Лекция 10
Сортировка.

Сергей Леонидович Бабичев

Задача сортировки.

Задача сортировки

- На прошлой лекции мы выяснили, что если числа упорядочить, то можно искать нужное число в массиве из N элементов за $\log N$.
- Искать можно не только числа, а строки, структуры, массивы.
- Искать можно не структуру целиком, а только её какое-то поле.
- Мы должны сравнить *ключи* о вынести решение: должен ли один ключ находиться раньше другого в упорядоченной последовательности.

Задача сортировки

Задача. Имеется последовательность из n *ключей*.

$$k_1, k_2, \ldots, k_n$$

Требуется: упорядочить ключи по *не убыванию* или *не возрастанию*.

- Это означает: найти перестановку ключей

  $$p_1, p_2, \ldots, p_n$$

  такую, что

  $$k_{p_1} \leqslant k_{p_2} \leqslant \cdots \leqslant k_{p_n}$$

  или

  $$k_{p_1} \geqslant k_{p_2} \geqslant \cdots \geqslant k_{p_n}$$

Задача сортировки

- Элементами сортируемой последовательности могут иметь любые типы данных. Обязательное условие — наличие *ключа*.

  Последовательность структур:
  `(Москва, 10000000), (Нью-Йорк, 12000000), (Париж, 9000000), (Токио, 20000000), (Лондон, 10000000), (Дели, 9000000)`

  Ключ — число жителей

Устойчивость сортировки

- Алгоритм сортировки *устойчивый*, если он сохраняет относительный порядок элементов.
- Начальная последовательность, сортируем по невозрастанию ключей. Две записи имеют одинаковые ключи. Красная *перед* зелёной.
  `(Москва, 10000000)`, `(Нью-Йорк, 12000000)`, `(Париж, 9000000)`, `(Токио, 20000000)`, `(Лондон, 10000000)`, `(Дели, 9000000)`
- Устойчивая сортировка: красная запись осталась перед зелёной.
  `(Токио, 20000000)`, `(Нью-Йорк, 12000000)`, `(Москва, 10000000)`, `(Лондон, 10000000)`, `(Париж, 9000000)`, `(Дели, 9000000)`
- Неустойчивая сортировка: красная запись оказалась *после* зелёной.
  `(Токио, 20000000)`, `(Нью-Йорк, 12000000)`, `(Лондон, 10000000)`, `(Москва, 10000000)`, `(Париж, 9000000)`, `(Дели, 9000000)`

Сортировки сравнением.

Сортировка сравнением

- Самый распространённый вид сортировки: *сортировка сравнением*.
- Требования к алгоритму: **для ключей должна существовать по крайней мере одна операция сравнения**

$$a < b$$

- Полагается, что

$$\overline{(a < b)} \wedge \overline{(b < a)} \rightarrow a = b$$

- Это необходимое условие для соблюдения *закона трихотомии*: для любых a, b либо $a < b$, либо $a = b$, либо $a > b$.

Готовая функция сортировки в языке Си

- Требуется функция сравнения элементов.
- `int cmp(void const *el1, void const *el2);`
- Должна возвращать 0, если элементы равны, что-то отрицательное, если первый меньше и что-то положительное, если первый больше.

```
int cmp_int(const void *el1, const void *el2) {
  return *(int *)el1 - *(int *)el2;
}
```

Сортировка в Си

- Функция `qsort` умеет сортировать всё, что угодно.
- Ей нужно подать:
  1. Указатель на сортируемый массив.
  2. Количество сортируемых элементов.
  3. Размер сортируемого элемента.
  4. Указатель на функцию сравнения.

