Сортировка за линейное время

Дискретный анализ 2012/13

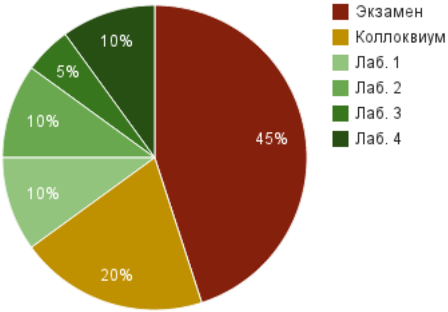
Андрей Калинин, Татьяна Романова

3 сентября 2012 г.

Прежде чем начать первую лекцию

- Романова Татьяна Сергеевна, `tr@umc8.ru`.
- Материалы: `http://k806.ru/daprogram`.
- Два семестра, два экзамена.
- В первом семестре 4 лабораторные работы: линейная сортировка, словарь, исследование качества программы, поиск подстрок.

Об оценках



- $\geqslant 90$ — «5»
- $\geqslant 70$ — «4»
- $\geqslant 60$ — «3»

Требования к выполнению лабораторных

- Обязательные (невыполнение снижает оценку до 0):
  - самостоятельная работа;
  - сдача в указанный срок;
  - оформление отчета.
- Желательные (невыполение снижает оценку):
  - корректная работа;
  - оптимальная реализация;
  - понятный код;
  - глубокое понимание темы.

Литература по курсу

- Д. Кнут, «Искусство программирования.»
- Т. Кормен, «Алгоритмы: построение и анализ.»
- Д. Гасфилд, «Строки, деревья и последовательности в алгоритмах.»
- Witten, «Managing gigabytes: compressing and indexing documents and images.»
- Г. Уоррен, «Алгоритмические трюки для программистов.»
- `http://k806.ru/books`

Литература

- Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.. Алгоритмы: построение и анализ, 2-е издание, М.:Вильямс, 2005, стр. 220-239, глава 8, «Сортировка за линейное время».

Раздел

Асимптотичске обозначения

- $f(n) = O(g(n))$, если существуют константы c и n_0 такие, что $0 \leqslant f(n) \leqslant cg(n)$ для всех $n \geqslant n_0$.

- $f(n) = \Omega(g(n))$, если существуют константы c и n_0 такие, что $0 \leqslant cg(n) \leqslant f(n)$ для всех $n \geqslant n_0$.

- $f(n) = \Theta(g(n))$, если существуют константы c_1, c_2 и n_0 такие, что $0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ для всех $n \geqslant n_0$.

Линейность

- $f(n) = O(n)$
- n — размерность задачи.
- Линейность (функция $f(n)$):
  - Количество элементарных операций.
  - Объём оперативной памяти.
  - Могут оцениваться другие параметры: количество процессорных ядер или серверов.

Скользкие места

- Не может быть алгоритма, работающего за $O(n)$ и не укладывающегося в $O(n)$ по памяти.
- В строгом смысле элементарные операции — выполняемые команды машиной Тьюринга (или эквивалентной моделью.)
- В практическом смысле элементарность операций определяется реальным процессором.

На практике

- Всегда есть физические ограничения: размер оперативной памяти, жёстких дисков.
- Иерархия оперативной памяти: регистры, кэш-память L1, L2, DRAM, жёсткие диски, распределённые файловые системы.
- Константа c может быть слишком большой.

Раздел

Петабайтная сортировка (Google, 2008)

- Петабайт данных (1024 терабайта, 10^{13} 100-байтных записей).
- 48 000 жёстких дисков на 4 000 серверах.
- Трёхкратное дублирование записей.
- Работа в течение 6 часов.
- На каждом запуске сортировки как минимум один диск выходит из строя.
- Применяемые решения: распределённая файловая система (Google FS) и алгоритмы распределения задач (MapReduce).

Раздел

Сортировка сравнением

- Последовательность $\langle a_1, a_2, \ldots, a_n \rangle$.
- При сортировке используются только попарные сравнения: $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$, $a_i > a_j$.
- Предполагаем, что все элементы различны, тем самым можно считать что используется только $a_i \leq a_j$.

