Иван Николаевич ТКАЧЕНКО[1]
Михаил Викторович ГРИГОРЬЕВ[2]
Инна Ивановна ГРИГОРЬЕВА[3]

УДК 004.021

СОРТИРОВКА ЗУБЧАТЫХ МАССИВОВ ДЛЯ ОЦЕНКИ РЕШЕНИЙ ТРАНСПОРТНЫХ ЗАДАЧ

[1] ведущий программист ООО «Техноком» (г. Тюмень)
tkachenko@technocom.tech

[2] кандидат технических наук, доцент
кафедры программной и системной инженерии,
Тюменский государственный университет
m.v.grigorev@utmn.ru

[3] кандидат технических наук, доцент
кафедры программной и системной инженерии,
Тюменский государственный университет
i.i.grigoreva@utmn.ru

Аннотация

В статье исследуются способы сортировки зубчатых массивов для упорядочивания строк по убыванию значимости по двум критериям. Зубчатый массив является представлением решений задачи нахождения оптимального пути, в котором каждая строка — одно из решений. Цель сортировки — выбор наиболее значимых решений и последующее сокращение количества объектов, хранимых в матрице достижимости. Сокращение количества объектов не влияет на размерность матрицы достижимости, но сокращает объем памяти, необходимый для хранения ячеек матрицы. Задача описана на примере оценки решений транспортной задачи доставки грузов.

В статье при сортировке строк зубчатых массивов подразумевается приоритет одних ячеек строки массива над другими, т. е. первая ячейка оказывает на результат большее значение, чем вторая, а вторая ячейка большее, чем третья, и так далее.

Цитирование: Ткаченко И. Н. Сортировка зубчатых массивов для оценки решений транспортных задач / И. Н. Ткаченко, М. В. Григорьев, И. И. Григорьева // Вестник Тюменского государственного университета. Физико-математическое моделирование. Нефть, газ, энергетика. 2017. Том 3. № 2. С. 100-114.
DOI: 10.21684/2411-7978-2017-3-2-100-114

Результатом исследования является формализация правил восьми различных подходов к сортировке — по количеству вариантов комбинирования двух критериев сортировки, приведен алгоритм и дана оценка вычислительной сложности. Критериями сортировки являются длина ключа и значение ключа. Все восемь подходов обеспечивают разные результаты сортировки. Для каждого подхода описаны и разобраны на примерах операции, необходимые для достижения требуемого результата. Для каждого подхода сформулирована и формализована задача.

Алгоритм сортировки зубчатых массивов основан на генерации текстового представления ключа строки. В статье рассмотрена только сортировка строк, сортировка столбцов в статье не рассматривается.

Сделаны выводы о том, какие из описанных подходов обеспечивают наилучшую скорость сортировки и почему. На примере задачи транспортировки грузов описано — к какому результату приведет выбор того или иного подхода к сортировке.

Ключевые слова

Сортировка зубчатых массивов, решение транспортной задачи.

DOI: 10.21684/2411-7978-2017-3-2-100-114

Введение

В настоящее время существует множество алгоритмов решения задачи нахождения оптимального пути: алгоритм Флойда — Уоршелла, алгоритм Джонсона и т. д.

Например, в алгоритме Флойда — Уоршелла [3] определена рекуррентная формула нахождения оптимального пути (1):

$$d_{ij}^{(k)} \begin{cases} w_{ij}, & \text{если } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & \text{если } k \geq 1, \end{cases} \quad (1)$$

где $d_{ij}^{(k)}$ — вес пути между вершинами i и j с промежуточными вершинами из множества $\{1, 2, ..., k\}$, k — возможная промежуточная вершина пути между i и j; при $k = 0$ промежуточных вершин нет, поэтому $d_{ij}^{(k)} = w_{ij}$.

В формуле (1) задана функция min, выбирающая из двух кратчайших расстояний.

Это может привести к значительному увеличению объема памяти, требуемой для хранения матрицы достижимости. Необходимо отсеивать похожие варианты, оставляя один из них с целью сократить объем, занимаемый матрицей.

