

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной
математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: А. А. Кабанов
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №1

Задача: Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.

Вариант сортировки: Поразрядная сортировка.

Вариант ключа: MD5-суммы (32-разрядные шестнадцатиричные числа).

Вариант значения: Числа от 0 до $2^{64} - 1$.

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки.

Основная идея заключается в применении устойчивой (элементы с одним и тем же значением находятся в том же порядке в выходном массиве, что и во входном) сортировки элементов по разрядам (в данном случае - от младшего разряда к старшему - LSD).

Применяемая устойчивая сортировка - сортировка подсчетом, идея которой - определение для каждого элемента x количества элементов, которые меньше x и размещение элемента x в подходящей для него позиции.

Псевдокод функции сортировки подсчетом в общем случае:

```
1 CountingSort(A, B, k):
2   C[0...k]
3   for i = 0 to k:
4     C[i] = 0
5   for j = 1 to len(A):
6     C[A[j]]++
7   for i = 1 to k:
8     C[i] = C[i] + C[i-1]
9   for j = len(A) downto 1:
10    B[C[A[j]]] = A[j]
11    C[A[j]]--
```

Общее время сортировки и количество используемой памяти - $O(k + n)$, где k - максимальный элемент входного массива, n - количество элементов во входном массиве.

Псевдокод поразрядной сортировки в общем случае:

```
1 RadixSort(A, d):
2   for i = 1 to d:
3     some-stable-sort(A, i)
```

Время работы - $O(d(n + k))$ в случае задачи сортировки n d -значных объектов, составные части которого принимают одно из k возможных значений.

Если d - константа, а $k = O(n)$, то время работы алгоритма поразрядной сортировки линейно зависит от количества входных элементов.

В контексте условия задачи и применения поразрядной сортировки нам известен аргумент $k = 16$ - количество значений разряда (т. к. шестнадцатеричные числа). В коде этот аргумент заменяется аргументом *pos* - позиции (разряду), по которой необходимо провести сортировку.

2 Исходный код

Создадим вектор элементов типа "ключ-значение" (вектор Item); заполним его содержимым непустых строк потока ввода; также создадим вектор результата (который будет выводиться после исполнения функции сортировки).

Вызовем функцию RadixSort, которая вызывает поразрядно функцию CountingSort и присваивает элементам исходного массива значения элементов массива, отсортированного по текущему разряду ключей Item.

RadixSort принимает ссылки на вектор входных данных и результирующий вектор. CountingSort принимает помимо вышеперечисленных аргументов позицию (разряд), по которой будет происходить сортировка ключей элементов. Функции имеют тип void, так как они изменяют исходные массивы-аргументы функций сортировок.

Небольшая особенность задачи заключается в прерывности значений символов строки в ASCII: с 48 по 57 позицию находятся цифры, а с 97 по 102 - буквы диапазона 'a'-'f'. Для корректного подсчета количества этих букв в каждом ключе на определенной позиции используется функция to_int().

Также для быстродействия была отключена синхронизация потоков ввода-вывода стандартной библиотеки C и C++.

После выполнения сортировки выводим полученный результирующий вектор.

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int MD5 = 16;
6
7  struct Item {
8      string key;
9      uint64_t val;
10 };
11
12 int to_int(char c) {
13     if (48 <= c && c <= 57) {
14         return c - 48;
15     } else if (97 <= c && c <= 102) {
16         return c - 97 + 10;
17     } else {
18         return 0;
19     }
20 }
21
22 void CountingSort(vector<Item>& A, vector<Item>& B, int pos) {
23     vector<int> count(MD5, 0);
24     for (size_t i = 0; i < A.size(); ++i) {
25         ++count[to_int(A[i].key[pos])];
```

