Шейкерная сортировка ShakerSort

Можно заметить, что в BubbleSort «легкие» элементы «всплывают» быстро, а «тяжелые» «тонут» медленно.

Пример. СИДОРОВ

ВО

ВР

ВО

ВД

ВИ

ВСИДОРО

Шейкерная сортировка ShakerSort

ДВА (**!**) **изменения** в алгоритме, которые были предложены для уменьшения трудоемкости:

1) Изменение направления просмотра массива на каждой итерации.

2) Установление границ рабочей части массива на место последнего обмена на каждой итерации.

Шейкерная сортировка ShakerSort

Алгоритм на псевдокоде

L, R – левая и правая границы рабочей части массива,
n – количество элементов в массиве.

```
L := 1, R := n, k := n,
DO
    DO ( j := R, R-1, ... L+1)
        IF (a_j < a_j-1)  a_j↔a_j-1,  k := j  FI
    OD
    L := k
    DO ( j := L, L+1, ... R-1)
        IF (a_j > a_j+1)  a_j↔a_j+1,  k := j  FI
    OD
    R := k
OD (L < R)
```


К У Р А П О В А
А В
А О
А П
А А
А Р
А У
А К
А К У Р А П О В
К У
Р У
А У
П У
О У
В У
А К Р А П О В У
В О
В П
А В
А Р
А К


А А К Р В П О У
К Р
В Р
П Р
О Р
А А К В П О Р У
О П
В О
В К
А А В К О П Р У
К О
О П
А А В К О П Р У

Два усовершенствования в алгоритме позволяют уменьшить только количество сравнений:

$$M_{ср} = \frac{3(n^2 - n)}{4}$$

$$C < \frac{n^2 - n}{2}$$

$$T = O(n^2),\ n \to \infty$$

Метод шейкерной сортировки сильно зависит от исходной упорядоченности массива по количеству сравнений.

Метод обеспечивает устойчивую сортировку.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит

Видео:
BubbleSort или ShakerSort ?

Метод прямого включения InsertSort

Начиная с $i = 2$ берём очередной i–й элемент массива и включаем его на нужное место среди первых $(i-1)$ элементов, при этом все элементы, которые больше a_i сдвигаются на одну позицию вправо.

Метод прямого включения

Алгоритм на псевдокоде

DO (i: = 2, 3, …, n)

$t := a_i$, j := i -1

DO (j > 0 и $t < a_j$)

$a_{j+1} := a_j$

j := j -1

OD

$a_{j+1} := t$

OD


К У Р А П О В А
К У Р А П О В А
К Р У А П О В А
А К Р У П О В А
А К П Р У О В А
А К О П Р У В А
А В К О П Р У А
А А В К О П Р У

Для определения трудоемкости оценим количество операций **для каждого значения *i* :**

$C_{min} = 1 \quad C_{max} = i - 1$

$M_{min} = 2 \quad M_{max} = i + 1$

Тогда **для всех *i*** :

$C_{min} = n - 1 \quad C_{max} = \frac{n^2 - n}{2}$

$M_{min} = 2(n-1) \quad M_{max} = \frac{n^2 - n}{2} + 2n - 2$

Средняя трудоемкость этого метода имеет квадратичный порядок $\mathbf{T = O(n^2)}$, $n \to \infty$.

Метод прямого включения устойчивый.

Метод сильно зависит от исходной упорядоченности массива.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит
InsertSort	$O(n^2)$	Устойчив	Сильно зависит

Видео: InsertSort

Метод Шелла ShellSort

Из оценок метода прямого включения *InsertSort* видно, что, чем лучше упорядочен массив, тем меньше операций потребуется для его сортировки.

Основная идея метода Шелла *ShellSort* состоит в предварительном улучшении порядка элементов массива, а затем окончательной сортировке его методом прямого включения.

Шелл предложил работать в методе прямого включения с шагом *k* большим, чем 1, что предварительно улучшило бы упорядоченность массива.

Чем больше величина шага, тем меньше операций сравнения и пересылки приходиться

Определение Предварительная сортировка массива методом прямого включения с шагом к > 1 называется к - сортировкой.

Метод Шелла заключается в проведении сначала нескольких к-сортировок, а затем окончательной сортировке массива с шагом к=1.

Обозначим

$H = (h_1, h_2, \ldots, h_m)$ – последовательность из *m* возрастающих шагов, где $h_1 = 1, h_1 < h_2 < h_3 < \ldots < h_m$.

Производя последовательно к-сортировки с шагами $h_m, h_{m-1}, .., h_1$, получим упорядоченный массив. Это гарантируется тем, что последний шаг $h_1 = 1$

Метод Шелла (ShellSort)

Алгоритм на псевдокоде

<вычисление массива шагов h>

DO (k := h_m , h_{m-1} , … 1)

DO (i := k+1, k+2, … n)

$t := a_i$, j := i - k

DO (j > 0 и $t < a_j$)

