

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Тема 5 Сортировка обменом



Outline

- 1 Задача сортировки
- 2 Классификация алгоритмов сортировки
- 3 Пузырьковая сортировка
- 4 Быстрая сортировка
- 5

Тема 5 Сортировка обменом

1 Задача сортировки



Постановка задачи

Исходные данные: последовательность a из n элементов; если специально не оговорено иное будем полагать, что элементы a_i последовательности a – это целые положительные числа.

Требования: найти перестановку элементов a , для которой выполняться заданное отношение порядка

$$a_1 \leq a_2 \leq \dots \leq a_n, \text{ или} \quad (1)$$

$$a_1 \geq a_2 \geq \dots \geq a_n. \quad (2)$$

По умолчанию сортируем по возрастанию (1).

Если специально не оговорено иное полагаем, что исходные



Актуальность

Для чего нужна сортировка?

Для обеспечения эффективности извлечения и обработки данных. И, прежде всего, для поиска.

В этом отношении говоря об эффективности следует всегда помнить о том, что общая трудоемкость T массовой обработки массива данных оценивается величиной

$$T = T_{\text{sorting}} + k \cdot (T_{\text{search}} + T_{\text{processing}}),$$

T_{sorting} – трудоемкость сортировки массива данных;

k – общее число обращений к массиву данных;

T_{search} – трудоемкость поиска в массиве данных;

$T_{\text{processing}}$ – трудоемкость обработки массива данных.



Актуальность

Упорядочение данных окупиться с лихвой. Потому, что поиск данных в упорядоченном массиве на порядки быстрее, чем в массиве неупорядоченном. В особенности для **Big Data**.

Для чего еще изучать сортировку?

Помимо практической значимости алгоритмы сортировки по праву считаются фундаментом ТА, поскольку в них заложено поистине огромное количество разнообразных **эффективных приемов обработки данных**. Изучение которых позволит успешно конструировать и применять новые алгоритмы обработки данных.



Виды сортировки

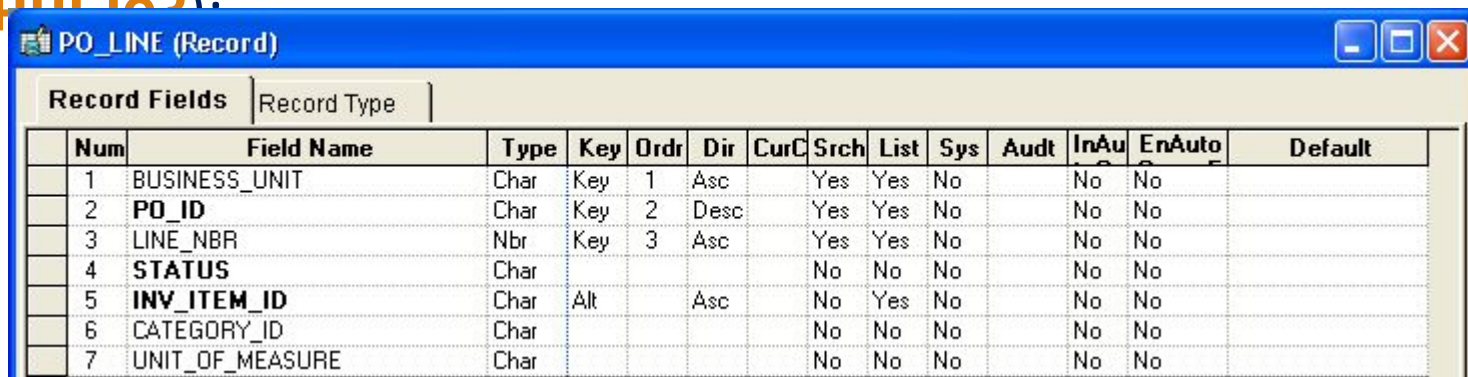
Алгоритм сортировки называется алгоритмом **внутренней сортировки**, если сортируемые данные и программный код, используемый для обработки этих данных, одновременно помещаются в оперативной памяти. Если это условие не выполняется, алгоритм сортировки называется алгоритмом **внешней сортировки**.

В наши дни дефицит оперативной памяти явление достаточно редкое. Поэтому, если специально не оговорено иное, будем полагать, что рассматриваемые алгоритмы относятся к классу **алгоритмов внутренней сортировки**.