По большому счету, сравнение редко вызывает затруднение, если оценку пути можно свести к единственному числовому значению. Однако если путь разбит на независимые сегменты, объединить и суммировать которые невозможно или нежелательно, то задача усложняется.

Примером такого пути может быть путешествие, когда, например, сначала нужно лететь 3 часа на самолете, а затем 15 часов на поезде — вариант 1, и вариант 2 — 5 часов на самолете. Можно, конечно, суммировать время и сказать, что первый вариант отнял 18 часов, но это будет нечестно, потому что противопоставляются 15 часов на поезде и 2 часа на самолете.

Сегменты, идущие друг за другом, в этом случае легко представить в виде вектора — одномерного массива. Множество решений из сегментов образуют матрицу — двумерный массив. Если же разные решения могут состоять из разного количества сегментов, то множество решений образуют зубчатый массив.

Поэтому задачу можно свести к сортировке зубчатого (ступенчатого) массива. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной [4].

Описание задачи

Для того чтобы представить область применения подхода к сортировке зубчатых массивов, рассмотрим прикладную задачу, в которой транспортными средствами осуществляются перевозки грузов.

Допустим, требуется осуществить доставку груза из пункта А в пункт Б, но при этом прямого сообщения между пунктами нет, и нужно осуществлять доставку груза через промежуточный пункт. Таких промежуточных пунктов может быть несколько, некоторые из них могут быть взаимозаменяемыми. Варианты с взаимозаменяемыми пунктами похожи друг на друга и отличаются лишь расстоянием, пройденным до этих пунктов.

Необходим алгоритм выбора, отсеивающий похожие друг на друга варианты. К примеру, программа должна возвращать ответ, при котором передача груза осуществляется на ближайшем допустимом пункте. Задача будет рассмотрена на примере целочисленных положительных значений одного порядка. В случае, если порядок заранее неизвестен или значения могут быть вещественными или отрицательными, для того, чтобы применение описанных подходов стало возможным, необходимо сначала провести оценку всех значений зубчатого массива, а затем модифицировать значения массива так, чтобы сортировка стала возможной.

Сортировка зубчатых массивов

Хорошим решением может быть способ, который основан на предварительном генерировании ключа для каждой строки, который будет конкатенацией ключей каждой ячейки. Конкатенация — операция склеивания объектов линейной структуры [2].

Затем берется строковое представление сгенерированного ключа, по которому происходит сортировка [1], результат сортировки по ключу представлен в таблице 1. В рассматриваемом примере значения в ячейках — это расстояние, которое следует проехать до пункта перегрузки.

Таблица 1

Сортировка по ключу

Исходный массив

3	4		“34”
4	3		“43”
9			“9”
1	2	1	“121”

Table 1

Sort by key

Результат сортировки массива

1	2	1	“121”
3	4		“34”
4	3		“43”
9			“9”

Таким образом, представляя ключ-строку, определяется приоритет первых ячеек над последними. Сортировка по возрастанию гарантирует, что участки, которым соответствуют первые ячейки, будут стремиться к минимуму. Т. е. передача груза будет проходить на ближайшем пункте.

Рассмотрим существующий пример и допустим, что есть еще пара строк, с весами “1” и “99”. Добавим их в массив и отсортируем его (результат сортировки представлен в таблице 2).

Таблица 2

Сортировка по ключу

Table 2

Sort by key

Результат сортировки массива

1			“1”
1	2	1	“121”
3	4		“34”
4	3		“43”
9			“9”
9	9		“99”

Можно заметить, что, даже несмотря на то, что общая длина ключа не играет первостепенной роли, она все же учитывается. Т. е. “1” всегда будет меньше, чем “121”, а “99” все же больше, чем “9”. Такую сортировку можно описать правилом: «Начальные сегменты стремятся к минимуму, чем общая длина ключа меньше — тем лучше».

Это как раз то, что нужно, если ищется удобный способ транспортировки. Но в других задачах возможны другие требования.

Это правило действует для одной предметной области, в других предметных областях может потребоваться сортировать по убыванию или установить приоритет длины ключа выше, чем у сегментов.