Указатели на функции

- Функция, как и любая переменная, может иметь адрес.
- Если этот адрес запомнить в переменную-указатель, его можно передать в другую функцию и в той вызвать нужную.
- Синтаксис описания указателей на функции сложен.
- Вот несколько примеров:

```
double (*integ)(double x);
int (*func)(int, int, int);
void (*dumm)();
```

Функция сравнения для функции qsort

- Прототип функции сортировки сложен:

```
#include <stdlib.h>

void qsort(void *ptr, size_t count, size_t size,
            int (*comp)(const void *, const void *));
```

- Для нас интересен аргумент comp. Он есть указатель на функцию, которая принимает два указателя const void * и возвращает int.
- Мы должны написать функцию сравнения элементов нашего типа, преобразование её аргументов мы должны сделать сами!

```
int cmp_int(const void *el1, const void *el2) {
    const int *l1 = el1;
    const int *l2 = el2;
    if (*l1 < *l2) return -1;
    if (*l1 > *l2) return 1;
    return 0;
}
```

Сортировка в Си

- Многие пишут так:

```
int cmp_int(const void *el1, const void *el2) {
  return *(int *)el1 - *(int *)el2;
}

int main() {
    int a[100];
    for (int i = 0; i < 100; i++) {
        scanf("%d", a+i);
    }
    qsort(a, 100, sizeof a[0], cmp_int);
    // Here a is sorted in ascending order
    ...
}
```

Алгоритмы сортировки сравнением.

Понятие *инверсии*

Определение: *Инверсия* — пара ключей с нарушенным порядком следования.

$$\{4, 15, 6, 99, 3, 15, 1, 8\}$$

- Имеющиеся следующие инверсии:
  `(4,3), (4,1), (15,6), (15,3), (15,1), (15,8), (6,3), (6,1), (99,3), (99,15), (99,1), (99,8), (3,1), (15,1), (15,8)`
- Перестановка соседних элементов, расположенных в ненадлежащем порядке, уменьшает количество инверсий ровно на 1.
- Количество инверсий в любом множестве конечно, в отсортированном — равно нулю.
- Количество обменов для сортировки конечно и не превосходит числа инверсий.
- Самая простая сортировка: найти инверсию и поменять элементы местами.

Сортировка пузырьком

- Один из простейших в реализации алгоритмов.
- Основная идея: до тех пор, пока соседние элементы не в порядке, меняем их местами.
- Самый большой элемент «всплывает» и оказывается на нужном месте.
- Повторяем до тех пор, пока есть инверсии.

$$\{\underbrace{10, 4}, 14, 25, 77, 2\}$$

$$\{4, \underbrace{10, 14}, 25, 77, 2\}$$

$$\{4, 10, \underbrace{14, 25}, 77, 2\}$$

$$\{4, 10, 14, \underbrace{25, 77}, 2\}$$

$$\{4, 10, 14, 25, \underbrace{77, 2}\}$$

$$\{4, 10, 14, 25, 2, 77\}$$

Сортировка пузырьком

```
void bubblesort(int *a, int n) {
   bool sorted = false;
   while (!sorted) {
      sorted = true;
      for (int i = 0; i < n-1; i++) {
        if (a[i] > a[i+1]) {
           int t = a[i];
           a[i] = a[i+1];
           a[i+1] = t;
           sorted = false;
        }
      }
      n--;
    }
}
```

Сортировка пузырьком: сложность

- Каждый обмен уменьшает количество инверсий на единицу.
- Сложность — $O(N^2)$.
- Устойчива.
- Медленная...

Сортировка вставками

- Первый проход — нужно поместить самый лёгкий элемент на первую позицию.
- В i-м проходе ищется, куда поместить очередной a_i внутри левых i элементов.
- Элемент a_i помещается на место, сдвигая вправо остальные внутри области $0 \ldots i$.

Инвариант: после i-го прохода обеспечена упорядоченность левых i элементов.

$$\{10, 4, 14, 77, 25, 2\}$$
$$\{\underline{2}, 10, 4, 14, 77, 25\}$$
$$\{\underline{2, 4}, 10, 14, 77, 25\}$$
$$\{\underline{2, 4, 10}, 14, 77, 25\}$$
$$\{\underline{2, 4, 10, 14}, 77, 25\}$$
$$\{\underline{2, 4, 10, 14, 25}, 77\}$$
$$\{\underline{2, 4, 10, 14, 25, 77}\}$$

Сортировка вставками

Инвариант сортировки вставками:

$$\underbrace{a_1, a_2, \ldots, a_{i-1}}_{a_1 \leqslant a_2 \leqslant \cdots \leqslant a_{i-1}}, a_i, \ldots, a_n$$

На шаге i имеется упорядоченный подмассив $a_1, a_2, \ldots, a_{i-1}$ и элемент a_i, который надо вставить в подмассив без потери упорядоченности.

