Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентрованное программирование»

Выполнил
студент группы 3331506/80401 _____ Р. О. Стойка

Руководитель _____ М. С. Ананьевский

«__»_____2021 г

Санкт-Петербург
2021

1. Введение

В работе будет рассмотрен алгоритм сортировки bucketsort. Основная идея - распределяем элементы по вёдрам (блокам, карманам, корзинам), т.е. группируем их по определённому признаку. Элементы в каждом ведре группируем по уточняющим признакам. Этот тип сортировки может обладать линейным временем исполнения. Но сильно зависит от входных данных. Этот алгоритм относится к сортировке распределением.

2. Идея алгоритма

Сортировка проходит в 3 этапа:

Первый этап. Разделим числа в исходном массиве на две большие части: на положительную и отрицательную. Каждую часть положим в отдельный массив. В отрицательный массив мы кладём все числа по модулю.

Второй этап. Теперь сама сортировка. Она будет осуществляться по нескольким признакам. Сортировать мы будем по последней цифре. Тогда получается у нас будет всего 10 корзин (векторов). Возьмём например массив из 13 элементов $\{ 59, 4, 27, 312, 417, 15, 7, 52, 320, 182, 312, 32, 1 \}$. Схематичное отображение результата после первого прохождения изображено на рисунке 1

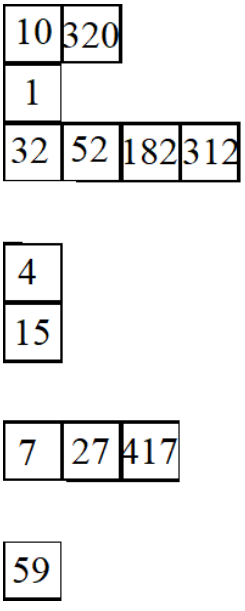


Рисунок 1 — Результат первого прохождения

Возвращаем мы в массив следующим образом. Мы считываем строку, и когда доходим до последнего элемента, который мы вставили в вектор (“ведро”) переходим на следующую.

Теперь мы будем смотреть на вторую цифру, затем на третью и так далее. Если цифры нет, то будем считать, что 0. Для этого мы делим с начало на 1, потом на 10, на 100 и так до 1000000000, это позволит обработать все числа в int, так как максимальное число в его диапазоне того же порядка. У нас в первом векторе можно выделить две части. В первой уже отсортированный массив, а во-второй неотсортированные элементы(не все, а только некоторые).

Третий этап. Мы возвращаем все числа обратно в исходный массив. Зная количество элементов в отрицательном векторе, мы можем начать заполнить массив отдельно отрицательными и положительными числами. Отрицательные числа мы будем вставлять справа налево предварительно каждое умножив на -1. На рисунке 2 изображена схема заполнения исходного массива.

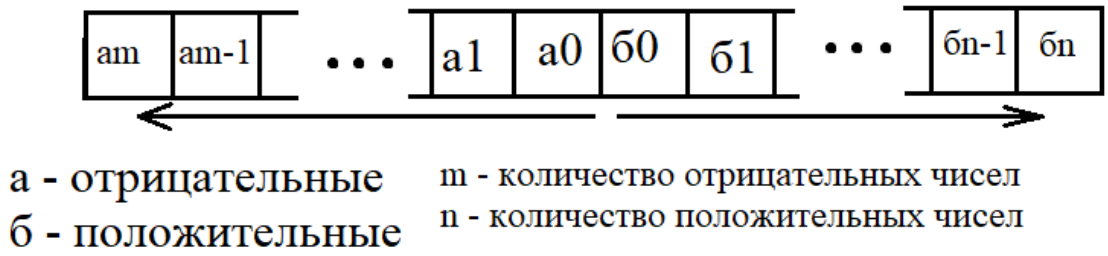


Рисунок 2 — Схема заполнения исходного массива

3. Эффективность

И сложность и скорость по времени зависят линейно от входных данных. Хотя зависимость не прямая и всё очень зависит от самих данных. Так например при 200000 элементов время выполнения программы составляла от 2.4 секунд до 3.8 секунд. Очень многое зависит от соотношения положительных и отрицательных чисел. Самый неблагоприятный вариант, когда их одинаковое количество.

