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. В. Семин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2022

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 шестнадцатеричные числа).`

Вариант значения: `числа от 0 до` 2^{64} `- 1.`

1 Описание

В интернет-ресурсе [2] для поразрядной сортировки дано следующее описание алгоритма:

Алгоритм поразрядной сортировки предназначен для сортировки целых чисел. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Так как выравнивать сравниваемые записи относительно друг друга можно в разную сторону, на практике существуют два варианта этой сортировки. Для чисел они называются в терминах значимости разрядов числа, и получается так: можно выровнять записи чисел в сторону менее значащих цифр (по правой стороне, в сторону единиц, least significant digit, LSD) или более значащих цифр (по левой стороне, со стороны более значащих разрядов, most significant digit, MSD).

При LSD сортировке (сортировке с выравниванием по младшему разряду, направо, к единицам) получается порядок, уместный для чисел. Например: 1, 2, 9, 10, 21, 100, 200, 201, 202, 210. То есть, здесь значения сначала сортируются по единицам, затем сортируются по десяткам, сохраняя отсортированность по единицам внутри десятков, затем по сотням, сохраняя отсортированность по десяткам и единицам внутри сотен, и т. п.

При MSD сортировке (с выравниванием в сторону старшего разряда, налево), получается алфавитный порядок, который уместен для сортировки строк текста. Например «b, c, d, e, f, g, h, i, j, ba» отсортируется как «b, ba, c, d, e, f, g, h, i, j». Если MSD применить к числам, приведённым в примере получим последовательность 1, 10, 100, 2, 200, 201, 202, 21, 210, 9.

Накапливать при каждом проходе сведения о группах можно разными способами — например в списках, в деревьях, в массивах, выписывая в них либо сами элементы, либо их индексы и т. п.

2 Исходный код

Для хранения необходимых данных в программе создан класс $TKeyVal$, состоящий из массива $key[33]$ и переменной val типа unsigned long long для хранения ключа и значения соответственно. После считывания данные сохраняются в векторе src, который в качестве аргумента передается в функию $RadixSort$, выполняющую уже саму поразрядную сортировку. В основе своей, очевидно, поразрядная сортировка использует сортировку подсчетом $CountingSort$, сортируя ключи 32-х битных чисел поразрядно (от меньшего разряда к большему). Сортировка подсчетом реализована следующим образом:

1. Создаем массив $count$ с размером в 16 ячеек (так как числа в шестнадцатиричной системе счисления) и инициализируем его нулями;
2. В ячейках массива $count$ инкрементируем элементы, индексы которых соответствуют разрядам сортируемых чисел;
3. Запускаем префикс-функцию по массиву $count$;
4. Обходим справа налево исходный вектор, элементы которого сопоставляем с их количеством вхождений, указанных в массиве count, и затем устанавливаем эти элементы на их новое место в выходном векторе.

Стоит отметить, что $CountingSort$ в своей реализации использует еще одну вспомогательную функцию $GetPos$, которая переводит шестнадцатиричные значения в соответствующие им числа в десятеричной системе счисления для обращения к ячейкам массива $count$. Также перегружен оператор присваивания для класса $TKeyVal$, поскольку в функции $CountingSort$ используется операция присваивания для векторов, хранящих объекты данного класса.

sort.hpp

```
1  #pragma once
2  #include <iostream>
3  #include <algorithm>
4  #include <chrono>
5  #include <vector>
6
7  typedef unsigned long long ull;
8  const int DIGITS = 32;
9
10 class TKeyVal {
11 public:
12   char key[DIGITS +1 ];
13   ull value;
14   TKeyVal& operator=(const TKeyVal& old);
```

```
15    friend bool operator< (const TKeyVal& st1, const TKeyVal& st2);
16  };
17
18  int GetPos(char c);
19  void CountingSort(std::vector<TKeyVal>& src, int id);
20  void RadixSort(std::vector<TKeyVal>& src);
```