Сортировка вставками

```
void insertion(int *a, int n) {
  for (int i = n-1; i > 0; i--) {
    if (a[i-1] > a[i]) {
      int t = a[i-1]; a[i-1] = a[i]; a[i] = t;
    }
  }
  for (int i = 2; i < n; i++) {
    int j = i;
    int tmp = a[i];
    while (tmp < a[j-1]) {
       a[j] = a[j-1]; j--;
    }
    a[j] = tmp;
  }
}
```

Сортировка вставками: особенности

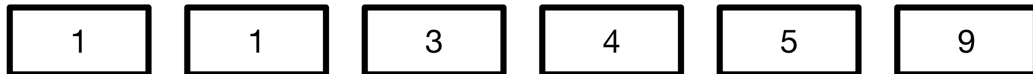
- Сортировка упорядоченного массива требует $O(N)$.
- Сложность в худшем случае $O(N^2)$.
- Алгоритм устойчив.
- Число дополнительных переменных не зависит от размера (*in-place*).
- Позволяет упорядочивать массив при динамическом добавлении новых элементов — *online*-алгоритм.

Быстрые сортировки

- Сортировки пузырьком и вставками слишком медленные для того, чтобы сортировать большие последовательности.
- На помощь приходит

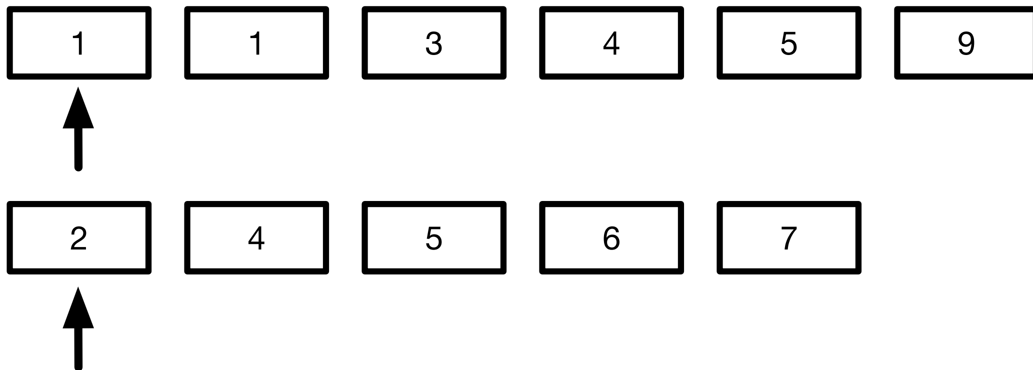
Быстрые сортировки

- Сортировки пузырьком и вставками слишком медленные для того, чтобы сортировать большие последовательности.
- На помощь приходит ... рекурсия.
- Мечта: а что, если бы мы имели два отсортированных массива, за какое время можно получить отсортированный массив, содержащий элементы обоих массивов?



Слияние массивов

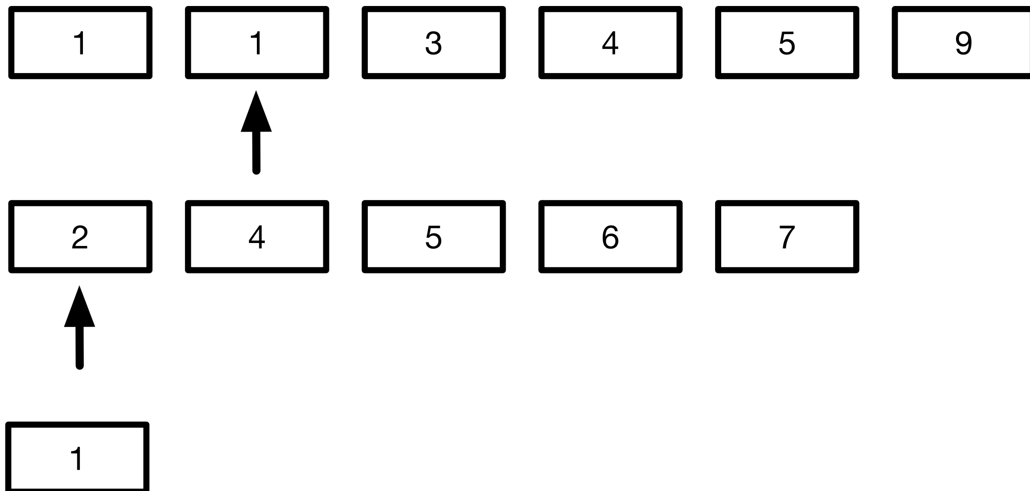
- Эта операция называется *слиянием* и она проводится за длину получившегося массива.
- Заведём два указателя и установим их на начало массивов.