В общем случае предлагаемый алгоритм сортировки зубчатых массивов представлен на рис. 1.

Шаги с первого по седьмой обеспечивают генерацию ключей массива, и их вычислительная сложность оценивается как количество всех элементов в массиве, по формуле (2):

$$O\left(\sum_{i=1}^{n} m_i\right), \tag{2}$$

где n — количество строк зубчатого массива, m_i — количество элементов в строке i.

Рассмотрим приведенный алгоритм на примере упомянутой выше задачи по транспортировке грузов. Поскольку грузы могут доставляться различными видами транспорта, то в некоторых случаях может потребоваться перемещение груза с одного вида транспорта на другой (или тот же вид транспорта, но без прямого сообщения, например, в местах пересечения различных железнодо-

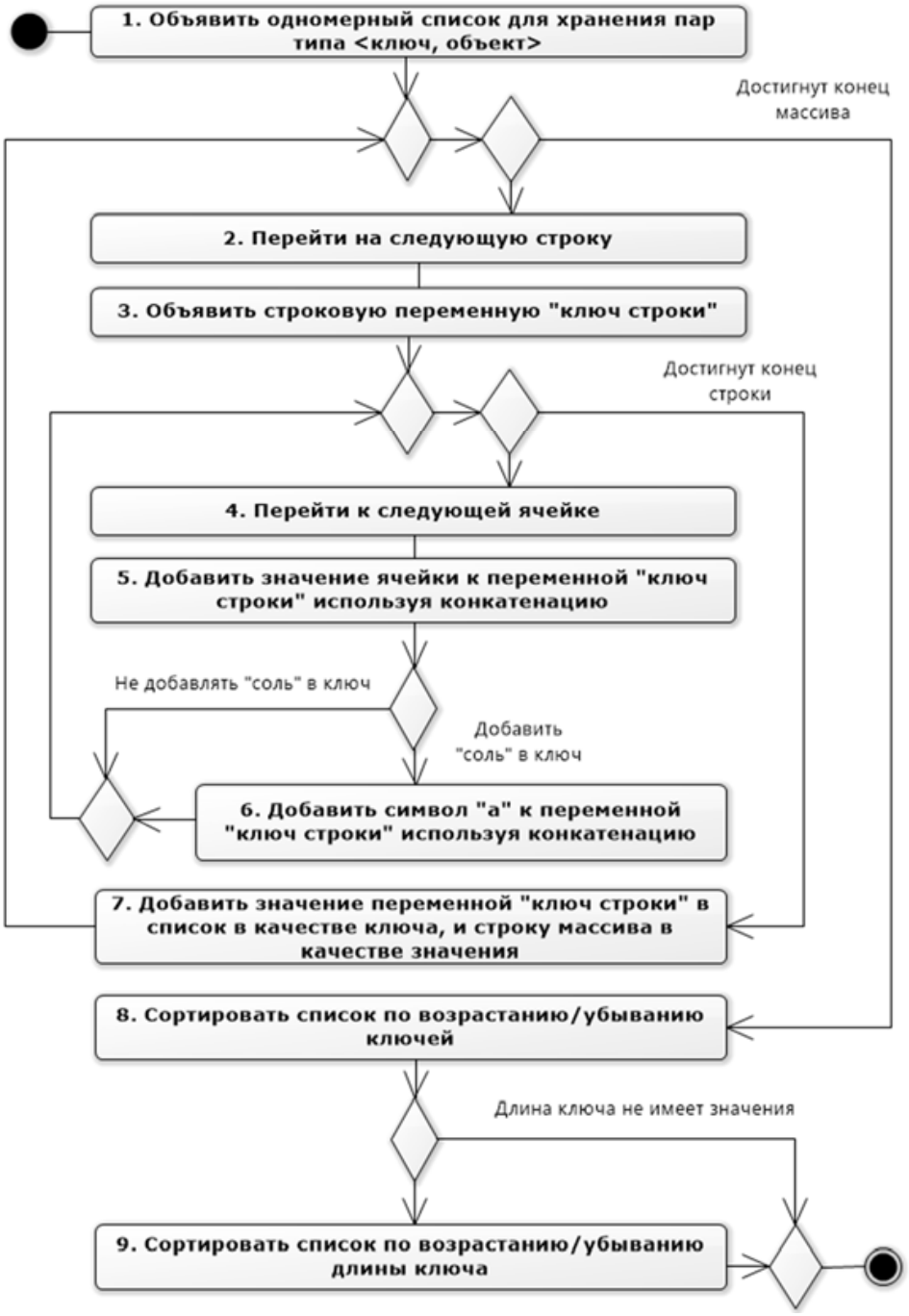

Рис. 1. Общий алгоритм сортировки зубчатых массивов

Fig. 1. General algorithm for sorting cogged arrays

рожных веток), для упрощения будем называть это перегрузкой. А пункт, в котором это происходит, — пунктом перегрузки, или узлом.

Формализуем задачу. Пусть потребуется упорядочить N элементов: R_1, R_2, ..., R_n. Каждый элемент представляет из себя вектор элементов $R_1: \{F_{11}, F_{12}, \ldots, F_{1m_1}\}$, $R_2: \{F_{21}, F_{22}, \ldots, F_{2m_2}\}, \ldots, R_n: \{F_{n1}, F_{n2}, \ldots, F_{nm_n}\}$. Каждый элемент вектора представляет собой запись F_{ij} и ключ K_{ij}. Конкатенация строковых представлений ключей F_{ij} является ключом вектора R_i; обозначим ключ как K_i. Ключ характеризуется двумя параметрами: длиной ключа — кратной количеству элементов m_i вектора R_i – *length* (K_i), и значением ключа K_i.

Задача сортировки формулируется в зависимости от выбранного правила сортировки.

Всего приведенный алгоритм можно использовать для исполнения восьми различных правил:

Правило № 1: начальные сегменты стремятся к минимуму, чем общая длина ключа меньше — тем лучше.

