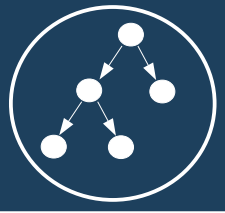


Data Structures and Algorithms

Алгоритмы.
Блочная сортировка

Описание сути алгоритма

Блочная сортировка (карманная, корзинная сортировка) — алгоритм сортировки не использующий сравнение элементов между собой. Предназначена для сортировки данных, ключи которых можно представить в виде целых или вещественных чисел (алгоритм разработан для вещественных чисел [0,1) но можно адаптировать и для целых чисел). В основе лежит рекурсивное разбиение ключей на несколько блоков (поддиапазонов), как только размер блока станет меньше или равен наперед заданному числу (одним из оптимальных размеров является 32), то он сортируется любым оптимальным алгоритмом (например сортировка вставкой). После чего отсортированные блоки объединяются в отсортированную последовательность.

Принцип разбиения на блоки

Основной задачей данного алгоритма является разбиение всех ключей на блоки. Диапазоны этих блоков должны быть не пересекающимися и должны быть расположены в возрастающем порядке. Для этого стоит найти минимальное и максимальное значение ключей, это позволит определить общий диапазон. После этого выбрать количество карманов, и разбить общий диапазон на нужное количество блоков. После это стоит определить функцию которая отнесет текущий элемент к тому или иному блоку.

Для целых чисел

$$F(x)=\frac{n\cdot(x-min)}{max-min+1}$$

Для вещественных чисел

$$F(x)=floor\left(\frac{n\cdot(x-min)}{max-min+1}\right)$$

Сведение о алгоритме

Сложность по времени в наихудшем случае $O\left(n+\frac{n^2}{k}\right)$

Дополнительно используемая память $O(n)$

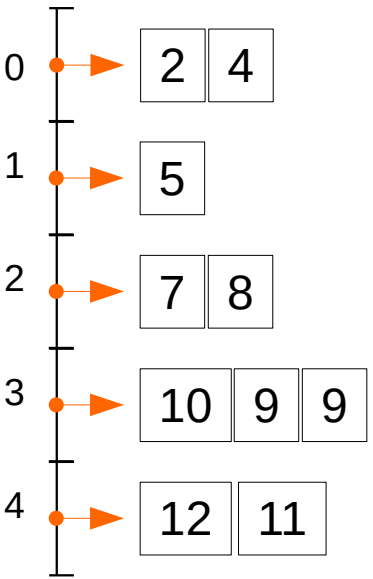
k — количество блоков

Описание алгоритма

1) Определяем значение минимального и максимального значения ключей сортировки. Разбиваем полученный диапазон на нужное количество блоков(в дальнейшем n). Для этого нужно создать массив списков размером n.
2) Заполняем блоки данными из базовой последовательности. Для определения индекса блока используется функция соответствия.
3) Выполняем проход по полученным блокам. Если размер блока равен или меньше 32, провести его сортировку используя сортировку вставкой (или любую иную оптимальную). Если размер больше то рекурсивно перейти к пункту 1.
4) Выполнить сборку отсортированных блоков в отсортированную последовательность.

Графическое пояснение алгоритма

[12, 2, 4, 7, 5, 10, 8, 9, 11, 9] → [min = 2, max = 12] Разобьем на 5 блоков



$$F(12)=\frac{5\cdot(12-2)}{12-2+1}=\frac{50}{11}=4 \qquad F(10)=\frac{5\cdot(10-2)}{12-2+1}=\frac{40}{11}=3$$

$$F(2)=\frac{5\cdot(2-2)}{12-2+1}=\frac{0}{11}=0 \qquad F(8)=\frac{5\cdot(8-2)}{12-2+1}=\frac{30}{11}=2$$

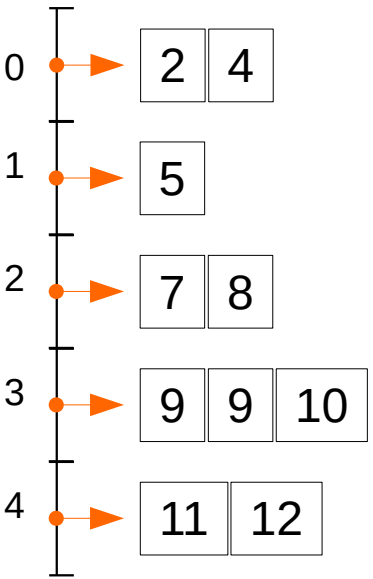
$$F(4)=\frac{5\cdot(4-2)}{12-2+1}=\frac{10}{11}=0 \qquad F(9)=\frac{5\cdot(9-2)}{12-2+1}=\frac{35}{11}=3$$