- Сравним значения по этим указателям.

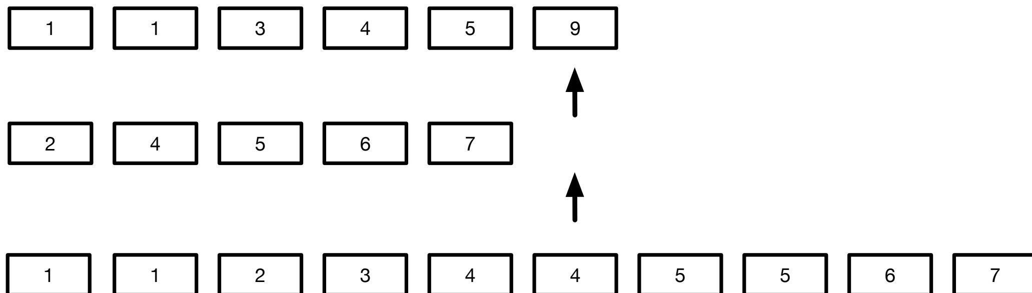
Слияние массивов

- Меньшее значение отправляем в массив-приёмник и передвигаем соответствующий указатель.



Слияние массивов

- Повторяем так до тех пор, пока один из указателей не выйдет за границы одно массива:



- И копируем остаток второго в выходной.

Слияние массивов

- Сложность операции слияния двух массивов длины N — $O(N)$.
- Как это поможет сортировать быстрее?
- Принцип *разделяй и властвуй*.
- Предположим, имелся массив размером 1000 элементов.
- Операция сортировки вставками потребовала бы примерно 1000000 операций.
- Если бы мы разбили массив на два по 500 и отсортировали бы каждый, общее количество операций было бы примерно $2 \cdot 500 \cdot 500 = 500000$.
- Слияние добавило бы ещё 1000 операций. Итого — 501000 операция вместо 1000000.
- Давайте используем рекурсию.

Сортировка слиянием

- Если массивы достаточно малы, выгоднее не отправляться на рекурсию и сортировать, например, вставками.

```
void mergeSort(int *a, int low, int high) {
    if (high - low < THRESHOLD) {
        plainSort(a, low, high);
    } else {
        int mid = (low + high) / 2;
        mergeSort(a, low, mid);
        mergeSort(a, mid+1, high);
        merge(a, low, mid ,high);
    }
}
```

Сортировка слиянием: особенности

- Обычно требует добавочно $\Theta(N)$ памяти (куда же сливать отсортированные массивы?)
- Сложность не зависит от входных данных и равна всегда $\Theta(N \log N)$.
- Прекрасная сортировка, если бы не добавочный расход памяти.
- Попробуем сортировать без добавочной памяти.

Быстрая сортировка

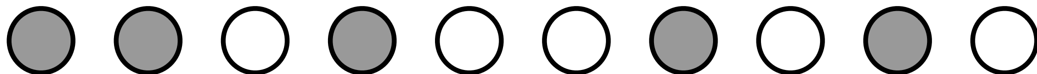
- Давайте выберем для массива какое-то число, близкое к медиане (идеально!).
- После этого все элементы, меньшие этого числа, перенесём в начало массива.
- Все элементы, большие — в конец массива.
- Можно ли это сделать быстро?

Задача о бармене

Задача. На столе в ряд стоят N непрозрачных стаканов, закрытых крышками. В части из них налито молоко, в части — квас.