Записи и их ключи

Строго говоря сортируемые элементы – это некоторые **записи**, каждая из которых имеет **ключ**, управляющий процессом сортировки и данные (Кнут Д., 295 с.). Такое определение пришло из БД; оно связано с необходимостью обработки таблиц (**где: записи, их ключи и данные?**).



Num	Field Name	Type	Key	Ordr	Dir	CurC	Srch	List	Sys	Audt	InAu	EnAuto	Default
1	BUSINESS_UNIT	Char	Key	1	Asc		Yes	Yes	No		No	No	
2	PO_ID	Char	Key	2	Desc		Yes	Yes	No		No	No	
3	LINE_NBR	Nbr	Key	3	Asc		Yes	Yes	No		No	No	
4	STATUS	Char					No	No	No		No	No	
5	INV_ITEM_ID	Char	Alt		Asc		No	Yes	No		No	No	
6	CATEGORY_ID	Char					No	No	No		No	No	
7	UNIT_OF_MEASURE	Char					No	No	No		No	No	

https://docs.oracle.com/cd/E41633_01/pt853pbh1/eng/pt/tapd/img/ia2cf27cn-7792.png

Если специально не оговорено иное **под элементами** будем **понимать** некоторую **числовую**



Тема 5
Сортировка
обменом

2
Классификаци
я алгоритмов
сортировки



Стандартная классификация

- ❑ Сортировка **обменом** (пузырьковая сортировка, шейкерная сортировка, быстрая сортировка ...).
- ❑ Сортировка **выбором** (линейная сортировка, выбор на дереве, пирамидальная сортировка ...).
- ❑ Сортировка **распределением** (поразрядная сортировка ...).
- ❑ Сортировка **слиянием** (двух путевое слияние ...).
- ❑ Сортировка **вставкой** (простая вставка, бинарная вставка ...).
- ❑ *Сортировка **подсчетом**, когда классическая сортировка (процедура перестановки элементов местами) заменяется процедурами подсчета частоты элементов и их вывода в виде упорядоченной

Стандартная классификация

Приведенная классификация **условна**, поскольку алгоритмы могут относиться к нескольким классам одновременно. Поскольку работают на основе нескольких принципов.

Помимо указанных выше существуют немало модификаций и дополнительных классификаций алгоритмов сортировки.

Источники и ресурсы по теме

- Дональд Э. Кнут Искусство программирования. Том 3. Сортировка и поиск <https://www.twirpx.com/file/32943/>
- Коллекция сортировок <http://sorting.valemak.com/radix/>
- Тема 1. Литература и ресурсы

Сегодня займемся рассмотрением алгоритмов

Сортировки обменом



Тема 5 Сортировка обменом

3 Пузырьковая сортировка



Идея алгоритма

Пожалуй, наиболее очевидный способ обменной сортировки – сравнивать попарно смежные элементы, $(a_i \leq a_{i+1})$, слева направо: первый со вторым, второй с третьим и так далее. Если порядок в паре неверный, $(a_i > a_{i+1})$, элементы меняются местами. Таким образом, наибольшие элементы продвигаются вправо – к концу массива, а наименьшие – к его началу.

В конце итерации наибольший элемент из неупорядоченной части массива занимает свое место в конце этой неупорядоченной части.

Повторив такую итерацию сравнений и обменов $n - 1$ раз (где n – число элементов массива) получим упорядоченный массив.

Рассмотрим пример.



Иллюстрация работы алгоритма

Внешняя итерация	Состояние массива
Исходный массив	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4
1	1, 3, 3, 3, 4, 5, 6, 6, 0, 4, 7
2	1, 3, 3, 3, 4, 5, 6, 0, 4, 6, 7
3	1, 3, 3, 3, 4, 5, 0, 4, 6, 6, 7
4	1, 3, 3, 3, 4, 0, 4, 5, 6, 6, 7
5	1, 3, 3, 3, 0, 4, 4, 5, 6, 6, 7
6	1, 3, 3, 0, 3, 4, 4, 5, 6, 6, 7
7	1, 3, 0, 3, 3, 4, 4, 5, 6, 6, 7
8	1, 0, 3, 3, 3, 4, 4, 5, 6, 6, 7
9*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7

Синим выделены элементы исходного массива, **жирным** – упорядоченная последовательность его элементов, **зеленым (жёлтым)** – проверяемая пара смежных элементов, для которой выполняется (не выполняется) отношение порядка: $a_i \leq a_{i+1}$.