Таблица 1 – Алгоритмическая сложность сортировки бинарным деревом

Сложность по времени	$O(n \times k)$
Сложность по памяти	$O(n \times k)$

4. Результат работы программы

В качестве входных данных будем подавать массивы разной длины, замеряем время выполнения для каждого случая

Таблица 2 – Результаты работы программы

Количество элементов	Время выполнения секунды
100	0.009
500	0.017
1000	0.034
5000	0.079
10000	0.452
50000	1.1
100000	1.86
200000	3.2

По таблице 2 построен график зависимости времени выполнения от количества элементов (рисунок 3)

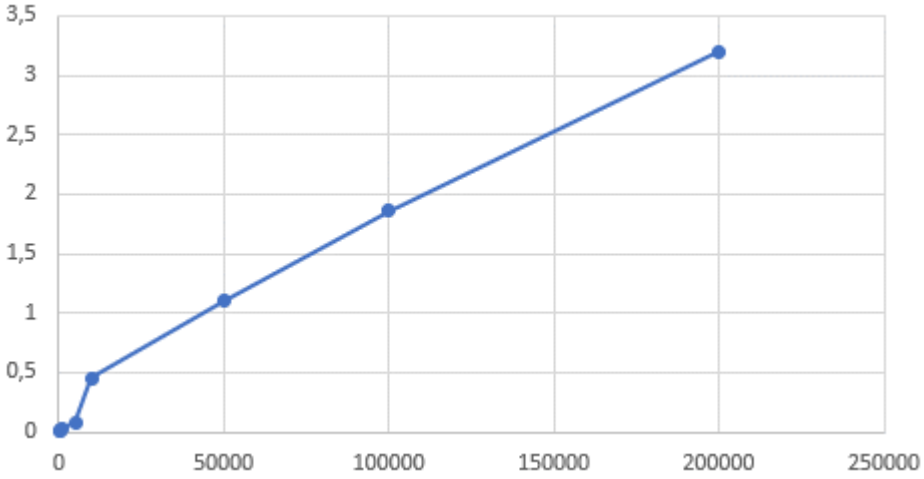


Рисунок 3 — График зависимости времени от количества элементов

5. СПИСОК ЛИТЕРАТУРЫ

1) Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е. — М.: Вильямс, 2005. — 1296 с.

2) Готтшлинг П. Современный C++ для программистов, инженеров и ученых. Серия «C++ In-Depth» = Discovering Modern C++: A Concise Introduction for Scientists and Engineers (C++ In-Depth). — М.: Вильямс, 2016. — 512 с.

3) Вирт Н. Алгоритмы и структуры данных — М.: Мир, 1989. — 360 с.