Задачей сортировки является нахождение такой перестановки записей p (1), p (2), ..., p (n) с индексами {1, 2, ..., N}, после которой ключи расположились бы в порядке неубывания: $K_{p(1)} \le K_{p(2)} \le \ldots \le K_{p(n)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Этот пример уже был рассмотрен выше. Для его достижения шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по возрастанию, шаг 9 — пропускаем, результат сортировки представлен в таблице 3.

Таблица 3 *Table 3*

Сортировка по ключу **Sort by key**

Результат сортировки массива			
1			“1”
1	2	1	“121”
3	4		“34”
4	3		“43”
9			“9”
9	9		“99”

Для примера с транспортировкой грузов такая сортировка будет означать, что чем короче дистанция между пунктами — тем лучше, т. е. перегрузка осуществляется при первой же возможности, однако стоит совершить перегрузку — и можно ездить очень долго, а самих узлов может быть неограниченно много, и это окажет лишь малое воздействие на результат.

Правило № 2: начальные сегменты стремятся к максимуму, чем общая длина ключа меньше — тем лучше.

Задачей сортировки является нахождение такой перестановки записей p (1), p (2), ..., p (n) с индексами {1, 2, ..., N}, после которой ключи расположились бы в порядке невозрастания: $K_{p(1)} \ge K_{p(2)} \ge \ldots \ge K_{p(n)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается больше любого символа.

Для его достижения шаг 6 на рис. 1 выполняем, на шаге 8 сортируем по убыванию, шаг 9 — пропускаем. Суть шага 6 в том, что в конец каждого ключа добавляется «соль». «Соль» — какой-нибудь символ, который имеет ASCII код заведомо больший, чем у любой цифры. Сразу после цифр идут символы: «:», «;», «<», «=», «>», «?», «@», латинский алфавит и т. д. Например, можно добавлять в конец каждого ключа латинскую букву «а», это обеспечивает выполнение правила «отсутствующий символ считается больше любого символа»; т. е. мы вместо отсутствующего добавляем новый символ, который всегда больше любого символа ключа [5]. Результат сортировки представлен в таблице 4.