$$F(7)=\frac{5\cdot(7-2)}{12-2+1}=\frac{25}{11}=2 \qquad F(11)=\frac{5\cdot(11-2)}{12-2+1}=\frac{45}{11}=4$$

$$F(5)=\frac{5\cdot(5-2)}{12-2+1}=\frac{15}{11}=1 \qquad F(9)=\frac{5\cdot(9-2)}{12-2+1}=\frac{35}{11}=3$$

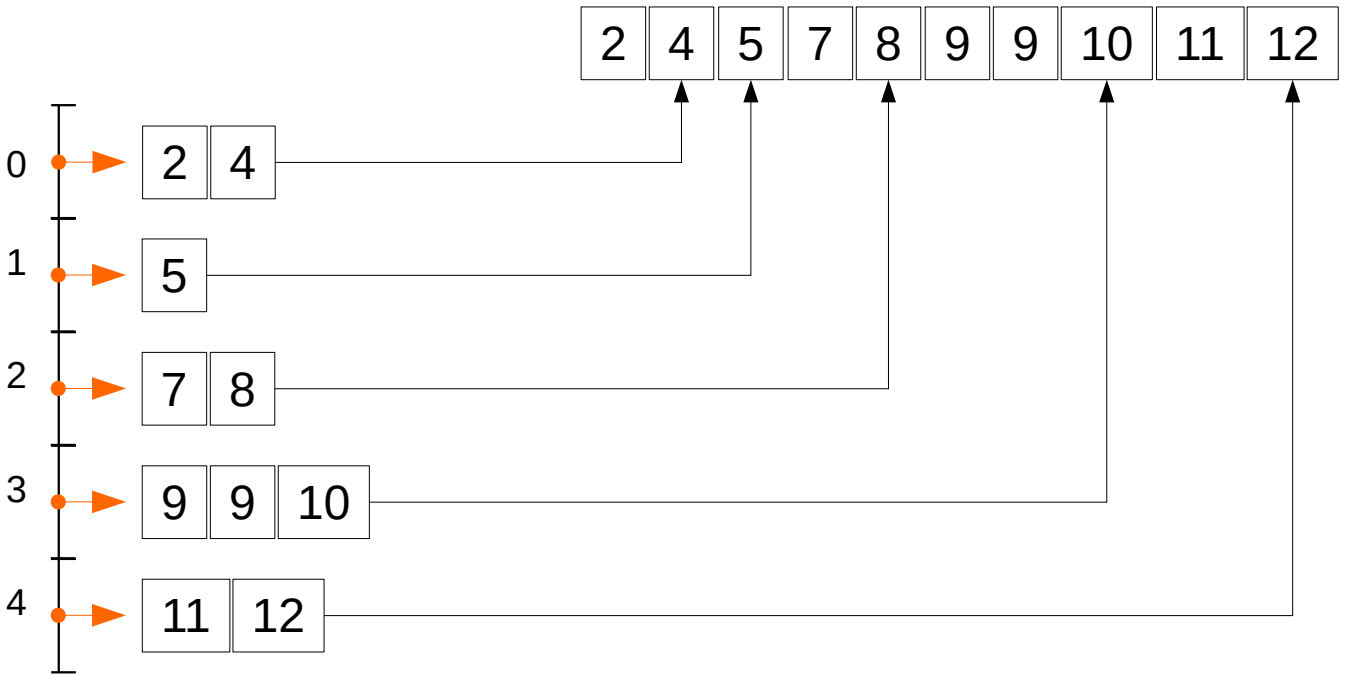
Графическое пояснение алгоритма

Сортируем каждый блок (размер каждого меньше 32)



Графическое пояснение алгоритма

Поочередно выбираем элементы из блоков для создания отсортированной последовательности.