Внутренняя итерация	Состояние массива
10*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
	Выход элемента на свое место
1.1	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4
1.2	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4
1.3	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4 → 1, 3, 3, 5, 3, 4, 6, 6, 7, 0, 4
1.4	1, 3, 3, 5, 3, 4, 6, 6, 7, 0, 4 → 1, 3, 3, 3, 5, 4, 6, 6, 7, 0, 4
1.5	1, 3, 3, 3, 5, 4, 6, 6, 7, 0, 4 → 1, 3, 3, 3, 4, 5, 6, 6, 7, 0, 4
1.6	1, 3, 3, 3, 4, 5, 6, 6, 7, 0, 4
1.7	1, 3, 3, 3, 4, 5, 6, 6, 7, 0, 4
1.8	1, 3, 3, 3, 4, 5, 6, 6, 7, 0, 4
1.9	1, 3, 3, 3, 4, 5, 6, 6, 7, 0, 4 → 1, 3, 3, 3, 4, 5, 6, 6, 0, 7, 4
1.10	1, 3, 3, 3, 4, 5, 6, 6, 0, 7, 4 → 1, 3, 3, 3, 4, 5, 6, 6, 0, 4, 7



Адекватность сортировки

Сортировка считается **адекватной** при одновременном выполнении трех условий:

- 1) количество элементов в исходном и отсортированном массивах должно совпадать;
- 2) поэлементный состав исходного и отсортированного массивов должен совпадать;
- 3) для элементов отсортированного массива a должно выполняться заданное отношение порядка по возрастанию или по убыванию

$$a_1 \leq a_2 \leq \dots \leq a_n, \text{ или} \quad (1)$$

$$a_1 \geq a_2 \geq \dots \geq a_n. \quad (2)$$



Модификации

- 1) Сортировка **по убыванию** (проверяем отношение $a_i \geq a_{i+1}$).
- 2) Изменение **направления просмотра** массива.

Внешняя итерация	Состояние массива
Исходный массив	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4
1	1, 3, 3, 3, 4, 5, 6, 6, 0, 4, 7
2	0, 1, 3, 3, 3, 4, 5, 6, 6, 4, 7
3	0, 1, 3, 3, 3, 4, 5, 6, 4, 6, 7
4*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
5*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
6*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
7*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
8*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
9*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
10*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7

- 3) **Шейкерная сортировка**
(или двунаправленная)

Внешняя итерация	Состояние массива
Исходный массив	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4
1	1, 3, 3, 3, 4, 5, 6, 6, 0, 4, 7
2	0, 1, 3, 3, 3, 4, 5, 6, 6, 4, 7
3	0, 1, 3, 3, 3, 4, 5, 6, 4, 6, 7
4*	0, 1, 3, 3, 3, 4, 4, 5, 6, 6, 7
5*	Обменов нет → ОСТАНОВ
6*	
7*	
8*	
9*	
10*	

- 4) **Остановка по готовности**



Задание

Отсортировать массив:

1) пузырьком и 2) шейкером с остановкой по готовности.

Внешняя итерация	Состояние массива
Исходный массив	7, 4, 6, 2, 1, 0, 3, 4, 9, 8, 8, 5, 1
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Внешняя итерация	Состояние массива
Исходный массив	7, 4, 6, 2, 1, 0, 3, 4, 9, 8, 8, 5, 1
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	



Отказываемся от обмена!

Алгоритм обмена с временным хранилищем

$$p = x$$

$$x = y$$

$$y = p$$

Арифметический алгоритм обмена переменных целого типа

$$x = x + y$$

$$y = x - y$$

$$x = x - y$$

Этот алгоритм будет прерываться на системах, которые перехватывают переполнение целого.

https://ru.wikipedia.org/wiki/Алгоритм_обмена_при_помощи_исключающего_ИЛИ



Отказываемся от обмена?