Таблица 4 *Table 4*

Сортировка по ключу **Sort by key**

Результат сортировки массива

9			“9a”
9	9		“99a”
4	3		“43a”
3	4		“34a”
1			“1a”
1	2	1	“121a”

В этом случае перегрузка будет осуществляться на последнем из возможных пунктов, а если есть возможность доехать до самого удаленного пункта, то она обязательно окажется в приоритете. Но чем меньше самих перегрузок — тем лучше.

Правило № 3: начальные сегменты стремятся к минимуму, чем общая длина ключа больше — тем лучше.

Задачей сортировки является нахождение такой перестановки записей $p(1)$, $p(2)$, ..., $p(n)$ с индексами $\{1, 2, ..., N\}$, после которой ключи расположились бы в порядке неубывания: $K_{p(1)} \le K_{p(2)} \le ... \le K_{p(n)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается больше любого символа.

Для его достижения шаг 6 на рис. 1 выполняем, на шаге 8 сортируем по возрастанию, шаг 9 — пропускаем. Результат сортировки представлен в таблице 5.

Таблица 5 *Table 5*

Сортировка по ключу **Sort by key**

Результат сортировки массива

1	2	1	“121a”
1			“1a”
3	4		“34a”
4	3		“43a”
9	9		“99a”
9			“9a”

Перегрузка осуществится, опять же, при первой удачной возможности, но при этом выбираются варианты с посещением наибольшего количества узлов.

Правило № 4: начальные сегменты стремятся к максимуму, чем общая длина ключа больше — тем лучше.

Задачей сортировки является нахождение такой перестановки записей $p(1)$, $p(2)$, ..., $p(n)$ с индексами $\{1, 2, ..., N\}$, после которой ключи расположились бы в порядке невозрастания: $K_{p(1)} \geq K_{p(2)} \geq ... \geq K_{p(n)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по убыванию, шаг 9 — пропускаем. Результат сортировки представлен в таблице 6.

Таблица 6 *Table 6*

Сортировка по ключу **Sort by key**

Результат сортировки массива

9	9		“99”
9			“9”
4	3		“43”
3	4		“34”
1	2	1	“121”
1			“1”

В этом случае перегрузка будет осуществляться на последнем из возможных пунктов, а из всех равноудаленных вариантов будет выбран тот, на котором больше всего посещений различных узлов.

Правило № 5: чем общая длина ключа меньше — тем лучше, начальные сегменты стремятся к минимуму.

Задачей сортировки является нахождение такой перестановки записей $p(1)$, $p(2)$, ..., $p(n)$ с индексами $\{1, 2, ..., N\}$, после которой длины ключей расположились бы в порядке неубывания: $length(K_{p(1)}) \leq length(K_{p(2)}) \leq ... \leq length(K_{p(n)})$.

Элементы с равными длинами ключей $length(K_{p(j)}) = length(K_{p(j+1)})$ упорядочиваются в порядке неубывания $K_{p(j)} \leq K_{p(j+1)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по возрастанию, на шаге 9 — по возрастанию. Важно, чтобы алгоритм сортировки был устойчивым. Сортировка называется устойчивой, если она удовлетворяет такому дополнительному условию, что записи с одинаковыми ключами остаются в прежнем порядке [1], формула (3):

$$p(i) < p(j) \text{ для любых } K_{p(i)} = K_{p(j)} \text{ и } i < j, \quad (3)$$

где $p(1), p(2), ..., p(n)$ — записи с индексами $\{1, 2, ..., N\}$ и ключами $K_{p(1)}, K_{p(2)}, ..., K_{p(n)}$.

Результат сортировки представлен в таблице 7.

Таблица 7 *Table 7*

Сортировка по ключу **Sort by key**

Результат сортировки массива

1			“1”
9			“9”
3	4		“34”
4	3		“43”
9	9		“99”
1	2	1	“121”

Это наиболее подходящий для транспортной системы вариант сортировки. Из всех возможных вариантов приоритет будет у тех, где меньше всего посещения транспортных узлов, а из вариантов с одинаковым количеством узлов будет выбран самый короткий. Если же обязательно требуется перегрузка, то она будет осуществлена на первом пригодном для этого узле.

