

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: М. В. Спиридонов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2017

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: Сортировка подсчётом.

Вариант ключа: Числа от 0 до 65535.

Вариант значения: Сортировка подсчётом по целочисленному ключу (числа от 0 до 65535). В качестве значения используются строки переменной длины (до 2048 символов)

1 Описание

Условие задачи подразумевает хранение строк, но использовать STL было запрещено и пришлось написать свой класс TString, который реализует все необходимые функции от строки (сравнение, копирование и присваивание). По сути была написана лишь красивая обёртка над `char*`.

Также нужно было как-то хранить пары «ключ-значение», поэтому был реализован свой класс TPair, который содержит два публичных поля: `Key(int)`, `Value(TString)`. Из основных элементов не хватало только динамического массива, поэтому был реализован свой TVector, который позволяет динамически добавлять элементы TPair и производить манипуляции с ними.

Все основные типы данных есть, осталась сортировка подсчётом и ввод данных.

Есть смысл сказать о методе ввода значений. Был создан буфер `char*` в 2048 символов, куда при помощи `scanf("%s")` считывалась строка до EOF, и позднее заносилась в TPair, который в свою очередь попадал в TVector.

Сортировка подсчётом имеет алгоритмическую сложность $O(n + k)$.

Применима в случае, если сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством.

Суть сортировки заключается в нахождении максимального ключа среди всех входных и создании дополнительного целочисленного массива, где будет подсчитано сколько каких ключей встречено. Далее в зависимости от значения ключа, в результирующий массив будут расставлены все исходные пары.

Я использовал стабильную версию сортировки для структур, рассмотренной на лекции.

2 Описание программы

По условию задачи было ясно, что красиво такую программу написать в одном файле не получится и придется её разбить на несколько основных файлов:

1. main.cpp (содержит основной метод main, метод вывода TVector'a - Print)
2. Model.h и Model.cpp (описание и реализация класса TPair)
3. TString.h и TString.cpp (описание и реализация класса TString)
4. TVector.h и TVector.cpp (описание и реализация класса TVector)

Для удобства сборки проекта были написаны:

1. makefile (для сборки и автоматического тестирования проекта из консоли)
2. CMakeLists.txt (для автоматической сборки проекта в CLion, Visual Studio Code, Visual Studio 17)

Также были взяты "test_generator.py" и "wrapper.sh" с GitHub'a[1] и переделаны.

| main.cpp | |
|---|--|
| void Print (TVector &inp) | Метод для вывода вектора |
| int main (int argc, char **argv) | Точка входа в программу |
| Model.h (описание методов/полей) | |
| TPair () | Пустой конструктор |
| TPair (const TPair &that) | Конструктор копирования |
| TPair (int key, TString value) | Конструктор создания с параметрами |
| TPair &operator = (const TPair &that) | Оператор присваивания |
| ~TPair () | Деструктор |
| int Key | Поле ключа |
| TString Value | Поле значения |
| TString.h (описание методов/полей) | |
| TString (const char *string =) | Конструктор создания из char* |
| TString (TString const &string) | Конструктор копирования |
| TString &operator = (const TString &string) | Оператор присваивания |
| TString &operator = (const char *string) | Оператор присваивания |
| bool operator == (const TString &string) const | Оператор сравнения |
| char &operator [] (int) const | Оператор обращения по индексу |
| char &operator [] (int) | Оператор обращения по индексу |
| const size_t Length () const | Метод, который возвращает длину строки |
| char *GetCharArray () | Метод получения внутреннего char* |
| char *GetCharArray () const | Метод получения внутреннего char* |
| ~TString() | Деструктор |
| static int const SIZE_DIFFERENCE = 1 | Приватная константа |
| char *privateString | Приватное поле типа char* |
| static int const DEFAULT_INT_VALUE = 0 | Приватная константа |