sort.cpp

```
1   #include "sort.hpp"
2
3   TKeyVal& TKeyVal::operator= (const TKeyVal& old) {
4     for (int i = 0; i < 32; i++)
5       key[i] = old.key[i];
6     value = old.value;
7     return *this;
8   }
9
10  bool operator< (const TKeyVal& st1, const TKeyVal& st2) {
11    return (st1.key < st2.key ? true : false);
12  }
13
14  int GetPos(char c) {
15    if (c >= '0' && c <= '9')
16      return c - '0';
17    else if (c >= 'a' && c <= 'f')
18      return c - 'a' + 10;
19    else if (c >= 'A' && c <= 'F')
20      return c - 'A' + 10;
21      else return 0;
22  }
23
24  void CountingSort(std::vector<TKeyVal>& src, int id) {
25    std::vector<TKeyVal> output(src.size());
26    ull count[16] = { 0 };
27
28    for (ull i = 0; i < src.size(); i++)
29      count[GetPos(src[i].key[id])]++;
30
31    for (int i = 1; i < 16; i++)
32      count[i] += count[i - 1];
33
34    for (int i = src.size() - 1; i >= 0; i--) {
35      output[count[GetPos(src[i].key[id])] - 1] = src[i];
36      count[GetPos(src[i].key[id])]--;
37    }
38
39    src = output;
40  }
41
```

```
42 void RadixSort(std::vector<TKeyVal>& src) {
43   int DIGITS = 32;
44   for (int id = DIGITS - 1; id >= 0; id--)
45     CountingSort(src, id);
46 }
```

lab1main.hpp

```
1  #include "sort.hpp"
2
3  int main() {
4      std::ios::sync_with_stdio(false);
5    std::vector<TKeyVal> src;
6    TKeyVal st;
7    while (std::cin >> st.key >> st.value) {
8      src.push_back(st);
9    }
10
11     if (src.size() > 0) {
12         RadixSort(src);
13     }
14
15   for (ull i = 0; i < src.size(); i++) {
16     std::cout << src[i].key << '\t' << src[i].value << '\n';
17   }
18 }
```

3 Консоль

```
a@WIN-THNQL51M105:~/Desktop/DA/L1_my$ g++ -pedantic -std=c++17 -Wall l1main.cpp
sort.cpp -o l1main
a@WIN-THNQL51M105:~/Desktop/DA/L1_my$ ./l1main
00000000000000000000000000000000        13207862122685464576
ffffffffffffffffffffffffffffffff        7670388314707853312
00000000000000000000000000000000        4588010303972900864
ffffffffffffffffffffffffffffffff        12992997081104908288

00000000000000000000000000000000        13207862122685464576
00000000000000000000000000000000        4588010303972900864
ffffffffffffffffffffffffffffffff        7670388314707853312
ffffffffffffffffffffffffffffffff        12992997081104908288
```

4 Тест производительности

Тест производительности проведем на основе сравнения скорости работы поразрядной сортировки, реализованной мной в данной работе и сортировки из STL. Для сравнения сортировок будут сгенерированы тесты, состоящие из 1000000 пар входных данных вида «ключ-значение».

```
a@WIN-THNQL51M105:~/Desktop/DA/L1_my ./wrapper.sh
[info][Sun Mar 27 22:32:57 MSK 2022] Stage #1. Compiling...
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable sort.o l1main.cpp
-o l1main
[info][Sun Mar 27 22:32:57 MSK 2022] Stage #2. Test generating...
[info][Sun Mar 27 22:32:57 MSK 2022] Stage #3. Checking...
[info][Sun Mar 27 22:32:57 MSK 2022] tests/01.t,lines = 295 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/02.t,lines = 843 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/03.t,lines = 330 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/04.t,lines = 258 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/05.t,lines = 467 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/06.t,lines = 62 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/07.t,lines = 400 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/08.t,lines = 174 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/09.t,lines = 930 OK
[info][Sun Mar 27 22:32:57 MSK 2022] tests/10.t,lines = 278 OK
[info][Sun Mar 27 22:32:57 MSK 2022] Stage #4 Benchmark test generating...
[info][Sun Mar 27 22:33:11 MSK 2022] Stage #5 Benchmarking...
make: 'benchmark'is up to date.
[info][Sun Mar 27 22:33:11 MSK 2022] Running benchmark.t
Count of lines is 1000000
Counting sort time: 5282673us
STL Sort time: 3603774us
```

По данным результатам видно, что для текущего размера входных данных сортировка STL работает быстрее, чем поразрядная. Стоит разобраться, почему так.

Сложность сортировки STL O(n*logn), а сложность порязрядной сортировки O(n). Однако, если не пренебрегать константой, то сложность порязрядной O(32*n), так как числа 32-х разрядные. Таким образом, вычислив n, обнаружим, что поразрядная сортировка начнет выигрывать по времени сортировку STL при размере входных данных $n > 2^{32}$.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я были изучены различные виды сортировок и реализовал два алгоритма сортировки за линейное время – поразрядная сортировка и сортировка подсчётом.

Сортировка за линейное время наиболее эффективна при обработке большого количества данных. Сложность сортировок этого типа – O(d*n), где d – количество разрядов, по которым происходит сортировка, а n – объём входных данных.

Также важным фактором линейных сортировок является их устойчивость – элементы с одинаковыми ключами не меняют порядок в отсортированном наборе данных.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Поразрядная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 14.03.2022).

[3] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).