

Лекция по алгоритмам #4

Тема: сортировки

23 сентября

Собрано 23 сентября 2014 г. в 18:29

Содержание

1 Квадратичные сортировки	1
2 Integer Sorting	1
2.1 Bucket Sort	1
2.2 Radix Sort	1
2.3 Сортировка чисел за $O(n \log_2 \log_2 m)$	2
2.4 Исторический экскурс	2
3 Сортировки за $O(n \log n)$	2
3.1 Исторический экскурс	3
3.2 STL	3
3.3 Адаптивные сортировки	3
3.4 Точные оценки на число сравнений	3
3.5 InplaceMerge	3

1 Квадратичные сортировки

Название	Количество сравнений	Количество копирований	Стабильная	Память
Selection	$\Theta(n^2)$	$\mathcal{O}(n)$	-	$\mathcal{O}(1)$
Insertion	$\Theta(n + inv)$	$\Theta(inv)$	+	$\mathcal{O}(1)$
InsertionBS	$\Theta(n \log n)$	$\Theta(inv)$	+	$\mathcal{O}(1)$
Bubble	$\Theta(n^2)$	$\mathcal{O}(n^2)$	+	$\mathcal{O}(1)$
ShakerBubble	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	+	$\mathcal{O}(1)$
ShellSort	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(n \log^2 n)$	-	$\mathcal{O}(1)$

2 Integer Sorting

2.1 Bucket Sort

Отсортируем n целых чисел.

1. Найдем минимум l и максимум r .

2. Разобьем числовую прямую на n равных по длине полуинтервалов (buckets).
3. Положили каждое число x_i в нужный bucket: $\lfloor \frac{r-x_i}{r-l+1} \rfloor$.
4. Каждый bucket отсортируем. Если чисел мало, то InsertionSort, иначе BucketSort. Сортировка устойчива. В среднем работает за $\Theta(n)$. В худшем за $\Theta(n \log_n m)$.

2.2 Radix Sort

Сортируем n целых чисел от 1 до m .
 Сортировка подсчетом за $\Theta(n + m)$.
 Сортировка чисел от 1 до n^2 за $\Theta(n)$.
 Radix sort: сортировка чисел от 1 до m за $O(n \lceil \log_n m \rceil)$.
 Сортировка чисел, как битовых строк, за $O(n \log_2 m)$.
 Сортировка n строк длины L над алфавитом Σ за $\Theta(\Sigma + nL)$
 Сортировка длинных чисел: $O(n \lceil \log_n m \rceil^2)$.

2.3 Сортировка чисел за $O(n \log_2 \log_2 m)$

Kirkpatrick, Reisch'84 [1]

1. Пусть все числа не более 2^{2^k} , то есть имеют длину 2^k бит.
2. $2^{2^k} \leq n \Rightarrow$ отсортируем подсчетом за $\mathcal{O}(n + 2^{2^k}) = \mathcal{O}(n)$.
3. За $\mathcal{O}(n)$ перейдем к рекурсивному вызову сортировки от $(k - 1)$:

```
Sort(k, x) {
  if ( $2^{2^k} \leq n$ ) { CountSort(x); return; }
   $x_i$  делим на две половины  $a_i, b_i$  по  $2^{k-1}$  бит каждая.
  для каждого  $a$ :
    насчитываем список всех парных  $b$ : ListB[a].
    отщепляем от списка максимальный элемент: max[a], длина списка уменьшилась на один
  Sort(k-1, ListB[a])
  создали список всех  $a$ : ListA
  Sort(k-1, ListA)
  перебираем  $a$  в отсортированном порядке
  перебираем парные  $b$  в отсортированном порядке
  result.push_back(pair(a,b))
  result.push_back(pair(a,max[a]))
}
```

Конец сортировки. Время на сортировку чисел от 1 до m : $\Theta(n(\log \log m - \log \log n))$.

2.4 Исторический экскурс

Название	Время	Автор	Год
RadixSort	$\Theta(n \log_n m)$	1887	Hollerith
V.E.B. trees	$\Theta(n \log \log m)$	1975	Van Emde Boas
-	$\Theta(n \log \log m)$	1984	Kirkpatrick
Fuzzy Trees	$\Theta(n \frac{\log n}{\log \log n})$	1990	Fredman, Willard
Fuzzy Trees	$\Theta(n \sqrt{\log n})$	1993	Fredman, Willard
Exponential Trees	$\Theta(n \log \log n)$	1995	Andersson
-	randomized $\Theta(n)$	1998	группа авторов
-	$\Theta(n \sqrt{\log \log n})$	1995	Han, Thorup

3 Сортировки за $O(n \log n)$

Из любой сортировки можно сделать стабильную.

MergeSort можно писать без рекурсии.

IntroSort = QuickSort + HeapSort + InsertionSort.

3.1 Исторический экскурс

Название	Количество сравнений	Количество копирований	Stable	Rand	Память	Автор	Год
QuickSort	$\Theta(n \log n)$	$\Theta(n \log n)$	-	+	$\Theta(\log n)$	1962	Hoare
IntroSort	$\Theta(n \log n)$	$\Theta(n \log n)$	-	-	$\Theta(\log n)$	1997	Musser
MergeSort	$\Theta(n \log n)$	$\Theta(n \log n)$	+	-	$\Theta(n)$	1948	Neumann
InplaceMergeSort	$\Theta(n \log^2 n)$	$\Theta(n \log^2 n)$	+	-	$\mathcal{O}(1)$	1977	Pardo
HeapSort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	-	-	$\mathcal{O}(1)$	1964	Williams
AdaptiveHeapSort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	-	-	$\mathcal{O}(n)$	1993	Petersson
InplaceMergeSort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	+	-	$\mathcal{O}(1)$	1995	Symvonis
InplaceMergeSort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	+	-	$\mathcal{O}(1)$	2008	Kim, Kutzner

3.2 STL

В C/C++:

sort = IntroSort = Quick + Heap + Insertion

stable_sort = MergeSort with inplace_merge (пытается выделить память, если не получается, делает merge за $\mathcal{O}(n \log n)$).

3.3 Адаптивные сортировки

AdaptiveHeapSort = построение декартова дерева + Heap

SplaySort = сортировка Splay деревом

AdaptiveHeapSort:

- Уже отсортированный в возрастающем или убывающем порядке сортируется за линию.
- Если массив можно разбить на k возрастающих или убывающих подпоследовательностей, время сортировки $\mathcal{O}(n \log k)$
- Если массив содержит x инверсий, время сортировки $\mathcal{O}(n \log(\frac{x}{n} + 2))$

3.4 Точные оценки на число сравнений

- Lower bound: $\log_2(n!) = n \log_2 n - 1.44n$
- Текущий оптимум: $n \log_2 n - 1.38n$ (чистая теория)
- HeapSort при $n = 2^k$: $Build + Sort = n + (n \log n - 2n) = n \log n - n$
- MergeSort при $n = 2^k$: $n(\log n - 1) - n = n \log n - 2n$

3.5 InplaceMerge

Reverse \rightarrow Rotate \rightarrow Partition \rightarrow InplaceMerge за $\mathcal{O}(n \log n)$

Список литературы

- [1] *Upper bounds for sorting integers on random access machines*
David Kirkpatrick, Stefan Reisch, TCS 28 (1984) 263-276