2	3	4	5	6
queue_num	0	1	2	3	4	5	6
low_link	0	0	2	0	0	5	5
presents_in_Stack	F	F	F	F	F	F	F
stack							

SCC: 2 SCC: 5,6 SCC: 0,1,3,4

2 Исследование алгоритма

Временная сложность. Как можно заметить в данном алгоритме, каждая вершина графа так или иначе посещается один раз. Также в DFS_Tarjan рассматривается каждое ребро не более одного раза. Таким образом, время работы алгоритма Тарьяна – $O(V + E)$, где V – число узлов графа, E – число ветвей графа. На рисунке 2 представлен график зависимости скорости выполнения алгоритма от размера входных данных, для случайного графа.

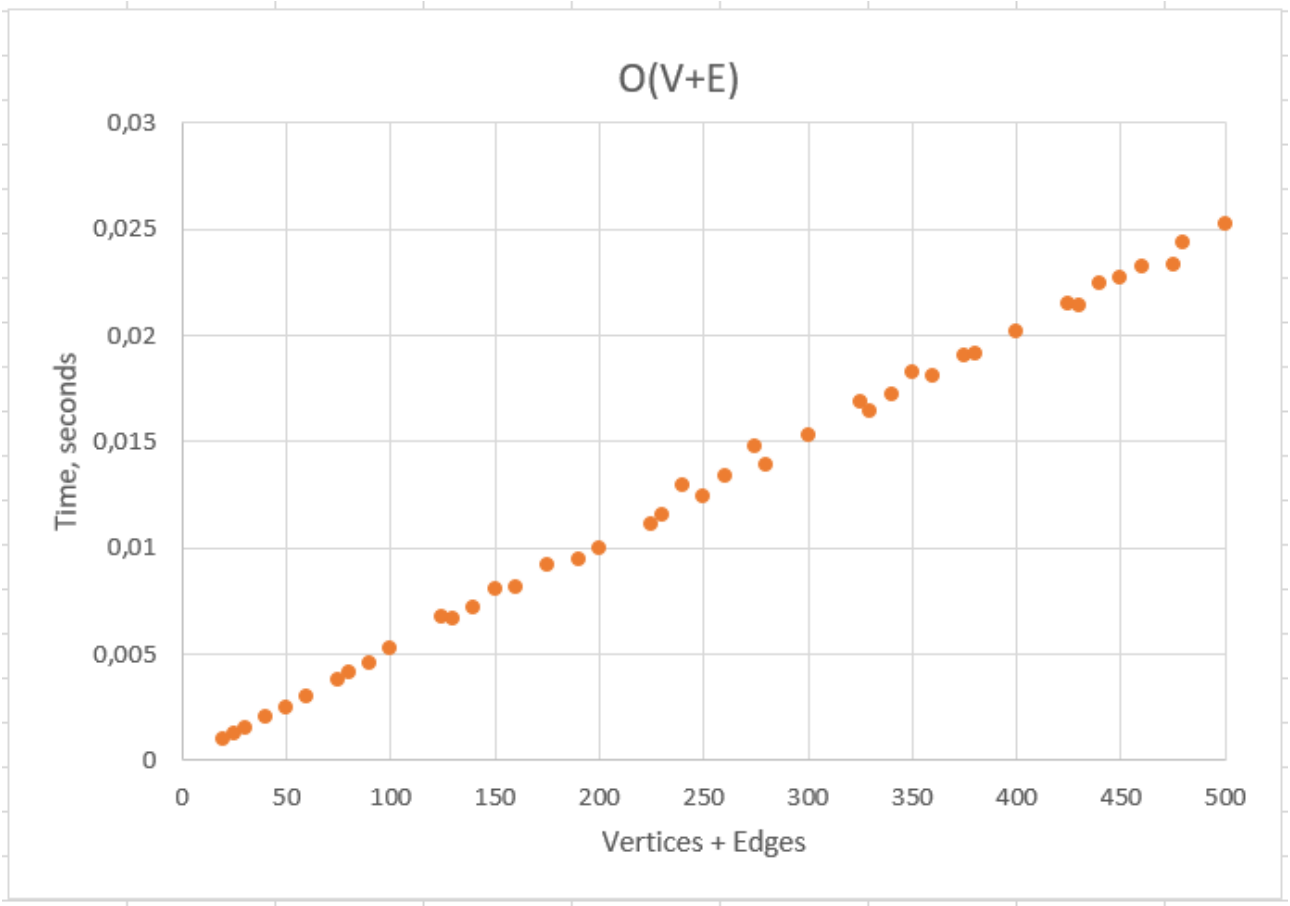


Рисунок 2 – График зависимости скорости выполнения алгоритма от размера входных данных

Также в дополнение была снята зависимость времени выполнения алгоритма, на полном графе (все вершины связаны между собой). Данная зависимость представлена на рисунке 3.