Python

Реализация алгоритма на Python

Функция для сортировки вставкой

```
def insertion_sort(sequence):
    for i in range(1, len(sequence)):
        paste_element = sequence[i]
        while i > 0 and sequence[i-1] > paste_element:
            sequence[i] = sequence[i-1]
            i = i-1
        sequence[i] = paste_element
```

Функция для поиска минимума и максимума

```
def find_min_max(sequence):
    if len(sequence) == 0:
        return [0, 0]
    min_max = [sequence[0], sequence[0]]
    for element in sequence:
        if element < min_max[0]:
            min_max[0] = element
        if element > min_max[1]:
            min_max[1] = element
    return min_max
```

Функция для блочной сортировки

```
def bucket_sort(sequence, n):
    buckets = []
    for i in range(n):
        buckets.append([])
    min_max = find_min_max(sequence)
    if(min_max[0] == min_max[1]):
        return
    for element in sequence:
        buckets[(n * (element - min_max[0])) //
                (min_max[1]-min_max[0]+1)].append(element)
    for bucket in buckets:
        if(len(bucket) <= 32):
            insertion_sort(bucket)
        else:
            bucket_sort(bucket, n)
    insert_index = 0
    for bucket in buckets:
        for element in bucket:
            sequence[insert_index] = element
            insert_index += 1
```

Java

Реализация алгоритма на Java

Метод для сортировки вставкой

```
public static void insertionSort(int[] array) {
    for (int i = 0; i < array.length; i++) {
        int pasteElement = array[i];
        int j;
        for (j = i; j > 0; j--) {
            if (array[j - 1] <= pasteElement) {
                break;
            }
            array[j] = array[j - 1];
        }
        array[j] = pasteElement;
    }
}
```

Метод для поиска минимума и максимума

```
public static int[] findMinMax(int[] array) {
    if (array.length == 0) {
        return new int[] { 0, 0 };
    }
    int min = array[0];
    int max = array[0];
    for (int i = 0; i < array.length; i++) {
        if (array[i] < min) {
            min = array[i];
        }
        if (array[i] > max) {
            max = array[i];
        }
    }
    return new int[] { min, max };
}
```

Метод для вычисления размера каждого блока

```
public static int[] calculateBlockSize(int[] array, int n) {
     int[] blockSize = new int[n];
     int[] minMax = findMinMax(array);
     for (int i = 0; i < array.length; i++) {
     int blockNumber = (int) (1L * n * (array[i] - minMax[0]) / (minMax[1] - minMax[0] + 1));
     blockSize[blockNumber] += 1;
     }
     return blockSize;
}
```

Java

Метод блочной сортировки

```
public static void bucketSort(int[] array, int n) {
      int[] minMax = findMinMax(array);
      if (minMax[0] == minMax[1]) {
            return;
      }
      int[][] buckets = new int[n][];
      int[] addIndex = new int[n];
      int[] blockSize = calculateBlockSize(array, n);
      for (int i = 0; i < buckets.length; i++) {
            buckets[i] = new int[blockSize[i]];
      }
      for (int i = 0; i < array.length; i++) {
            int blockNumber = (int) (1L * n * (array[i] - minMax[0]) / (minMax[1] - minMax[0] + 1));
            buckets[blockNumber][addIndex[blockNumber]++] = array[i];
      }
      for (int[] bucket : buckets) {
            if (bucket.length <= 32) {
                  insertionSort(bucket);
            } else {
                  bucketSort(bucket, n);
            }
      }
      int index = 0;
      for (int[] bucket : buckets) {
            for (int element : bucket) {
                  array[index++] = element;
            }
      }
}
```

Fortran

Реализация алгоритма на Fortran

Функция для поиска минимума и максимума

```
function find_min_max(array)
    integer::find_min_max(2)
    integer, intent(in)::array(:)
    integer:: i
    if (size(array)==0) then
        find_min_max = [0,0]
        return
    end if
    find_min_max(1) = array(1)
    find_min_max(2) = array(1)
    do i = 1, size(array)
        if( array(i) < find_min_max(1)) then
            find_min_max(1) = array(i)
        end if
        if( array(i) > find_min_max(2)) then
            find_min_max(2) = array(i)
        end if
    end do
end function find_min_max
```

Процедура сортировки вставкой

```
subroutine insertion_sort(array)
    integer, intent(inout)::array(:)
    integer:: i, j, insert_element
    do i = 1, size(array)
        j = i
        insert_element = array(j)
        do
            if (j==1 .or. array(j-1) <= insert_element) then
                exit
            end if
            array(j) = array(j - 1)
            j = j - 1
        end do
        array(j) = insert_element
    end do
end subroutine insertion_sort
```