| TVector.h (описание методов/полей) | |
|--|---|
| TVector () | Пустой конструктор |
| TVector (const TVector&inp) | Конструктор копирования |
| TVector (const int n) | Конструктор создания |
| void FastPushBack (TPair &data) | Метод добавления элемента |
| void Clear () | Метод опустошения вектора |
| int Size () const | Метод, который возвращает размер вектора |
| int Capacity () const | Метод, который возвращает реальное количество элементов в векторе |
| void CountingSort() | Метод сортировки подсчётом |
| ~TVector () | Деструктор |
| TPair *privateArray | Приватное поле вектора |
| int privateArraySize | Приватное поле размера вектора |
| int privateArrayOccupiedSize | Приватное поле, которое содержит количество реально занятых ячеек вектора |
| static int const SIZE_DIFFERENCE = 1 | Приватная константа |
| static int const DEFAULT_INT_VALUE = 0 | Поле значения |

3 Код программы

Метод сортировки подсчётом

```
1 void TVector::CountingSort() {
2     int* tmp = new int[max_key + 1];
3     TElem* result = new TElem[size];
4
5     for(int i = 0; i < max_key + 1; ++i) {
6         tmp[i] = 0;
7     }
8     for(int i = 0; i < size; ++i) {
9         ++tmp[array[i].key];
10    }
11    for(int i = 1; i < max_key + 1; ++i) {
12        tmp[i] += tmp[i - 1];
13    }
14    for(int i = size - 1; i >= 0; --i) { //
15        result[--tmp[array[i].key]] = array[i];
16    }
17    delete [] array;
18    delete [] tmp;
19    array = result;
20 }
```

4 Код дополнительных файлов

- makefile

```
1 | FLAGS=-g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
2 | CC=g++
3 |
4 | all: string model vector main
5 |
6 | main: main.cpp
7 |     $(CC) $(FLAGS) -c main.cpp
8 |     $(CC) $(FLAGS) -o CounterSort TString.o Model.o TVector.o main.o -lm
9 |
10 | vector: TVector.cpp
11 |     $(CC) $(FLAGS) -c TVector.cpp
12 |
13 | string: TString.cpp
14 |     $(CC) $(FLAGS) -c TString.cpp
15 |
16 | model: Model.cpp
17 |     $(CC) $(FLAGS) -c Model.cpp
18 |
19 | clear:
20 |     rm -f *.o
21 |     rm -fr *.dSYM
22 |
23 | runtests:
24 |     rm -rf tests
25 |     mkdir tests
26 |     sh wrapper.sh
27 |
28 | bench: benchmark.cpp
29 |     $(CC) $(FLAGS) -c benchmark.cpp
30 |     $(CC) $(FLAGS) -o benchmark benchmark.o -lm
```

- test_generator.py

```
1 | import random
2 | import string
3 |
4 | MAX_KEY_VALUE = 65535
5 | MAX_VALUE_LEN = 500
6 |
7 | def generate_random_value():
8 |     return "".join([random.choice(string.ascii_lowercase)
9 |                     for _ in range(1, MAX_VALUE_LEN)])
10 |
11 | if __name__ == "__main__":
12 |     for num in range(1, 21):
13 |         values = list()
```



```

14     output_filename = "tests/{:02d}.t".format(num)
15     with open(output_filename, 'w') as output:
16         for _ in range(0, random.randint(10, 100)):
17             key = random.randint(0, MAX_KEY_VALUE)
18             value = generate_random_value()
19             values.append((key, value))
20             output.write("{}\t{}\n".format(key, value))
21     # Answer.
22     # values[0][0] -- key
23     # values[0][1] -- value
24     output_filename = "tests/{:02d}.a".format(num)
25     with open(output_filename, 'w') as output:
26         values = sorted(values, key=lambda x: x[0])
27         for value in values:
28             output.write("{}\t{}\n".format(value[0], value[1]))