Правило № 6: чем общая длина ключа больше — тем лучше, начальные сегменты стремятся к минимуму.

Задачей сортировки является нахождение такой перестановки записей $p\,(1)$, $p\,(2)$, ..., $p\,(n)$ с индексами $\{1, 2, ..., N\}$, после которой длины ключей расположились бы в порядке невозрастанию: $length\,(K_{p\,(1)}) \geq length\,(K_{p\,(2)}) \geq ... \geq length\,(K_{p\,(n)})$.

Элементы с равными длинами ключей $length\,(K_{p\,(j)}) = length\,(K_{p\,(j+1)})$ упорядочиваются в прядке неубывания $K_{p\,(j)} \geq K_{p\,(j+1)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по возрастанию, на шаге 9 — по убыванию. Результат сортировки представлен в таблице 8.

Таблица 8 *Table 8*

Сортировка по ключу **Sort by key**

Результат сортировки массива

1	2	1	“121”
3	4		“34”
4	3		“43”
9	9		“99”
1			“1”
9			“9”

Из всех вариантов наибольший приоритет будет у того, где максимальное количество транспортных узлов, а сама перегрузка осуществится на первом пригодном для этого пункте.

Правило № 7: чем общая длина ключа меньше — тем лучше, начальные сегменты стремятся к максимуму.

Задачей сортировки является нахождение такой перестановки записей p (1), p (2), ..., p (n) с индексами {1, 2, ..., N}, после которой длины ключей расположились бы в порядке неубывания: $length\ (K_{p\,(1)}) \le length\ (K_{p\,(2)}) \le ... \le length\ (K_{p\,(n)})$.

Элементы с равными длинами ключей $length\ (K_{p\,(j)}) = length\ (K_{p\,(j+1)})$ упорядочиваются в порядке невозрастания $K_{p\,(j)} \ge K_{p\,(j+1)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по убыванию, на шаге 9 — по возрастанию. Результат можно представить в таблице 9.

Таблица 9 *Table 9*

Сортировка по ключу **Sort by key**

Результат сортировки массива

9			“9”
1			“1”
9	9		“99”
4	3		“43”
3	4		“34”
1	2	1	“121”

Этот вариант тоже может подойти для транспортной системы. Здесь также приоритет у вариантов с наименьшим количеством транспортных узлов, а если перегрузка необходима, то она будет осуществлена на последнем пригодном для этого пункте. Однако из вариантов без перегрузок приоритет будет выше у того, который длиннее.

Правило № 8: чем общая длина ключа больше — тем лучше, начальные сегменты стремятся к максимуму.

Задачей сортировки является нахождение такой перестановки записей p (1), p (2), ..., p (n) с индексами {1, 2, ..., N}, после которой длины ключей расположились бы в порядке невозрастания: $length\ (K_{p\,(1)}) \ge length\ (K_{p\,(2)}) \ge ... \ge length\ (K_{p\,(n)})$.

Элементы с равными длинами ключей $length\ (K_{p\,(j)}) = length\ (K_{p\,(j+1)})$ упорядочиваются в порядке невозрастания $K_{p\,(j)} \ge K_{p\,(j+1)}$.

Ключи сортируются по правилу сортировки строк, и отсутствующий символ считается меньше любого символа.

Шаг 6 на рис. 1 пропускаем, на шаге 8 сортируем по убыванию, на шаге 9 — по убыванию. Результат можно представить в таблице 10.

Таблица 10 *Table 10*

Сортировка по ключу **Sort by key**

Результат сортировки массива

1	2	1	“121”
9	9		“99”
4	3		“43”
3	4		“34”
9			“9”
1			“1”

Из всех вариантов наибольший приоритет будет у того, где максимальное количество перегрузок, а само перемещение груза осуществится на последнем пригодном для этого пункте.