Функция вычисления размера блока

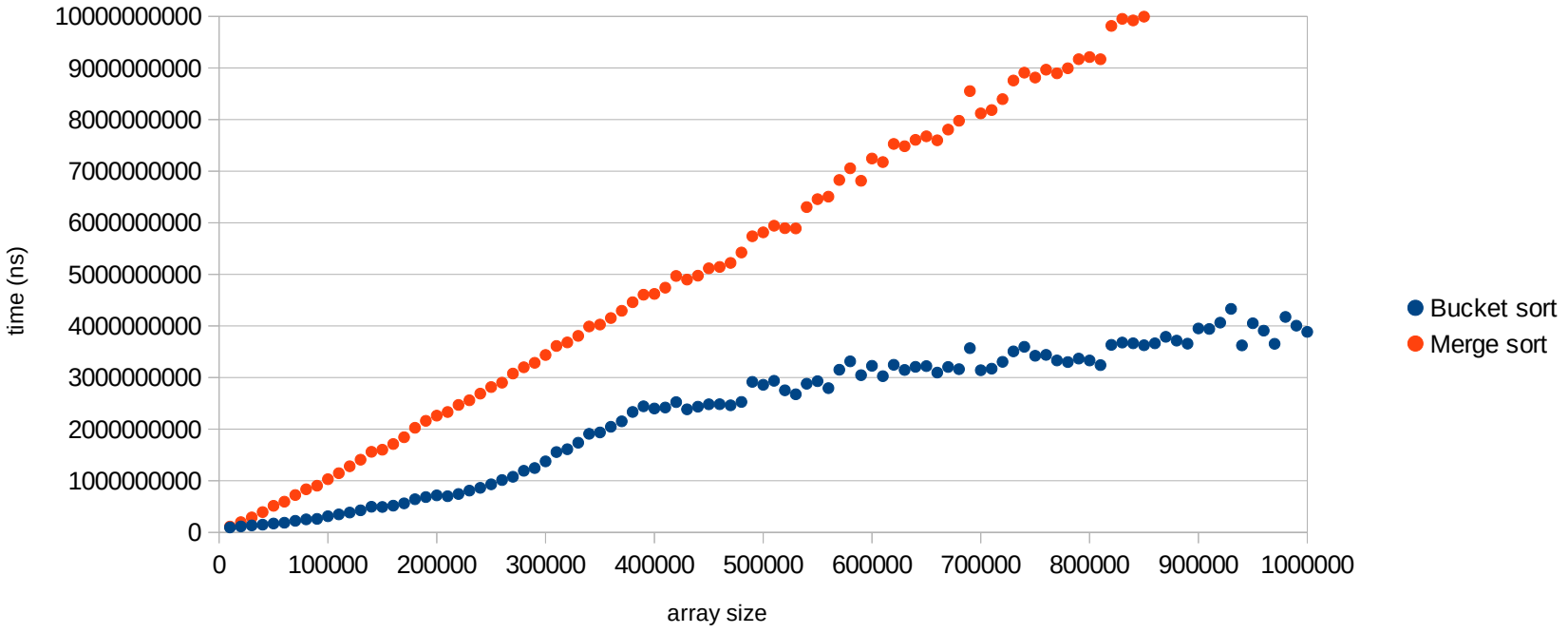
```
function calculate_bucket_size(array, n)
    integer, intent(in)::n
    integer, intent(in)::array(:)
    integer::calculate_bucket_size(n)
    integer::min_max(2)
    integer::i, add_index
    calculate_bucket_size = 0
    min_max = find_min_max(array)
    do i = 1, size(array)
        add_index = (n * (array(i) - min_max(1)))/(min_max(2) - min_max(1) + 1) + 1
        calculate_bucket_size(add_index) = calculate_bucket_size(add_index) + 1
    end do
end function calculate_bucket_size
```

Процедура блочной сортировки

```
recursive subroutine bucket_sort(array, n)
    type Bucket
        integer,allocatable::elements(:)
    end type Bucket
    integer, intent(inout)::array(:)
    integer,intent(in)::n
    type(Bucket)::buckets(n)
    integer::min_max(2)
    integer:: bucket_size(n), add_index(n)
    integer::i, j, insert_index
    min_max = find_min_max(array)
    if(min_max(1)==min_max(2)) then
        return
    end if
    add_index = 1
    bucket_size = calculate_bucket_size(array,n)
    do i = 1, n
        allocate(buckets(i)%elements(bucket_size(i)))
    end do
    do i = 1, size(array)
        j = (n * (array(i) - min_max(1)))/(min_max(2) - min_max(1) + 1) + 1
        buckets(j)%elements(add_index(j)) = array(i)
        add_index(j) = add_index(j) + 1
    end do

    do i = 1, n
        if (size(buckets(i)%elements) <= 32) then
            call insertion_sort(buckets(i)%elements)
        else
            call bucket_sort(buckets(i)%elements, n)
        end if
    end do

    insert_index = 1

    do i = 1, n
        do j = 1, size(buckets(i)%elements)
            array(insert_index) = buckets(i)%elements(j)
            insert_index = insert_index + 1
        end do
        deallocate(buckets(i)%elements)
    end do

end subroutine bucket_sort
```

