Продвинутые сортировки

Алексей Владыкин

СПбГУ ИТМО

02 ноября 2009

- Сортировки с менее чем квадратичной трудоёмкостью.
- Идеи:
  - «разделяй и властвуй»;
  - слияние упорядоченных массивов;
  - использование вспомогательных структур данных;
  - использование свойств сортируемых объектов.

Быстрая сортировка (Ч. Хоар, 1962)

- Quick Sort
- Идея: «разделяй и властвуй».
- *Среднее* время работы — $O(n \log_2 n)$.
- Сортировка неустойчива.

```
void quick_sort(int a[], int n) {
    int l = 0, r = n - 1, p = a[n / 2];
    while (l < r) {
        while (a[l] < p) { ++l; }
        while (p < a[r]) { --r; }
        if (l <= r) {
            swap(&a[l], &a[r]);
            ++l; --r;
        }
    }
    if (0 < r) { quick_sort(a, r + 1); }
    if (l < n - 1) { quick_sort(a + l, n - l); }
}
```

Быстрая сортировка

То же на функциональном языке Clojure:

```
(defn quick-sort
    ([] [])
    ([x & xs] (concat
        (apply quick-sort (filter #(< % x) xs))
        [x] (apply quick-sort (filter #(>= % x) xs)))))
```

Сортировка слиянием (Дж. фон Нейман, 1945)

- Merge Sort
- Идеи: «разделяй и властвуй», слияние отсортированных массивов.
- Время работы — $O(n \log_2 n)$.
- Сортировка устойчива.
- Вариации:
  - как выбирать серии для слияния;
  - сколько серий сливать за раз;
  - сливать с дополнительной памятью или без.

Сортировка слиянием

```
void merge_sort_impl(int a[], int n, int t[]) {
    if (n < 2) { return; }
    int m = n / 2, i, j, k;
    if (0 < m) { merge_sort_impl(a, m, t); }
    if (m < n - 1) { merge_sort_impl(a + m, n - m, t); }
    int *l = t, *r = t + m + 1;
    memcpy(l, a, m * sizeof(int)); l[m] = INT_MAX;
    memcpy(r, a + m, (n - m) * sizeof(int)); r[n - m] = INT_MAX;
    i = j = 0;
    for (k = 0; k < n; ++k) {
        if (l[i] <= r[j]) { a[k] = l[i]; ++i; }
        else { a[k] = r[j]; ++j; }
    }
}

void merge_sort(int a[], int n) {
    int *t = malloc((n + 2) * sizeof(int));
    merge_sort_impl(a, n, t);
    free(t);
}
```

Сортировка слиянием

То же на функциональном языке Clojure:

```
(defn merge* [left right] (cond
    (nil? left) right
    (nil? right) left
    :else (let [[l & ll] left [r & rr] right]
            (if (<= l r)
                (cons l (merge* ll right))
                (cons r (merge* left rr))))))

(defn merge-sort [& x] (cond
    (< (count x) 2) x
    :else (let [[left right] (split-at (/ (count x) 2) x)]
        (merge*
            (apply merge-sort left)
            (apply merge-sort right)))))
```

Пирамидальная сортировка

- Heap Sort
- Идея: использование вспомогательной структуры данных.
- Время работы — $O(n \log_2 n)$.
- Сортировка неустойчива.
- Вспомогательная структура данных — куча (heap):
  - добавить элемент;
  - извлечь максимальный элемент.

Пирамидальная сортировка

```
void heap_sift(int a[], int i, int n) {
    while (2 * i + 1 < n) {
        int j = 2 * i + 1;
        if (j + 1 < n && a[j] < a[j + 1]) { ++j; }
        if (a[i] < a[j]) {
            swap(&a[i], &a[j]);
            i = j;
        } else { break; }
    }
}
void heap_sort(int a[], int n) {
    int i = (n - 2) / 2;
    while (0 <= i) {
        heap_sift(a, i, n);  --i;
    }
    i = n - 1;
    while (0 < i) {
        swap(&a[0], &a[i]);
        heap_sift(a, 0, i);
        --i;
    }
}
```

Поразрядная сортировка

- Radix Sort
- Идея: использование конечной длины сортируемых чисел
- Сортируем отдельно по каждому разряду, начиная со старшего
- Время работы — $O(n)$
- Сортировка устойчива

Блочная сортировка

- Bucket Sort
- Идея: «разделяй и властвуй»
- Разделяем массив на m подмассивов, сортируем независимо, объединяем
- *Среднее* время работы — $O(n)$
- Сортировка устойчива

Сортировка подсчетом

- Counting Sort
- Идея: использование небольшого разнообразия сортируемых значений
- Считаем количество каждого входного значения; заполняем массив в соответствии со счетчиками
- Время работы — $O(n)$
- Устойчива ли эта сортировка?