САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Эмпирический анализ алгоритма Косарайю поиска компонент сильной связности в ориентированном графе

По дисциплине: Алгоритмы и анализ сложности

Направление: Фундаментальная информатика и информационные технологии

Выполнила студентка 3 курса 20.Б12-пу

Радькова Ирина Тимофеевна

Санкт-Петербург

2022г.

Содержание

1. Описание алгоритма

1.1. История создания и краткое описание

Алгоритм Косарайю для определения сильно связных компонент в ориентированном графе был описан в 1978 году Самбасивой Рао Косарайю, а опубликован в 1983 году в книге «Структуры данных и алгоритмы» А.Ахо, Д.Хопкрофта и Д.Ульмана.

Работа алгоритма представима в виде трёх этапов:

1. Осуществление поиска в глубину на исходном гафе G. Вершины нумеруются в порядке выхода из рекурсивно вызываемой функции поиска в глубину;
2. Создание нового ориентированного графа G_r путём обращения направления всех дуг графа G;
3. Осуществление поиска в глубину на графе G_r, начиная с вершины, имеющей наибольшей номер. Если поиск не охватывает всех вершин, то начинается новый поиск с вершины, имеющей наибольший номер среди оставшихся;

Результатом работы алгоритма является остовной лес, каждое дерево которого является сильно связной компонентой G.

1.2. Область применения алгоритма

Данный алгоритм используется при решении задачи поиска 2-SAT. Выделив компоненты сильной связности, анализируют, принадлежат ли парные вершины одной компоненте. Если никакие две парные вершины не попали в одну компоненту сильной связности, то решение задачи существует.

2. Математический анализ алгоритма

Пусть V, E – количество вершин и рёбер ориентированного графа G соответственно. Выясним сложность изучаемого алгоритма.

Работа алгоритма сводится к выполнению двух обходов в глубину. Для того, чтобы совершить один обход, потребуется просмотреть все вершины графа, что потребует $\Theta(V)$ времени. Для того, чтобы переместиться из текущей вершины v в новую непомеченную вершину, потребуется $\Theta(|adj[v]|)$ времени, где $adj[v]$ – множество вершин, к которым идёт ребро из v. Поскольку

$$\sum_{v \in V} |adj[v]| = \Theta(E),$$

то итоговое время работы обхода в глубину можно оценить как $\Theta(V+E)$. В таком случае алгоритм Косарайю также будет работать за $\Theta(V+E)$.

3. Входные данные и единицы измерения трудоёмкости

3.1. Характеристики

Входными данными алгоритма являются граф с числом вершин, равным $V \in [2, 2^{20}], V \in N$, и количеством рёбер, равным E. V, E выбираются таким образом, чтобы выполнялось равенство $V + E = 2^i$, где i – натуральное число.

Количество тестов – 30, в каждом из них алгоритм Косарайю выполняет 20 итераций, на каждой из которых i инкрементируется.

В качестве единиц измерения трудоёмкости выберем время работы алгоритма, измеряемое в наносекундах.

3.2. Генерация входных данных

Будем генерировать V, E случайным образом.

```
// generate random edge function
tuple<int, int> generateEdge(int V) {
	tuple <int, int> newEdge;
	get<0>(newEdge) = rand() % V;
	get<1>(newEdge) = rand() % V;

	return newEdge;
}

void makeTest(int N, ofstream& outFile) {
	int V, E;
	int n = 2; // n = 2^i

	for (int i = 0; i < N; i++) {
		// generating random values of V, E
		V = rand() % n + 1;
		E = n - V;

		// if V+E > 2^i, then regenerate V, E
		while (E > V * (V - 1)) {
		V = rand() % n + 1;
		E = n - V;

		// generating new edge
		tuple <int, int> edge;
		tuple<int, int>* edges = new tuple<int, int>[E];

		Graph g(V);
		for (int i = 0; i < E; i++) {
			// generating new edge
			edge = generateEdge(V);

			// if edge exists, regenerate it
			while (g.edgeExists(get<0>(edge), get<1>(edge))) {
				edge = generateEdge(V);
			}

			g.addEdge(get<0>(edge), get<1>(edge));
		}

	}
```

(Листинг 1. Генерация числа вершин и рёбер на каждой итерации i)

