Отчет по исследованию алгоритма Дейкстры

Голубев Роман

7 декабря 2020 г.

Содержание

Резюме

Введение

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его используют протоколы маршрутизации OSPF и IS-IS. Появился в результате работы Дейкстры над задачей по оценке производительности компьютера ARCMAC, установленного в Математическом Центре.

Постановка задачи

Пусть задан ориентированный граф $G = (V, E, \phi)$, где:

- V — множество вершин графа.
- $E \subset V \times V$ — множество ребер.
- $\phi: E \to \mathbb{R}_+$ — весовая функция, сопоставляющая каждому ребру т.н. "вес" — некоторое неотрицательное число.

Для фиксированной вершины $v_0 \in V$ требуется найти кратчайшее расстояние до любой вершины $u \in V$, т.е. величину $\mathrm{dist}_{v_0}(u)$:

$$\mathrm{dist}_{v_0}(u) = \min \{\, d \mid \exists n \in \mathbb{N}_0 :\ d = \sum_{i=0}^{n} \phi((v_i, v_{i+1})),\ \ \forall i \in \overline{0,n} : v_i \in V \wedge v_n = u \,\}$$

В случае, когда множество возможных значений d пусто, минимум предполагается равным ∞.

Неформальное описание алгоритма:

В начале расстояние до вершины v_0 полагается равным 0, для всех остальных ∞.

В алгоритме поддерживается множество вершин S, для которых уже вычислены окончательные кратчайших расстояния к ним из вершины v_0. На каждом шаге алгоритма выбирается вершина $u \in V \setminus S$, которой на данном этапе соответствует минимальная оценка кратчайшего пути. После этого, вершина u добавляется в множество S и происходит релаксация по всем исходящим из нее ребер.

Работа алгоритма завершается на этапе, когда на текущем шаге невозможно выбрать новую вершину. В случае связного графа — когда просмотрены все вершины, в случае несвязного — когда просмотрены все вершины компоненты связности.

Формальное описание алгоритма

Ниже приведен псевдокод реализующий алгоритм Дейкстры:

```
Dijkstra(G, s):
1   Initialize(G, v_0)
2   S := ∅
3   Q := G.V
4   while Q ≠ ∅
5       u := ExtractMin(Q)
6       S := S ∪ { u }
7       for v ∈ G.N(u):
8           Relax(u, v)
```

Не теряя общности будем считать граф G связным - в противном случае множество Q в приведенной ниже реализации надо инициализировать компонентой связности G содержащей вершину v_0.

Множество $N(u) \subset V$ — множество соседей вершины $u \in V$

Пояснения к процедурам:

Процедура Initialize принимает граф G и начальную вершину v_0 и инициализирует расстояния (см. нефор-мальное описание).

Процедура Extract извлекает из Q вершину с наименьшей оценкой минимального расстояния. Таким образом поддерживается инвариант $Q = V \setminus S$.

Процедура Relax производит релаксацию по всем соседям вершины u — если расстояние до вершины u плюс $w(u, v)$ меньше чем текущее расстояние v, то происходит обновление расстояния до вершины v

Математический анализ алгоритма

В наивной реализации множество Q устроено как массив. Тогда сложность процедур устроена так:

- Сложность процедуры Initialize — $\Theta(V)$.
- Количество итераций цикла **while** — $\Theta(V)$
- Сложность процедуры ExtractMin — $\Theta(V)$.
- Общее количество итераций цикла **for** — $\Theta(E)$ (в сумме по всем итерациям цикла **while**).
- Сложность процедуры Relax — $\Theta(1)$.

Итоговая сложность

$$\Theta(V) + \Theta(V) \cdot \Theta(V) + \Theta(E) \cdot \Theta(1) = \Theta(V^2 + E) = \Theta(V^2)$$

Отметим, что существуют и другие реализации, использующие фиббоначевы/двоичные кучи, которые дают другие ассимтотические оценки.

Характеристики входных данных

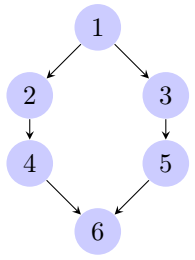
Структура входных данных алгоритма

Входные данные должны содержать описание графа и фиксированную вершину. Не теряя общности, положим что вершинами графа являются натуральные числа, а сам граф не содержит кратных ребер и петель.

Первая строка содержит два числа n и m — количество вершин и ребер в графе

Следующие m строк содержат три числа u, v, w — означающие что в графе есть ребро (u, v) с весом w.

Следующая строка содержит одно число p — номер фиксированной вершины.




```
6 6
1 2 1
1 3 1
3 5 1
2 4 1
4 6 1
5 6 1
1
```

Рис. 1: Иллюстрация: в графе слева все ребра имеют вес равный 1, справа дано его формальное описание

Генерация входных данных

Числа n и m выбираются из фиксированного диапазона, причем $\max(m) \leqslant n^2$. После чего генерируется m чисел из диапазона $[0; n^2)$. Каждое число соответствует ребру (u, v) по следующему правилу:

$$x \mapsto (x \text{ div } n, x \bmod n)$$

После чего каждому ребру назначается еще одно число из фиксированного диапазона — вес. Так как диапазон весов не влияет на время работы программы, будем считать вес целочисленной величиной.