```

26     }
27
28     for (size_t i = 1; i < MD5; ++i) {
29         count[i] += count[i - 1];
30     }
31
32     for (int i = static_cast<int>(A.size()) - 1; i >= 0; --i) {
33         size_t p = count[to_int(A[i].key[pos])] - 1;
34         B[p] = A[i];
35         --count[to_int(A[i].key[pos])];
36     }
37
38 }
39
40 void RadixSort(vector<Item>& A, vector<Item>& B) {
41     int maxLength = 32;
42     for (int i = maxLength - 1; i >= 0; i--) {
43         CountingSort(A, B, i);
44         for (size_t j = 0; j < A.size(); ++j) {
45             A[j] = B[j];
46         }
47     }
48 }
49
50 int main() {
51     ios::sync_with_stdio(false);
52     cin.tie(0);
53
54     vector<Item> v;
55     Item curr;
56
57     while (cin >> curr.key >> curr.val) {
58         v.push_back(curr);
59     }
60
61     vector<Item> result(v.size());
62     RadixSort(v, result);
63
64     for (size_t i = 0; i < result.size(); ++i) {
65         cout << result[i].key << '\t' << result[i].val << '\n';
66     }
67
68     return 0;
69 }

```

3 Консоль

[illegible]

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение написанной мной radix-сортировки и `std::stable_sort()`, работающей за $O(N * \log(n))$ (при условии доступности дополнительной памяти).

Тесты состоят из 10 (тесты 0-1), 10^6 (тест 2), 10^7 (тест 3) и 10^8 (тест 4) строк вида "ключ значение".

```
[anton@home lab1]$ make
g++ main.cpp -o main
g++ sort.cpp -o sort
[anton@home lab1]$ ./main < tests/test0.txt
[RADIX] Sorted in 3 microseconds.
[anton@home lab1]$ ./sort < tests/test0.txt
[CPP_SORT] Sorted in 3 microseconds.
[anton@home lab1]$ ./main < tests/test1.txt
[RADIX] Sorted in 4 microseconds.
[anton@home lab1]$ ./sort < tests/test1.txt
[CPP_SORT] Sorted in 3 microseconds.
[anton@home lab1]$ ./main < tests/test2.txt
[RADIX] Sorted in 1371239 microseconds.
[anton@home lab1]$ ./sort < tests/test2.txt
[CPP_SORT] Sorted in 1090689 microseconds.
[anton@home lab1]$ ./main < tests/test3.txt
[RADIX] Sorted in 15232532 microseconds.
[anton@home lab1]$ ./sort < tests/test3.txt
[CPP_SORT] Sorted in 13666058 microseconds.
[anton@home lab1]$ ./main < tests/test4.txt
[RADIX] Sorted in 160521969 microseconds.
[anton@home lab1]$ ./sort < tests/test4.txt
[CPP_SORT] Sorted in 185347382 microseconds.
```

Видно, что `stable_sort()` выигрывает у radix-сортировки в большинстве случаев. Я думаю, что это происходит из-за того, что в случае поразрядной сортировки мы вынуждены выполнить несколько проходов по всем символам строки и требует дополнительной памяти для хранения временных данных. Также цикл присваивания элементов (строки 44-46) замедляет работу программы, что проявляется на большом объеме входных данных (применим `std::vector::swap`).

Файл для проведения теста №4 весил около 5 Гб. Несмотря на то, что поразрядная сортировка проработала эффективнее на 15% по времени, программа потратила примерно на 7 Гб ОЗУ больше, чем программа с `stable_sort()`.

`stable_sort()` может работать быстрее и эффективнее для сортировки строк в большинстве случаев.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я познакомился с поразрядной сортировкой и сортировкой подсчета, которая является устойчивой и работает за линейное время, но требует $O(k)$ дополнительной памяти, что может сказаться на производительности при большом объеме входных данных.

Также существуют небольшие проблемы использования такой сортировки на небольших по объему входных данных: во-первых, встроенные в STL функции сортировки работают в таком случае эффективнее и по памяти, и по времени; во-вторых, если в данных всего лишь несколько элементов с большой разницей между минимумом по значению и максимумом, то создается массив, состоящий из большого количества нулей; в-третьих, неудобно сортировать элементы, которые не являются/нельзя преобразовать в целочисленные элементы.

Также я вспомнил про применение отключения синхронизации стандартных потоков C и C++, которое я использовал в контексте олимпиадного программирования.

Поразрядная сортировка - довольно мощный и хороший метод сортировки большого количества объектов, несмотря на большое использование памяти.

Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-е издание*. — Издательский дом «Вильямс», 2013. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 978-5-8459-1794-2 (рус.))
- [2] *Сортировка подсчётом* — *Википедия*.
URL: http://ru.wikipedia.org/wiki/Сортировка_подсчётом (дата обращения: 16.12.2013).
- [3] *Сортировка подсчётом* — *Википедия*.
URL: https://ru.wikipedia.org/wiki/Поразрядная_сортировка (дата обращения: 18.01.2023)
- [4] *std::stable_sort* - *cppreference*.
URL: https://en.cppreference.com/w/cpp/algorithm/stable_sort (дата обращения: 09.10.2023)