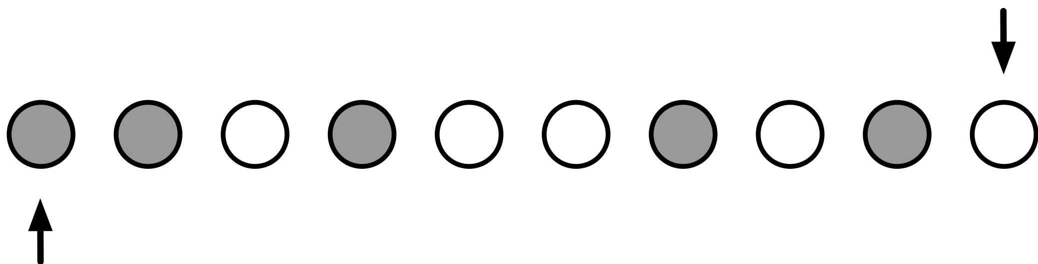
За одну операцию бармен может либо открыть крышку и посмотреть цвет напитка, либо переставить любые два стакана местами.

Память у бармена неважная, он может запомнить два-три факта. Помогите ему расставить стаканы так, чтобы слева стояли все стаканы с молоком, а за ним — все стаканы с квасом.

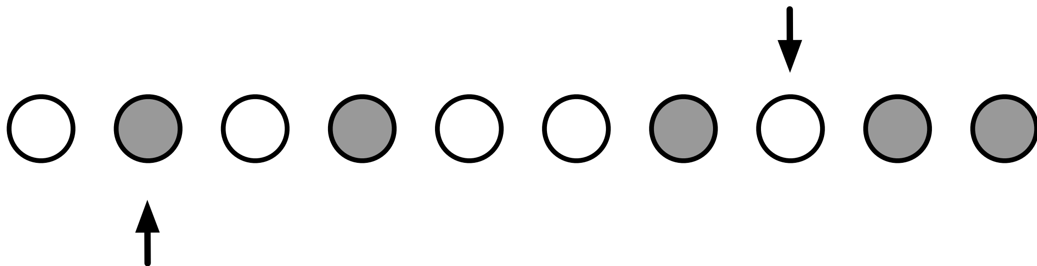


Решение задачи о бармене

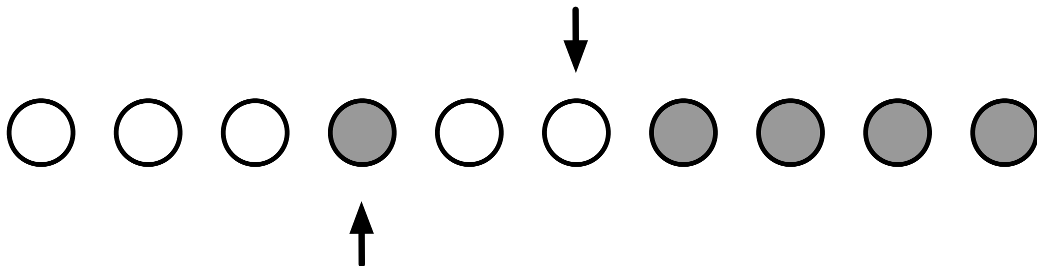
- Хотя память у бармена неглубокая, он способен запомнить положение двух стаканов.
- Сначала он идёт слева и находит первый стакан с квасом и запоминает его позицию.
- Затем он идёт справа и запоминает первый стакан с молоком.



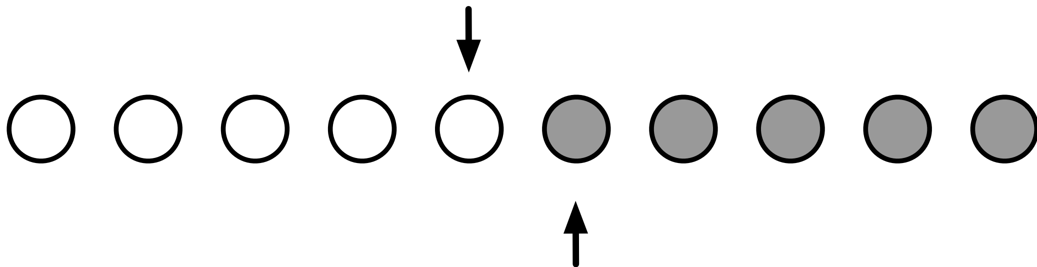
- Он меняет стаканы местами и повторяет операцию ищет стаканы: слева с квасом, справа — с молоком.