Используя алгоритм обмена при помощи исключающего ИЛИ не требуется ни временное хранилище, ни определение переполнения. Алгоритм в данном случае таков

$$X = X \text{ XOR } Y$$

$$Y = Y \text{ XOR } X$$

$$X = X \text{ XOR } Y$$

https://ru.wikipedia.org/wiki/Алгоритм_обмена_при_помощи_исключающего_ИЛИ

**Стоит ли уходить от простого обмена
с временным хранилищем???**



Вывод оценок трудоемкости

Давайте оценим трудоемкость сортировки применением пузырька, шейкера и шейкера с остановкой по готовности.

Кто желает помочь?



Оценки трудоемкости

Трудоемкость пузырька (по порядку величины)

$$T(n) = n^2.$$

Трудоемкость шейкера с остановкой по готовности

1) **лучший случай** (массив упорядочен, требуется один проход)

$$T(n) = n;$$

2) **худший случай** (массив упорядочен в обратном порядке, требуется $n - 1$ проход)

$$T(n) = n^2.$$



Достоинства и недостатки

Рассматривается шейкер с остановкой по готовности.

Достоинства:

- ❑ применим для любых чисел
- ❑ не требуются вспомогательные структуры данных
- ❑ достаточно высокая эффективность реализации на:
 - значительно упорядоченных выборках,
 - малых выборках, так как не требуется обслуживать вспомогательные структуры данных; и это несмотря на квадратичную трудоемкость.

Недостаток – квадратичная трудоемкость (плохо на больших, и тем более, – на неупорядоченных выборках).



Теория и эксперимент

В первом приближении трудоемкость оценивается по порядку величины лишь **числом операций сравнения** без учета операций тела цикла. При этом для многих алгоритмов характерна ситуация наличия **двух граничных оценок трудоемкости, отвечающих лучшему и худшему случаю**; что будет в конкретной ситуации заранее неизвестно.

Поэтому перед практическим использованием алгоритма сортировки целесообразно экспериментально уточнить оценку трудоемкости (числом арифметических операций) и произвести сравнительный анализ его трудоемкости с аналогами. Причем для статистически значимой выборки данных.

В такой ситуации можно объективно выбрать наилучший



Тема 5 Сортировка обменом

4 Быстрая сортировка



Идея алгоритма

Вначале сравнивают первый и последний элементы ($a_i \leq a_j$, $i = 0$, $j = n - 1$). Если порядок в паре верный, j уменьшается на единицу, первый элемент сравнивается с предпоследним и т.д.; иначе, ($a_i > a_j$), элементы меняются местами и свечка сжигается с другого конца – теперь при сравнениях не j уменьшается, а i увеличивается на единицу. Если в ходе сравнений произойдет еще один обмен, опять будем уменьшать j и т.д. В конце такой итерации индексы сойдутся в некотором элементе a_k . Этот элемент занял свое место; слева от него массив элементов не больших a_k , а справа – массив элементов не меньших a_k . Эти массивы сортируют аналогично. Так продолжается до тех пор, пока массивы не превратятся в элементы. Рассмотрим пример.



Иллюстрация работы алгоритма

Внешняя итерация	Состояние массива	Стек
Вход	1, 3, 5, 3, 3, 4, 6, 6, 7, 0, 4	[0, 10]
1	0, 1 , 5, 3, 3, 4, 6, 6, 7, 3, 4	[0, 0], [2, 10]
2	0, 1 , 4, 3, 3, 4, 3, 5 , 7, 6, 6	[0, 0], [2, 6], [8, 10]
3	0, 1 , 4, 3, 3, 4, 3, 5 , 6, 6, 7	[0, 0], [2, 6], [8, 9]
4	0, 1 , 4, 3, 3, 4, 3, 5 , 6 , 6, 7	[0, 0], [2, 6], [9, 9]
5	0, 1 , 4, 3, 3, 4, 3, 5 , 6 , 6 , 7	[0, 0], [2, 6]
6*	0, 1 , 3, 3, 3, 4, 4 , 5 , 6 , 6 , 7	[0, 0], [2, 5]
7*	0, 1 , 3 , 3, 3, 4, 4 , 5 , 6 , 6 , 7	[0, 0], [3, 5]
8*	0, 1 , 3 , 3 , 3, 4, 4 , 5 , 6 , 6 , 7	[0, 0], [4, 5]
9*	0, 1 , 3 , 3 , 3 , 4, 4 , 5 , 6 , 6 , 7	[0, 0], [5, 5]
10*	0, 1 , 3 , 3 , 3 , 4, 4 , 5 , 6 , 6 , 7	[0, 0]
11*	0, 1 , 3 , 3 , 3 , 4, 4 , 5 , 6 , 6 , 7	

Синим выделены элементы исходного массива, **красным** – элементы на своих местах, **зеленым (жёлтым)** – проверяемая пара элементов, для которой выполняется (не выполняется) отношение порядка: $a_i \leq a_{i+1}$. **Жирным** выделена сдвигаемая граница.