4. Описание реализации алгоритма

В качестве языка программной реализации выбран C++.

```
// The main function that finds and prints all strongly connected
// components
void Graph::printSCCs()
{
	stack<int> Stack;

	// Mark all the vertices as not visited (For first DFS)
	bool* visited = new bool[V];
	for (int i = 0; i < V; i++)
		visited[i] = false;

	// Fill vertices in stack according to their finishing times
	for (int i = 0; i < V; i++)
		if (visited[i] == false)
			fillOrder(i, visited, Stack);

	// Create a reversed graph
	Graph gr = getTranspose();

	// Mark all the vertices as not visited (For second DFS)
	for (int i = 0; i < V; i++)
		visited[i] = false;

	// Now process all vertices in order defined by Stack
	while (Stack.empty() == false)
	{
		// Pop a vertex from stack
		int v = Stack.top();
		Stack.pop();

		// Print Strongly connected component of the popped vertex
		if (visited[v] == false)
		{
			gr.DFSUtil(v, visited);
			//cout << endl;
		}
	}
}
```

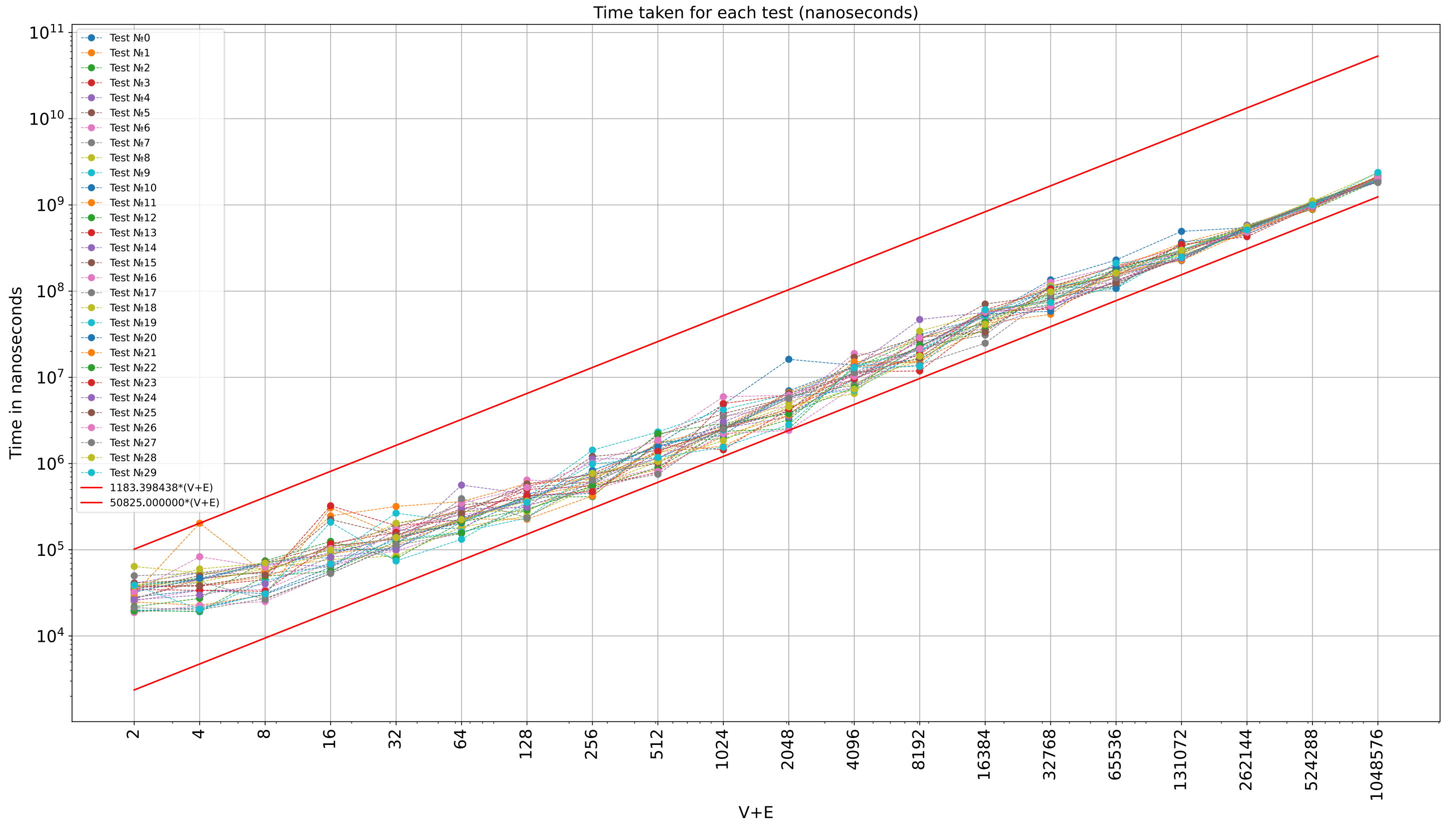
(Листинг 2. Функция, реализующая алгоритм Косарайю)

Полная реализация алгоритма доступна по ссылке: https://github.com/baddabudda/kosaraju.