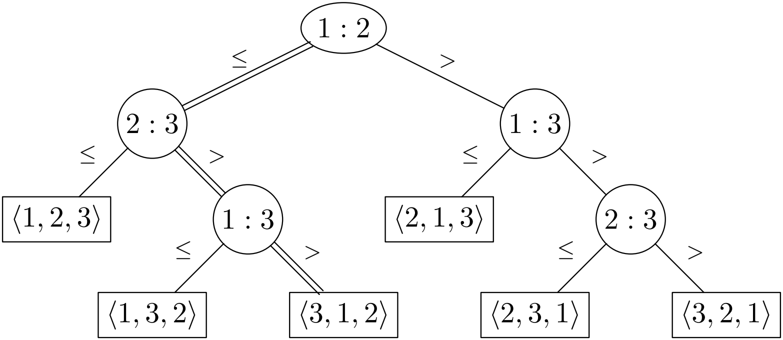
Модель дерева решений

Дерево решений:

- Полное бинарное дерево, в котором представлены операции сравнения элементов.
- Внутренние узлы помечены меткой $i : j$, $1 \leq i, j \leq n$, указывающие на сравнение a_i и a_j.
- Лист помечен перестановкой $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$, дающее окончательное упорядочение элементов $\langle a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)} \rangle$.
- Корректность сортировки сравнением: соответствующее дерево решений должно содержать все $n!$ перестановок исходных n элементов, к которым можно проложить путь реального выполнения сортировки («достижимые» листья.)

Сортировка вставкой трёх элементов

$\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$



Нижняя оценка для наихудшего случая

Теорема

В наихудшем случае в ходе выполнения любого алгоритма сортировки сравнением выполняется $\Omega(n \log_2 n)$ *сравнений.*

Доказательство.

Для дерева решений сортировки n элементов и высотой h с l достижимыми листьями:

$$n! \leq l \leq 2^h \Rightarrow h \geq \log_2(n!) = \Omega(n \log_2 n)$$

$\square$

Раздел

Общая идея

- $\langle a_1, \ldots, a_n \rangle$, a_i — целое и $0 \leq a_i \leq k$.
- Если $k = O(n)$ то время работы равно $\Theta(n)$
- Для каждого a_i определяется $c_i = |\{a_k | a_k < a_i\}|$.
- c_i определяет местоположение a_i в отсортированной последовательности.

Алгоритм

Counting-Sort(A)

for $i \leftarrow 0$ **to** k
    $C[i] \leftarrow 0$
for $j \leftarrow 1$ **to** $length[A]$
    $C[A[j]] \leftarrow C[A[j]] + 1$
// В $C[i]$ хранится количество элементов, равных i.
for $i \leftarrow 1$ **to** k
    $C[i] \leftarrow C[i] + C[i-1]$
// В $C[i]$ — количество элементов, не превышающих i.
for $j \leftarrow length[A]$ **downto** 1
    $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Пример

A

2	5	3	0	2	3	0	3

$, 0 \le a_i \le 5$

C

2	0	2	3	0	1

$\rightarrow$

2	2	4	7	7	8

B

						3	
	0					3	
	0				3	3	
	0		2		3	3	
0	0		2		3	3	
0	0		2	3	3	3	
0	0		2	3	3	3	5
0	0	2	2	3	3	3	5

C

2	2	4	**6**	7	8
1	2	4	6	7	8
1	2	4	**5**	7	8
1	2	**3**	5	7	8
0	2	3	4	7	8
0	2	3	**3**	7	8
0	2	3	3	7	**7**
0	2	**2**	3	7	7

Свойства сортировки подсчётом

- Не является сортировкой сравнением: ни одна пара элементов не сравнивается друг с другом.
- Линейная (вернее, $\Theta(k+n)$, но при $k = O(n)$ время выполнения $\Theta(n)$).
- Устойчивая (стабильная.)
- Требует дополнительную память под массивы C и B размером n.

Раздел

Общая идея

- d-значные числа последовательно сортируются по разрядам: от младшего к старшему.
- При использовании устойчивой сортировки — корректное упорядочивание.
- Если внутренняя сортировка линейная, то и поразрядная сортировка тоже линейная.

Пример

329
457
657
839
436
720
355

Пример

329
457
657
839
436
720
355

$\Rightarrow$

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

Пример

$$
\begin{array}{|c|}
\hline
329 \\
457 \\
657 \\
839 \\
436 \\
720 \\
355 \\
\hline
\end{array}
\Rightarrow
\begin{array}{|ccc|}
\hline
7 & 2 & 0 \\
3 & 5 & 5 \\
4 & 3 & 6 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
3 & 2 & 9 \\
8 & 3 & 9 \\
\hline
\end{array}
\Rightarrow
\begin{array}{|ccc|}
\hline
7 & 2 & 0 \\
3 & 2 & 9 \\
4 & 3 & 6 \\
8 & 3 & 9 \\
3 & 5 & 5 \\
4 & 5 & 7 \\
6 & 5 & 7 \\
\hline
\end{array}
$$