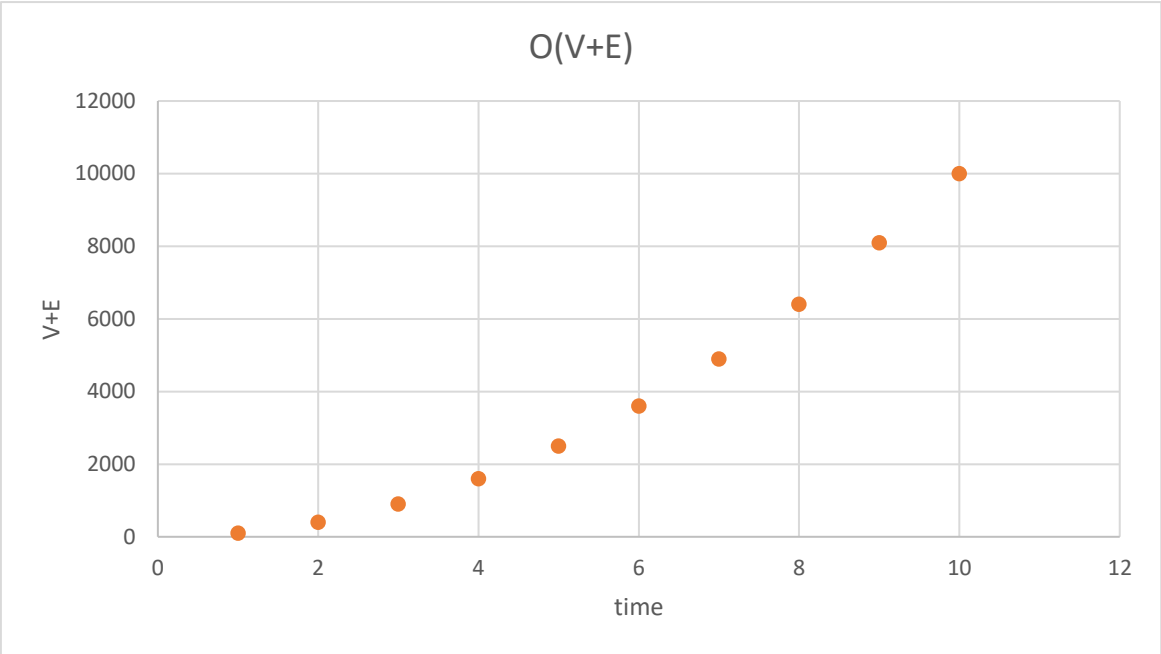


Рисунок 3 – График зависимости скорости выполнения алгоритма от размера входных данных на полном графе

Данная зависимость уже нельзя назвать идеально линейной.

ЗАКЛЮЧЕНИЕ

В данной работе был рассмотрен и реализован на языке C++ алгоритм Тарьяна для нахождения SCC ориентированного графа. Было проведено исследование зависимости скорости выполнения алгоритма от размера входных данных для двух разных случаев. В результате было получено, что временная сложность данного алгоритма линейна по отношению к сумме узлов и ветвей графа($O(V+E)$). Одна из интересных особенностей данного алгоритма – обратная топологическая сортировка, образованная сильно связанными компонентами.

СПИСОК ЛИТЕРАТУРЫ

1. Tarjan's strongly connected components algorithm [Электронный ресурс]//https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

2. Design & Analysis of Algorithms, Lecture #19: Depth First Search and Strong Components, [Электронный ресурс] // https://www.cs.cmu.edu/~15451-f18/lectures/lec19-DFS-strong-components.pdf?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=sc\

3. 2 – satisfiability [Электронный ресурс] // https://en.wikipedia.org/wiki/2-satisfiability

4. Роберт Седжвик, Фундаментальные алгоритмы на C++. Часть 5: алгоритмы на графах. – 3-е изд. – СПб: ООО «ДиаСофтЮП», 2002. – 496 с.