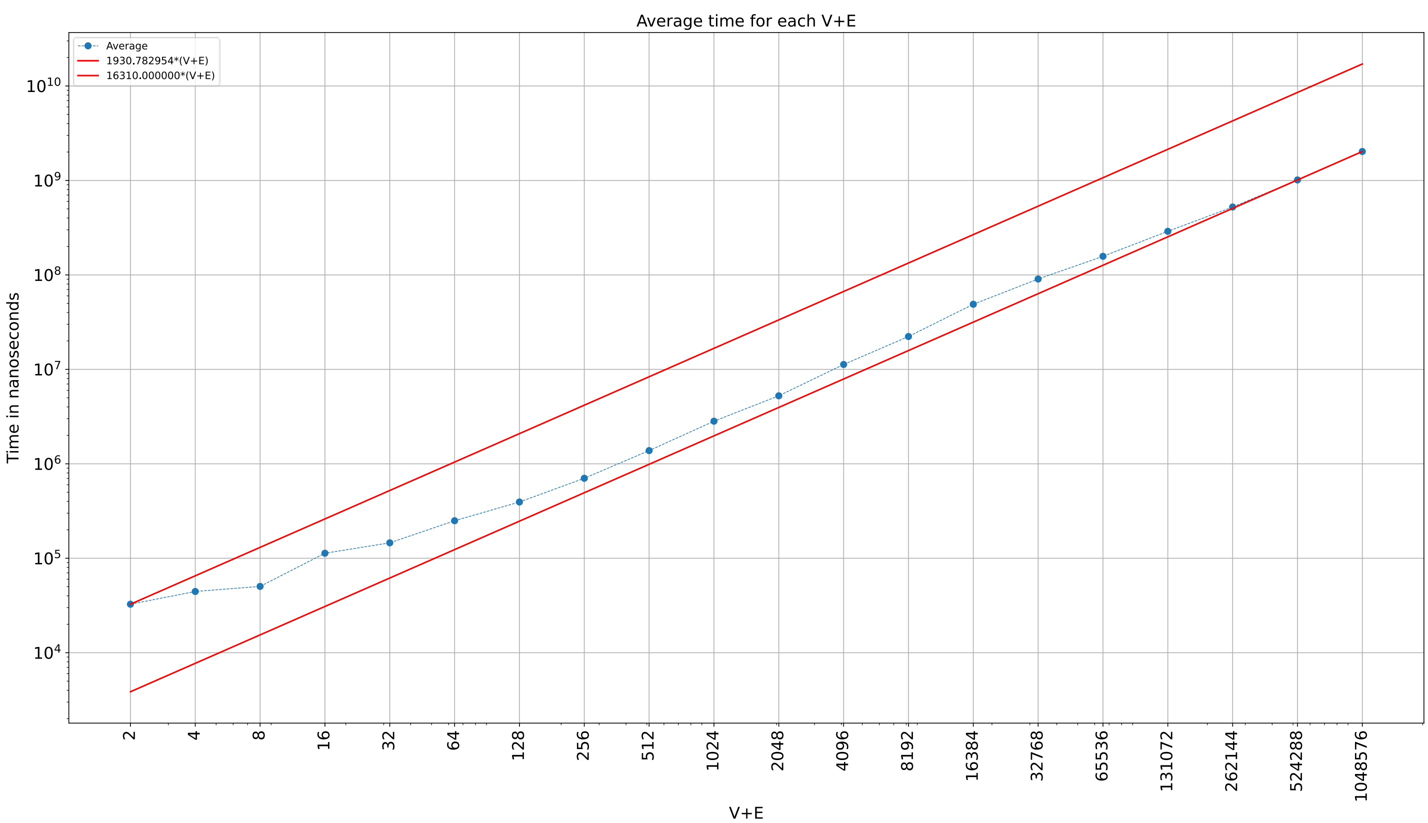
5. Анализ вычислительного эксперимента

В соответствии с входными данными, описанными в п.3, был проведён вычислительный эксперимент.

На *Рис.1* представлена зависимость среднего времени работы алгоритма от величины $V + E$ для каждого из 30 тестов. Видно, что значения времени не выходят за пределы теоретической функции, домноженной на константу.