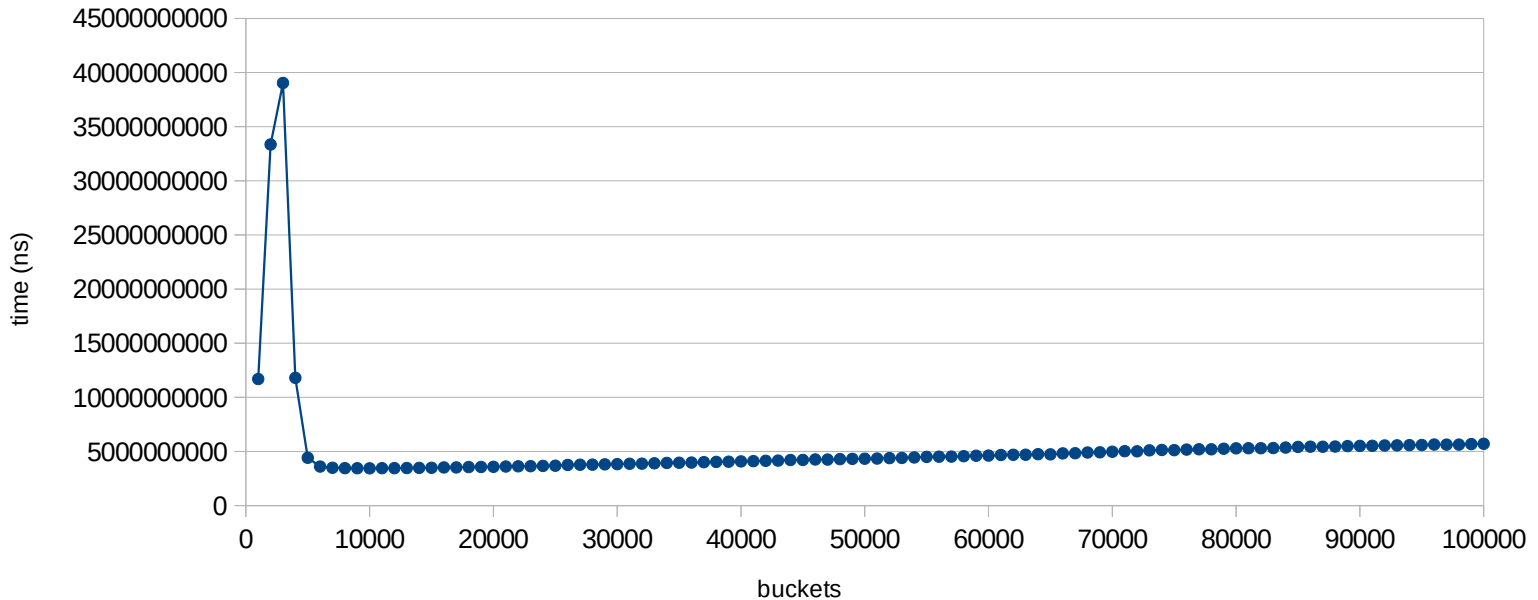
Вычислительный эксперимент

Для проверки оптимальности реализации алгоритма был проведен вычислительный эксперимент по сравнению скорости сортировки массивов разного размера (заполненных случайными числами от 0 до 100 000). Сравнение было сортировки блоками (100 блоков) с алгоритмом сортировки слиянием.



Вычислительный эксперимент

Определенный интерес представляет вопрос об оптимальном количестве блоков. Для массива размером 100 000 было замерено среднее время сортировки для разного количества блоков. Оптимальным оказалось количество блоков 10 000 или 1/10 от общего размера массива.



Список литературы

1) Томас Х. Корман, Чарльз И.Лайзерсон, Рональд Л. Ривест, Клиффорд Штайн - «Алгоритмы построение и анализ, 3-е изд.» : ООО «И.Д. Вильямс», 2013 — 1328 с. ISBN 978-5-8459-1794-2