Пример

329		7	2	0		7	2	0		3	2	9
457		3	5	5		3	2	9		3	5	5
657		4	3	6		4	3	6		4	3	6
839	$\Rightarrow$	4	5	7	$\Rightarrow$	8	3	9	$\Rightarrow$	4	5	7
436		6	5	7		3	5	5		6	5	7
720		3	2	9		4	5	7		7	2	0
355		8	3	9		6	5	7		8	3	9

Алгоритм

$\textsc{Radix-Sort}(A, d)$

1 **for** $i \leftarrow 1$ **to** d
2 Устойчивая сортировка массива A по i-ой цифре

Теорема

Для n d-значных чисел, в которых каждая цифра принимает одно из k значений, алгоритм $\textsc{Radix-Sort}$ *позволяет выполнить корректную сортировку за время $\Theta(d(n+k))$, если внутренняя устойчивая сортировка имеет время работы $\Theta(n+k)$.*

Какие цифры выбрать?

Теорема

Для n b-битовых чисел и натурального числа $r \leq b$ (цифры из r битов) алгоритм Radix-Sort *выполнит сортировку за время* $\Theta\left(\frac{b}{r}(n+2^r)\right)$.

Какие цифры выбрать?

Теорема

Для n b-битовых чисел и натурального числа $r \le b$ (цифры из r битов) алгоритм Radix-Sort *выполнит сортировку за время* $\Theta\left(\frac{b}{r}(n + 2^r)\right)$.

Тем самым:

- Для $b < \lfloor \log_2(n) \rfloor$ асимптотически оптимален выбор $r = b$.
- А для $b \ge \lfloor \log_2(n) \rfloor$: $r = \lfloor \log_2(n) \rfloor$

Свойства поразрядной сортировки

- Линейная, устойчивая, требуется дополнительная память (из-за сортировки подсчётом.)
- Может понадобиться много проходов.
- Несмотря на асимптотическую линейность, для конкретных значений n и r сортировки сравнением могут быть предпочтительнее из-за разных значений постоянных множителей.

Раздел

Общая идея

- На вход поступают n вещественных чисел, порождённых случайным процессом и равномерно распределённых в интервале $[0, 1)$.
- $[0, 1)$ разбивается на n одинаковых интервалов («карманов», buckets)
- Числа помещаются в список, соответствующий каждому карману. Т.к. распределене равномерное, в одном кармане появляется не очень много чисел.
- Все списки сортируются (вставкой.)
- Результат получается объединением (один проход по всем спискам.)

Алгоритм

$\textsc{Bucket-Sort}(A)$

```
n ← length[A]
for i ← 1 to n
        Вставить элемент A[i] в список B[⌊nA[i]⌋]
for i ← 0 to n − 1
        Сортировка вставкой списка B[i]
Объединение списков B[0], B[1], ..., B[n − 1]
```

Пример

	A
1	.78
2	.17
3	.39
4	.26
5	.72
6	.94
7	.21
8	.12
9	.23
10	.68

	B
0	$\emptyset$
1	$\langle .12, .17\rangle$
2	$\langle .21, .23, .26\rangle$
3	$\langle .39\rangle$
4	$\emptyset$
5	$\emptyset$
6	$\langle .68\rangle$
7	$\langle .72, .78\rangle$
8	$\emptyset$
9	$\langle .94\rangle$

Линейность

n_i — количество элементов в кармане $B[i]$. Тогда время работы алгоритма:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \Rightarrow E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$

Для всех $i = 0, 1, \ldots, n-1$:

$$E[n_i^2] = 2 - \frac{1}{n} \Rightarrow T(n) = \Theta(n) + n \cdot O\left(2 - \frac{1}{n}\right) = \Theta(n)$$

Количество элементов в одном кармане

$$X_{ij} = I\{A[j] \in B[i]\} \Rightarrow n_i = \sum_{j=1}^{n} X_{ij},$$

$$E[n_i^2] = \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ k \ne j}} E[X_{ij} X_{ik}],$$

$$E[X_{ij}^2] = 1 \cdot \frac{1}{n} + 0 \cdot \left(1 - \frac{1}{n}\right) = \frac{1}{n},$$

$$E[X_{ij} X_{ik}] = E[X_{ij}] E[X_{ik}] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2},$$

$$E[n_i^2] = n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n}.$$