После генерации достаточно выбрать одну вершину, которая будет считаться фиксированной.

Формальная процедура генерации выглядит так:

GenGraph(N) :

1 $M \coloneqq \text{GetRandomInt}\left([N^{3/2}; N^2]\right)$

2 $E \coloneqq \text{GetRandomSeq}\left([0; N^2 - 1], M\right)$

3 $\text{Write}(N, M)$

4 **for** $e \in E$:

5 $\quad \text{Write}\left(\text{e div N}, \text{e mod N}, \text{GetRandomInt}\left([1; 10]\right)\right)$

6 $\text{Write}\left(\text{GetRandomInt}\left([0; N - 1]\right)\right)$

Ограничение на количество ребер снизу вызвано тем, что при малых значениях M граф может получена разряженным, что нежелательно.

Вычислительный эксперимент

Цели

- Установить эмпирическую зависимость времени исполнения программы-алгоритма при росте параметра N от 5 вершин до 100;
- Полученные результаты сравнить с теоретическими оценками;

Методология

Для каждого $n \in [5; 100]$ кратного 5 сгенерировать 100 тестовых случаев и взять медиану по времени исполнения.

Полученные результаты сравнить с теоретическими оценками следующим образом:

- Посмотреть зависимость величины $\frac{\mathcal{T}(n)}{n^2}$ от параметра n.
- Посмотреть зависимость величины $\frac{\mathcal{T}(2n)}{\mathcal{T}(n)}$

Здесь $\mathcal{T}(n)$ — медиана времени работы программы по всем тестам размера n.

Результаты

Численные результаты представлены в таблице 1

N	$\mathcal{T}(N)$	$\frac{\mathcal{T}(N)}{N^2}$	$\frac{\mathcal{T}(2N)}{\mathcal{T}(N)}$
5	103	4.12	2.15
10	221.5	2.22	3.20
15	433.5	1.93	3.55
20	708	1.77	3.67
25	978.5	1.57	4.28
30	1540.5	1.71	3.47
35	2400.5	1.96	3.05
40	2598.5	1.62	3.54
45	3576.5	1.77	3.21
50	4187	1.67	3.80
55	4685	1.55	-
60	5342	1.48	-
65	6210	1.47	-
70	7313.5	1.49	-
75	8598	1.53	-
80	9193.5	1.44	-
85	10822	1.50	-
90	11466.5	1.42	-
95	13246	1.47	-
100	15898	1.59	-
Арифмит.		**1.76**	**3.39**
Медиана		**1.58**	**3.50**

Таблица 1: Результаты эксперимента

Зависимость $\mathcal{T}(n)$ представлена на рисунке 2:

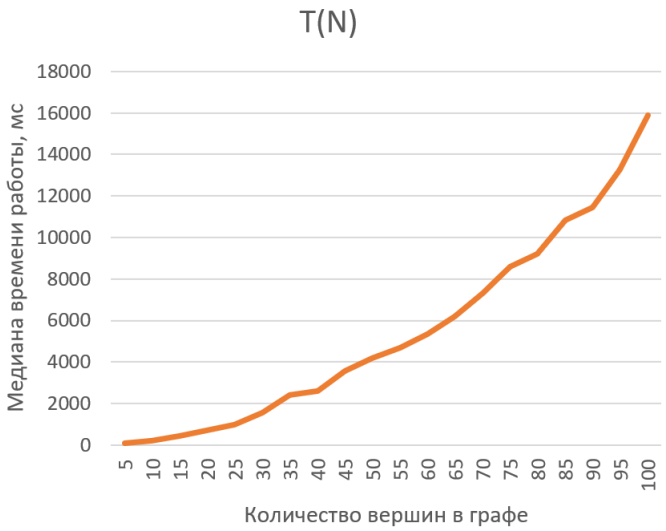


Рис. 2: $\mathcal{T}(n)$

Выводы

Эмпирически полученная зависимость $\mathcal{T}(n)$ действительно имеет временную сложность $\Theta(n^2)$. При малых значениях n наблюдаются выбросы, однако при асимптотическом росте n, отношение $\frac{\mathcal{T}(n)}{n^2}$ стабилизируется к 1.5. Отношение $\frac{\mathcal{T}(2n)}{\mathcal{T}()}$ должно быть около 4, однако наблюдаемые значения находятся ближе к 3.5 — скорее всего, из-за малых размеров n.

Список литературы

- Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. Алгоритмы: Построение и анализ [2005]
- Edsger Dijkstra. A note on two problems in connexion with graphs [1959]