Веса ячеек могут иметь различную разрядность. Представляя ключ ячейки строкой, можно просто добавлять нули в начало тех ячеек, где разрядность меньше максимальной. Таким образом, все определенные правила будут также хорошо работать. Например, правило № 5 «начальные сегменты стремятся к минимуму, чем общая длина ключа меньше — тем лучше», можно представить в таблице 11.

Таблица 11

Сортировка по ключу

Table 11

Sort by key

Результат сортировки массива

1			“01”
1	2	1	“010201”
3	4		“0304”
4	3		“0403”
9			“09”
9	9		“0909”
12			“12”

Как видно, строки для этого подходят как нельзя лучше. Необходимо только определить максимальную разрядность в массиве. Например, найдя максимальное по модулю число, переведя его в строку и вернув длину получившейся строки. Либо, если мы заранее знаем, что значения не могут быть больше, например, 247, то сразу понятно, что максимальная разрядность чисел 3, и достаточно добавить один или 2 нуля на шаге 5, там, где это необходимо.

Если необходимо учесть наличие отрицательных или дробных весов, то можно привести их к целым положительным. Например, прибавлением ко всем весам матрицы максимального по модулю отрицательного числа и умножением всех весов на 10n, где n — наибольшее количество разрядов после запятой.

Хорошим примером для использования алгоритмов сортировки зубчатых массивов может стать газотранспортная система, поскольку по мере движения газа по магистральному трубопроводу давление уменьшается, так как потенциальная энергия расходуется на преодоление гидравлических сопротивлений. Используя эти подходы, можно сортировать возможные схемы транспортировки по расстоянию между соседними компрессорными станциями. Хотя стоит заметить, что в этом случае одни участки все-таки будут иметь приоритет над другими, например, расстояние от хранилища до первой компрессорной станции будет более значимо, чем от первой станции до второй и т. д.

Для этого примера подойдет сортировка по правилу «чем общая длина ключа больше — тем лучше, начальные сегменты стремятся к минимуму». Коли-

чество элементов в строке будет соответствовать количеству компрессорных станций + 1, а в качестве значений ячеек можно использовать расстояние (скажем, в километрах) до этой станции (в последней ячейке — расстояние до газораспределительной станции, например, или хранилища). Сортировка обеспечит приоритет тех вариантов, где количество компрессорных станций больше, а расстояние между ними — меньше, что будет иметь значение для поддержания давления.

Также можно в качестве ячеек представить количество газораспределительных станций и хранилищ, а значениями ячеек — количество компрессорных станций на участке магистрали между хранилищами и газораспределительными станциями. Тогда подойдет правило «чем общая длина ключа меньше — тем лучше, начальные сегменты стремятся к максимуму»: такое правило обеспечит приоритет тех маршрутов, где количество хранилищ и газораспределительных станций меньше, т. е. меньше транзитных издержек, а компрессорных станций — больше, что, опять же, снизит затраты на поддержание давления.

Заключение

В статье приводятся принципы построчной сортировки зубчатых массивов, основанных на предварительной генерации ключа строки, для оценки и ранжирования решений транспортных задач. Всего приведено и описано восемь различных принципов сортировки, обеспечивающих разный результат в зависимости от задачи сортировки, приведены шаги алгоритма, обеспечивающие достижение заявленного результата.

В результате работы можно сделать выводы, что для описанных алгоритмов вычислительная сложность подходов 5-8 будет больше, чем у подходов 1-4, т. к. 5-8 требуют дополнительной сортировки по ключу (шаг 9). У подходов 2-3 также выше вычислительная сложность, чем у 1 и 4, т. к. 2 и 3 требуют дополнительного символа «соль» (шаг 6), в результате чего увеличивается не только количество шагов, но и длина ключа. Таким образом, наименьшее количество шагов требуется для подходов 1 и 4, соответственно, они обеспечивают наилучшую скорость сортировки.

Алгоритмы эффективны при заранее известном порядке, на целочисленных положительных значениях. Данные подходы характеризуются тем, что не требуют предварительной оценки значений массивов, что делает их простыми в применении и реализации. Однако при наличии отрицательных числовых значений и/или значений с плавающей запятой подходы требуют предварительной обработки и модификации массивов, что снижает их эффективность.