$a_{j+k} := a_j$

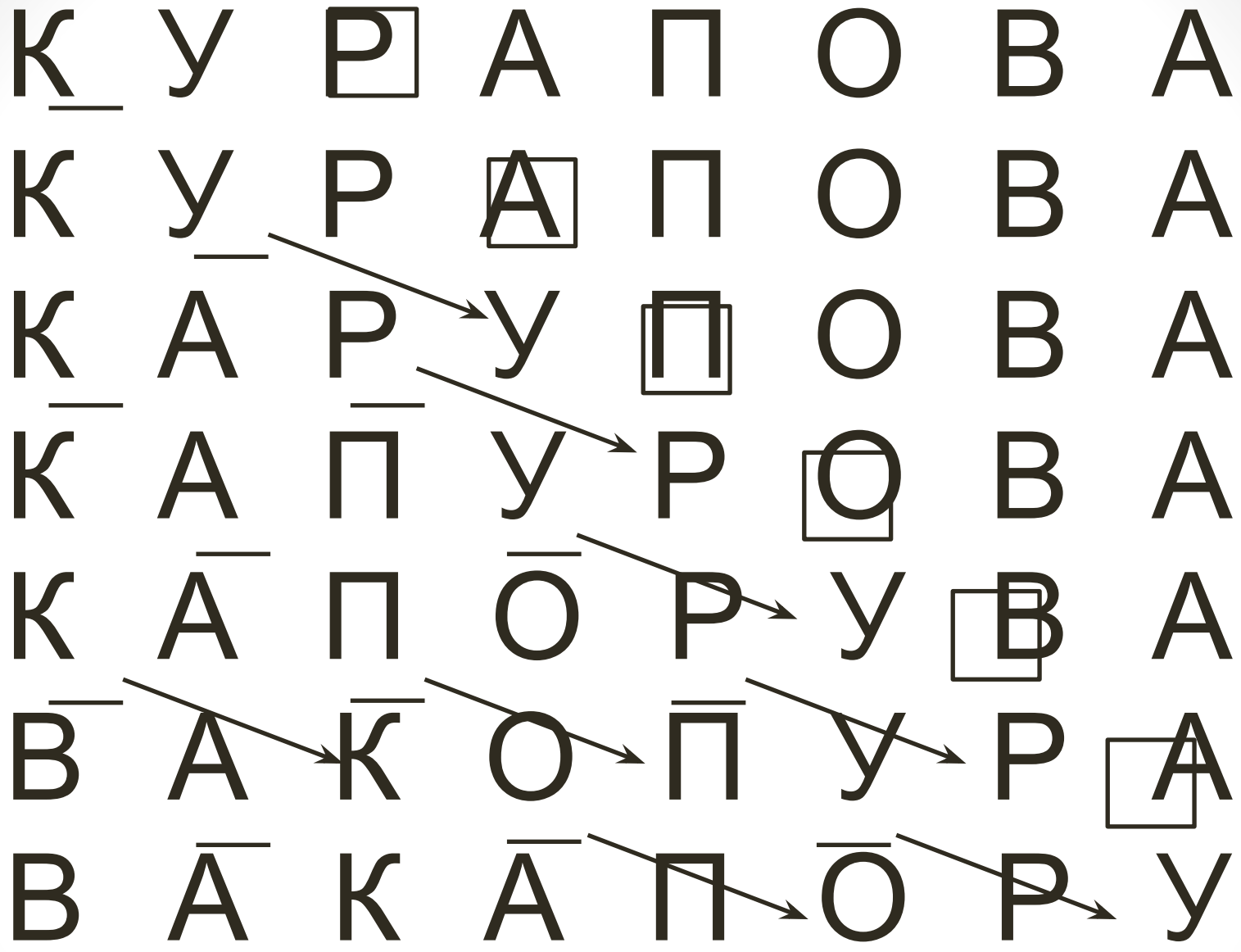
j := j - k

OD

$a_{j+k} := t$

OD

OD


К У Р А П О В А
К У Р А П О В А
К А Р У П О В А
К А П У Р О В А
К А П О Р У В А
В А К О П У Р А
В А К А П О Р У


В А К А П О Р У
А В К А П О Р У
А В К А П О Р У
А А В К П О Р У
А А В К П О Р У
А А В К О П Р У
А А В К О П Р У

Эффективность метода Шелла по времени работы зависит от выбора значений шагов.

Последовательность значений шагов, которая дает наилучшую трудоемкость, пока неизвестна, но существует и часто используется следующая последовательность шагов, предложенная **Д.Кнутом**:

$h_1 = 1, \quad h_i = 2h_{i-1} + 1, \quad i = 2,\ldots m, \quad m = \lfloor \log_2 n \rfloor - 1$

Пример

n = 100, $\mathbf{m} = \lfloor \log_2 100 \rfloor - 1 = \mathbf{5}$,

$\mathbf{h_1 = 1}, \quad \mathbf{h_2} = 2h_1 + 1 = \mathbf{3}$,

$\mathbf{h_3} = 2h_2 + 1 = \mathbf{7}, \quad \mathbf{h_4} = 2h_3 + 1 = \mathbf{15}, \quad \mathbf{h_5} = 2h_4 + 1 = \mathbf{31}$.

При использовании последовательности шагов, предложенной Д.Кнутом, метод имеет порядок трудоёмкости $\mathbf{O}(n^{1.2})$, $n \to \infty$.

Метод Шелла существенно быстрее методов с квадратичной трудоемкостью.

В примере: Insert – 46 операций,
Shell – 34 операции.

Метод Шелла не устойчив.

Метод зависит от исходной упорядоченности массива.

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
SelectSort	$O(n^2)$	Не устойчив	Не зависит
BubbleSort	$O(n^2)$	Устойчив	Зависит
ShakerSort	$O(n^2)$	Устойчив	Зависит
InsertSort	$O(n^2)$	Устойчив	Сильно зависит
ShellSort	**$O(n^{1,2})$**	**Не устойчив**	**Зависит**

Видео: ShellSort