(Рис.1 Время работы алгоритма на каждом из 30 тестирований)



(Рис.2 Среднее время работы алгоритма на каждом $V + E$)

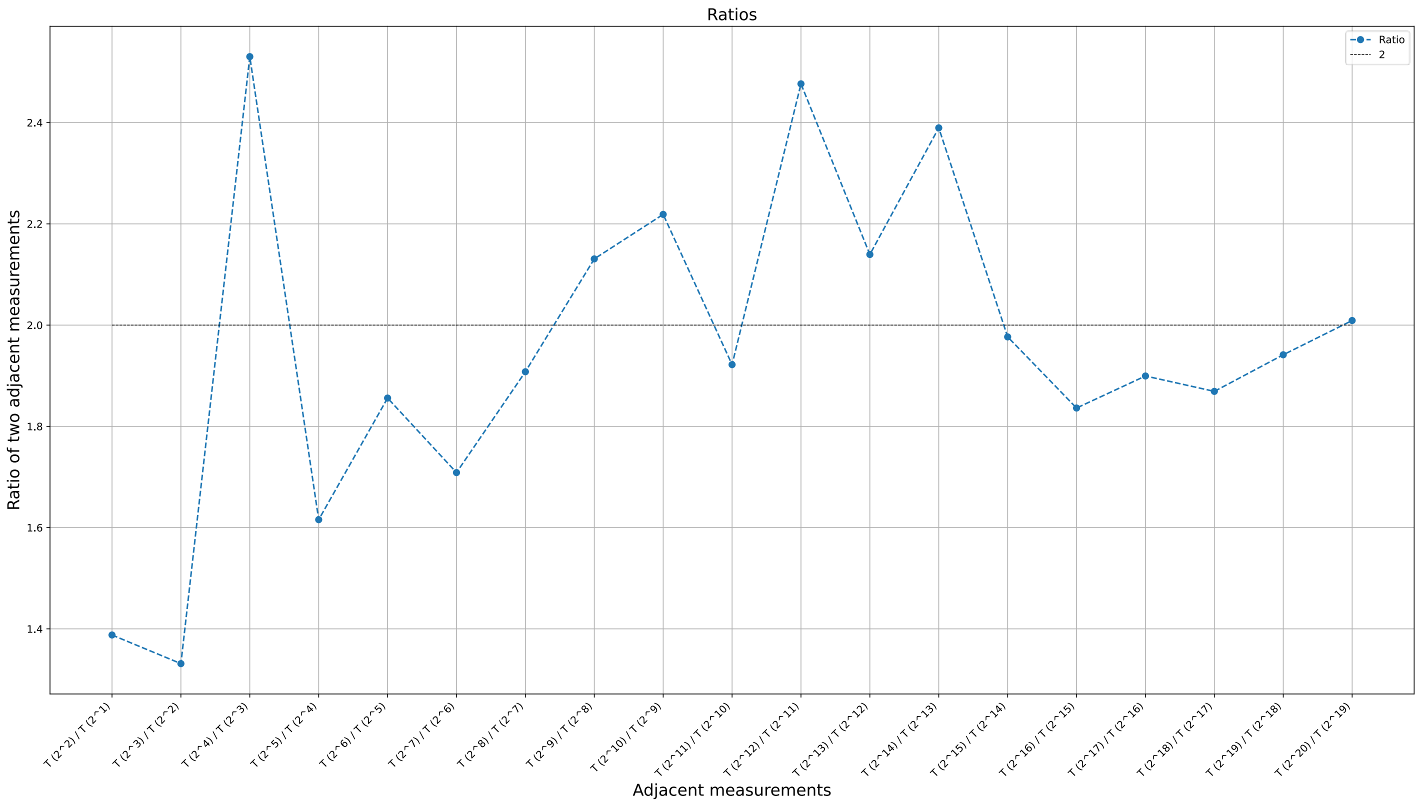
На *Рис.2* представлено среднее время работы 30 тестирований по каждому значению $V + E$. При больших значениях входных данных видно стремление к нижней теоретической оценке.

В целом, можно заключить, что алгоритм действительно принадлежит классу $\Theta(V + E)$.

Рассмотрим отношение значений измеренной трудоёмкости при удвоении размера входных данных. Отношение теоретических оценок следующее:

$$\frac{T(2(V+E))}{T(V+E)} = 2$$

С данными эксперимента проведём следующие манипуляции: вычислим отношение значений трудоёмкости для двух соседних размеров входов для каждого из 30 тестов, а затем усредним результаты по каждому из отношений. *Рис.3* иллюстрирует полученные результаты.



(*Рис.3 Отношение значений трудоёмкости для соседних размеров входных данных)*

Как видно из *Рис.3,* значения очень сильно колеблются. Однако при довольно больших размерах входных данных разброс между ними уменьшается и стремится к теоретической оценке.

6. Список использованных литературных источников

1. Ахо, Альфред В., Ульман, Джеффри Д., Хопкрофт, Джон Э. Структуры данных и алгоритмы (2019)
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms Third Edition (2009)

7. Характеристики использованного оборудования

7.1 Вычислительная среда

Microsoft Visual Studio Community 2019, версия 16.11.20.

7.2 Характеристики оборудования

- ОС: Windows 10 Pro
- Процессор: AMD Athlon 300U
- Оперативная память: 12Гб DDR4