Внутренняя итерация	Вывод элемента на свое место
1.1	1 , 3, 5, 3, 3, 4, 6, 6, 7, 0, 4 (i=0, j=10)
1.2	1 , 3, 5, 3, 3, 4, 6, 6, 7, 0 , 4 (i=0, j=9) → 0 , 3, 5, 3, 3, 4, 6, 6, 7, 1 , 4
1.3	0, 3 , 5, 3, 3, 4, 6, 6, 7, 1 , 4 (i=1, j=9) → 0, 1 , 5, 3, 3, 4, 6, 6, 7, 3 , 4
1.4	0, 1 , 5, 3, 3, 4, 6, 6, 7 , 3, 4 (i=1, j=8)
1.5	0, 1 , 5, 3, 3, 4, 6, 6 , 7, 3, 4 (i=1, j=7)
1.6	0, 1 , 5, 3, 3, 4, 6 , 6, 7, 3, 4 (i=1, j=6)
1.7	0, 1 , 5, 3, 3, 4 , 6, 6, 7, 3, 4 (i=1, j=5)
1.8	0, 1 , 5, 3, 3 , 4, 6, 6, 7, 3, 4 (i=1, j=4)
1.9	0, 1 , 5, 3 , 3, 4, 6, 6, 7, 3, 4 (i=1, j=3)
1.10	0, 1 , 5 , 3, 3, 4, 6, 6, 7, 3, 4 (i=1, j=2)



Модификации

- ❑ Сортировка **по убыванию** (проверяем отношение $a_i \geq a_{i+1}$).
- ❑ Изменение **направления просмотра** массива.
- ❑ Границы одноэлементных массивов, $(l = r)$, **в стек не помещаются** потому, что такие массивы уже упорядочены.
- ❑ Границы двухэлементных массивов, $(r - l = 1)$, в стек не помещаются. Такие массивы **обрабатывается «на лету»**: если $a_l > a_r$, тогда меняем эти элементы местами; иначе элементы уже на своем месте.



Задание

Отсортировать массив модифицированным алгоритмом быстрой сортировки без помещения в стек одно- и двух-элементных массивов.

Внешняя итерация	Состояние массива	Стек
Вход 1	7, 4, 6, 2, 1, 0, 3, 4, 9, 5, 1	[0, 10]



Вывод оценок трудоемкости

Давайте оценим трудоемкость быстрой сортировки. **Кто желает помочь?**



Оценки трудоемкости

Лучший случай (массив каждый раз делится пополам)

$$T(n) = n \cdot \log_2(n).$$

Худший случай (массив упорядочен)

$$T(n) = n^2.$$



Достоинства и недостатки

Особенности: 1) большинство обменов будет **полуобменами** (простые пересылки), поскольку один из **ключей** (который вначале итерации совпадает с левой границей массива) все время будет в регистре, и его не нужно записывать до самого конца итерации; 2) в стеке одновременно будет находиться не более $\log_2(n)$ пар ссылок (Кнут Д., 142 с.).

Достоинства – неплохая оценка трудоемкости для наилучшего случая, применимость для любых чисел.

Недостатки:

- ❑ квадратичная трудоемкость в наихудшем случае;
- ❑ неэффективность на малых объемах данных в связи с необходимостью использования стека и выполнения большого объема вспомогательных вычислений.



Задание на лабораторную работу

Тема лабораторной работы № 4: Программная реализация алгоритмов сортировки обменом

Задание (обязательно): программная реализация и сравнительный анализ эффективности алгоритмов (вычислять коэффициент прироста трудоемкости)

Типы программ	Пузыре к	Шейке р	Шейкер с остановкой по готовности
Стандартный Swap			
« Swap » пользователя с временным хранилищем			
« Swap » пользователя без временного хранилища			
Прямое встраивание с временным хранилищем			
Прямое встраивание без временного хранилища			

Задание (опционально): программная реализация быстрого сортировки.