- После очередного обмена:



- После очередного обмена указатель на молоко «забежал» за указатель на кофе — работа закончена.



- Даже не понадобилось считать, сколько стаканов какого цвета.

Быстрая сортировка

- Решение задачи о бармене дало нам быстрый алгоритм «расслоения» массива на две области.
- Стаканы с молоком — элементы, меньшие *ведущего* элемента, с квасом — не меньшие.
- Такой обмен даёт неустойчивую сортировку.
- Теперь осталось отсортировать всё внутри белых элементов и всё внутри чёрных элементов.
- Как? Например, рекурсивно.
- При разбиении массива на две примерно равные части получится нечто, похожее на сортировку слиянием, только без слияния :) и сложность будет $O(N \log N)$.

Быстрая сортировка

Массив $S = \{10, 5, 14, 7, 3, 2, 18, 4, 5, 13, 6, 8\}$

- Разделение 1. Пусть ведущий элемент = 8.

$$S_{1l} = \{5, 7, 3, 2, 4, 5, 6, 8\}, S_{1r} = \{10, 14, 18, 13\}$$

$$S_1 = \{\underbrace{5, 7, 3, 2, 4, 5, 6, 8}, \underbrace{10, 14, 18, 13}\}$$

- Рекурсивное разделение 2. Пусть ведущий элемент = 5
  $S_1 = \{5, 7, 3, 2, 4, 5, 6, 8\}$

$$S_{2l} = \{5, 3, 2, 4, 5\}, S_{2r} = \{7, 6, 8\}$$

$$S_2 = \{\underbrace{5, 3, 2, 4, 5}, \underbrace{7, 6, 8}\}$$

Быстрая сортировка

- А как выбрать ведущий элемент?
- Например, как один из элементов массива.
- А если мы будем выбирать его каждый раз плохо, число белых или чёрных будет равно единице?
- Тогда сложность станет $O(N^2)$.
- К счастью, при случайном выборе элемента вероятность такого мала.
- Обычно выбирают ведущий, как медиану трёх элементов — первого, последнего и среднего.

Сортировка с использованием свойств элементов

Сортировка подсчётом

- Есть ли алгоритмы сортировки со сложностью меньшей $O(N \log N)$?

  Да, если использовать свойства ключей.
- Пусть множество значений ключей ограничено $D(K) = \{K_{min}, \ldots, K_{max}\}$.
- Тогда при наличии добавочной памяти в $|D(K)|$ ячеек сортировку можно произвести за $O(N)$.

Сортировка подсчётом

- Сортируем массив $S = \{10, 5, 14, 7, 3, 2, 18, 4, 5, 13, 6, 8\}$
- Заранее известно, что значения массива натуральные числа, которые не превосходят 20.
- Заводим массив $F[1..20]$, содержащий вначале нулевые значения.

$$F = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ \hline\end{array}$$

- Проходим по массиву S. $S_1 = 10; F_{10} \leftarrow F_{10} + 1$.

$$F = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ \hline\end{array}$$

Сортировка подсчётом

- $S_2 = 5; F_5 \leftarrow F_5 + 1.$

$$F = \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$$

- $\ldots$
- $S_{12} = 5; F_8 \leftarrow F_8 + 1.$

$$F = \boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{2}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{0}$$

- Заключительный вывод по ненулевым элементам F:

$$S = \{2, 3, 4, 5, 5, 6, 7, 8, 10, 13, 14, 18\}$$

Сортировка подсчётом: особенности

- Ключи должны быть перечислимы.
- Пространство значений ключей должно быть ограниченным.
- Требуется дополнительная память $O(|D(K)|)$.
- Сложность $O(N) + O(|D(K)|)$