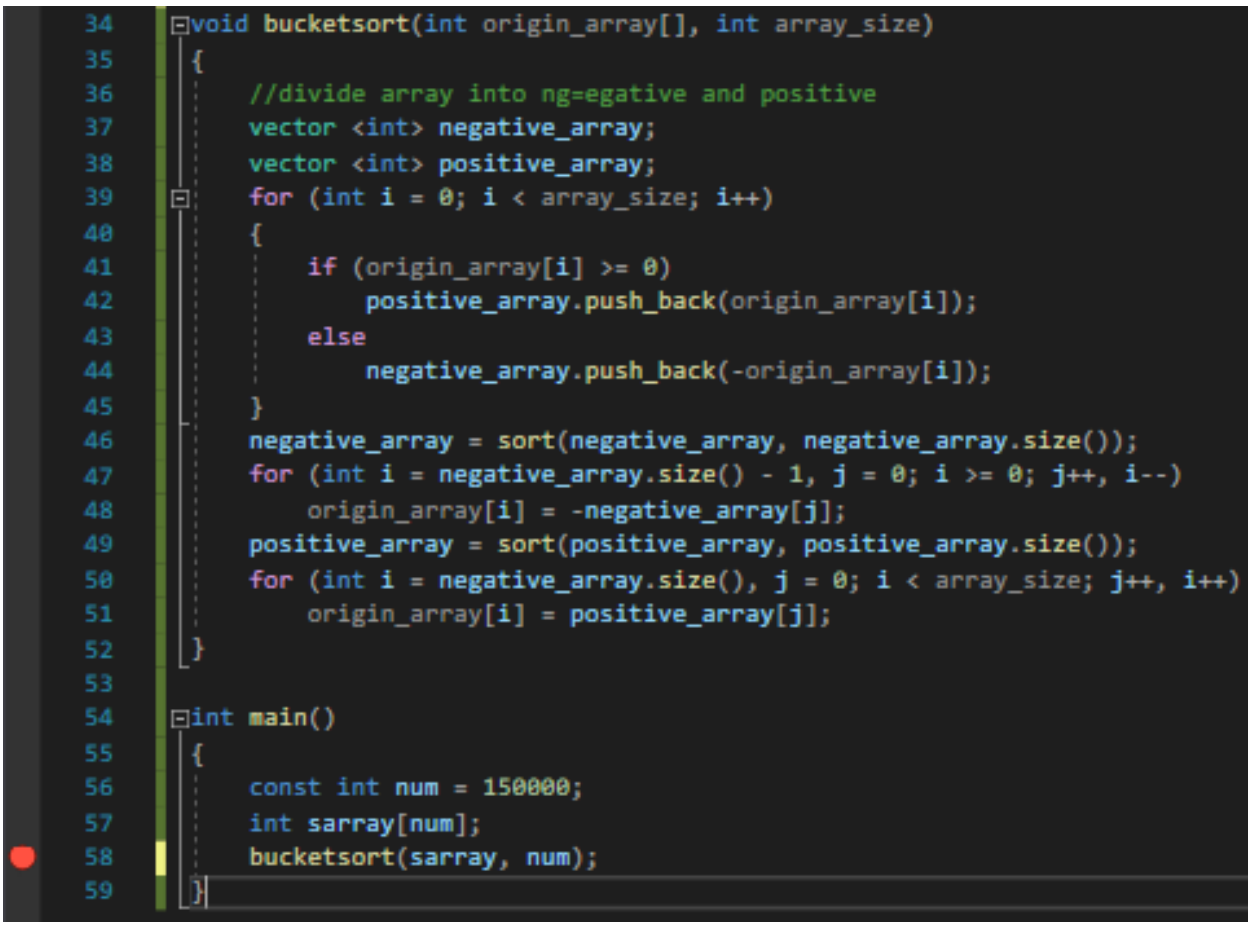
6. Приложение

Приложение 1

```cpp
#include <iostream>
#include <time.h>
#include <vector>

using namespace std;

vector <int> sort(vector <int> parray, int array_size)
{
    vector < vector <int> > arr(10, vector <int>());
    // main loop for each digit position
    for (int digit = 1; digit <= 1000000000; digit *= 10)
    {
        for (int i = 0; i < 10; i++)
            std::vector<int>().swap(arr[i]);
        // array to bucket
        for (int i = 0; i < array_size; i++)
        {
            // get the digit 0-9
            int dig = (parray[i] / digit) % 10;
            // add to bucket and increment count
            arr[dig].push_back(parray[i]);
        }
        // bucket to array
        int idx = 0;
        for (int x = 0; x < 10; x++)
        {
            for (int y = 0; y < arr[x].size(); y++)
                parray[idx++] = arr[x][y];
        }
    }
    return parray;
}
```



```
34  void bucketsort(int origin_array[], int array_size)
35  {
36      //divide array into ng=egative and positive
37      vector <int> negative_array;
38      vector <int> positive_array;
39      for (int i = 0; i < array_size; i++)
40      {
41          if (origin_array[i] >= 0)
42              positive_array.push_back(origin_array[i]);
43          else
44              negative_array.push_back(-origin_array[i]);
45      }
46      negative_array = sort(negative_array, negative_array.size());
47      for (int i = negative_array.size() - 1, j = 0; i >= 0; j++, i--)
48          origin_array[i] = -negative_array[j];
49      positive_array = sort(positive_array, positive_array.size());
50      for (int i = negative_array.size(), j = 0; i < array_size; j++, i++)
51          origin_array[i] = positive_array[j];
52  }
53
54  int main()
55  {
56      const int num = 150000;
57      int sarray[num];
58      bucketsort(sarray, num);
59  }
```