СПИСОК ЛИТЕРАТУРЫ

1. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск / Д. Э. Кнут. Москва: Вильямс, 2007. 832 с.
2. Конкатенация. URL: https://ru.wikipedia.org/wiki/Конкатенация (дата обращения: 29.06.2017).
3. Кормен Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. 2-е изд. М.: Вильямс, 2006. 1296 с.
4. Шилдт Г. C# 4.0. Полное руководство / Г. Шилдт. Москва: Вильямс, 2011. 1056 с.
5. American National Standard for Information Systems. Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII) ANSI-X3.4-1986 (R 2007), 2007. 31 pp.

Ivan N. TKACHENKO[1]
Mikhail V. GRIGOREV[2]
Inna I. GRIGOREVA[3]

SORTING JAGGED ARRAYS TO ASSESS SOLUTIONS TO TRANSPORTATION PROBLEMS

[1] Senior Software Engineer, Ltd. «Technocom» (Tyumen)
tkachenko@technocom.tech

[2] Cand. Sci. (Tech.), Associate Professor,
Department of Program and System Engineering,
Tyumen State University
m.v.grigorev@utmn.ru

[3] Cand. Sci. (Tech.), Associate Professor,
Department of Program and System Engineering,
Tyumen State University
i.i.grigoreva@utmn.ru

Abstract

This article explores the ways of sorting the jagged arrays for ordering rows in descending order by two criteria. A jagged array is a representation of the solutions of the problem of finding the optimal path, in which each row is one of the solutions. The purpose of sorting is to select the most significant solutions and then reduce the number of objects stored in the reachability matrix. Reducing the number of objects does not affect the dimensionality of the reachability matrix, but reduces the amount of memory needed to store the cells of the matrix. The task is described on the example of assessing the solutions to the transport task of cargo delivery.

In the article, when sorting rows of jagged arrays, the priority of some cells of the array row is assumed over the others, that is, the first cell exerts a greater value on the result than the second, and the second cell has more than the third, etc.

The result of the study is the formalization of the rules of eight different approaches to sorting — by the number of variants of combining the two sorting criteria, the algorithm is given and the estimation of computational complexity is given. The criteria for sorting are

Citation: Tkachenko I. N., Grigorev M. V., Grigoreva I. I. 2017. “Sorting Jagged Arrays to Assess Solutions to Transportation Problems”. Tyumen State University Herald. Physical and Mathematical Modeling. Oil, Gas, Energy, vol. 3, no 2, pp. 100-114.
DOI: 10.21684/2411-7978-2017-3-2-100-114

the key length and key value. All eight approaches provide different sorting results. For each approach, the operations necessary to achieve the desired result are described and analyzed with provided examples. In each case, a problem is formulated and formalized.

The algorithm for sorting the jagged arrays is based on generating a text representation of the row key. The article only considers sorting rows, sorting columns in the article is not considered.

Conclusions are drawn about which of the described approaches provide the best sorting speed and why. On the example of the problem of cargo transportation, it is described to what effect the choice of this or that approach to sorting will result.

Keywords

Sorting jagged arrays, solving the transport problem.

DOI: 10.21684/2411-7978-2017-3-2-100-114

REFERENCES

1. Knut D. E. 2007. Iskusstvo programmirovaniya [The Art of Computer Programming], vol. 3. Sortirovka i poisk [Sorting and Searching]. Moscow: Williams.
2. Concatenation. Accessed on June 29, 2017. https://en.wikipedia.org/wiki/Concatenation
3. Corman Th. H., Leisserson Ch. I., Rivest R. L., Stein C. 2006. Algoritmy: postroyeniye i analiz [Introduction to Algorithms]. 2nd ed. Moscow: Williams.
4. Schildt H. 2011. C# 4.0. Polnoye rukovodstvo [C # 4.0. The Complete Reference]. Moscow: Williams.
5. American National Standard for Information Systems. Coded Character Sets — 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII) ANSI-X3.4-1986 (R 2007), 2007.