```

- benchmark.cpp

```

1  #include <algorithm>
2  #include <cassert>
3  #include <ctime>
4  #include <iostream>
5  #include <vector>
6
7  bool CheckOrder (const std::vector <std::pair <int, std::string> > &data) {
8      for (size_t idx = 1; idx < data.size(); ++idx) {
9          if (data[idx - 1].first > data[idx].first) {
10             return false;
11         }
12     }
13     return true;
14 }
15
16 bool PairComparer (const std::pair <int, std::string> &a, std::pair <int, std:::
17     string> &b) {
18     return a.first < b.first;
19 }
20 void CountingSort (std::vector <std::pair <int, std::string> >&inp) {
21     int size = inp.capacity();
22     int max = 0;
23     for (int i = 0; i < size; i++) {
24         if (inp[i].first > max) {
25             max = inp[i].first;
26         }
27     }
28     std::vector <std::pair <int, std::string> >res(size);
29     int *countMass = new int[max + 1];
30     for (int i = 0; i < max + 1; ++i) {
31         countMass[i] = 0;

```

```

32     for (int i = 0; i < size; ++i) {
33         countMass[inp[i].first]++;
34     }
35
36     for (int i = 1; i <= max; ++i) {
37         countMass[i] = countMass[i] + countMass[i - 1];
38     }
39
40     for (int i = size - 1; i >= 0; i--) {
41         int tmp = countMass[(inp[i].first) - 1];
42         res[tmp] = inp[i];
43         countMass[inp[i].first] = countMass[inp[i].first] - 1;
44     }
45     for (int i = 0; i < size; ++i) {
46         inp[i] = res[i];
47     }
48     delete[] countMass;
49 }
50 void StdSort (std::vector <std::pair <int, std::string> > &data) {
51     std::sort(data.begin(), data.end(), PairComparer);
52 }
53
54 void RunStdSort (std::vector <std::pair <int, std::string> > &data) {
55     clock_t start = clock();
56     StdSort(data);
57     clock_t end = clock();
58     assert(CheckOrder(data) == true);
59     double time = (double)(end - start);
60     printf("\nStd SortTime = %lf\n",time/CLOCKS_PER_SEC);
61 }
62 void RunCountingSort (std::vector <std::pair <int, std::string> > &data) {
63     clock_t start = clock();
64     CountingSort(data);
65     clock_t end = clock();
66     assert(CheckOrder(data) == true);
67     double time = (double)(end - start);
68     printf("\nCounting SortTime = %lf\n",time/CLOCKS_PER_SEC);
69 }
70
71 int main (int argc, const char *argv[]) {
72     std::vector <std::pair <int, std::string> > data;
73     int tempInt = 0;
74     char *bufStr = new char[2048];
75     std::pair <int, std::string> pair;
76     while (scanf("%d", &tempInt) != EOF) {
77         scanf("%s", bufStr);
78         pair.first = tempInt;
79         pair.second = bufStr;
80

```

```

81 |         data.push_back(pair);
82 |     }
83 |     delete[] bufStr;
84 |     std::vector <std::pair <int, std::string> > dataCopy(data);
85 |     RunStdSort(data);
86 |     RunCountingSort(data);
87 |     return 0;
88 | }

```

5 Консоль

```
$ CounterSort make
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c TString.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c Model.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c TVector.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c main.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -o CounterSort TString.o Model.o TVector.o main.o -lm

$ CounterSort cat 01.t
60693 nmbfyzfqu
43600 piqyjgkhf
25976 irudfyioj
3679 duqjersbu
38028 wuxtiogk
3639 mefjaasbe
3915 loxlyyyqcs
42023 sslttackw
58339 zoxkqedkb
$ CounterSort ./CounterSort < 01.t > 01.answ
$ CounterSort cat 01.answ
3639 mefjaasbe
3679 duqjersbu
3915 loxlyyyqcs
25976 irudfyioj
38028 wuxtiogk
42023 sslttackw
43600 piqyjgkhf
58339 zoxkqedkb
60693 nmbfyzfqu
$ CounterSort cat 02.t
14517 tyxzddarc
64044 xelwhtsro
39337 kwktcfiiu
9162 qjevarqqqs
64981 mrwlghlcs
```

```
9480 vagjwzilk
44021 egaeqezha
41017 xpcfbjuly
51997 jyjntrrgf
41622 pkabowgux
44235 yborezsfg
$ CounterSort ./CounterSort < 02.t > 02.answ
$ CounterSort cat 02.answ
9162 qjevarqq
9480 vagjwzilk
14517 tyxzddarc
39337 kwktcfiiu
41017 xpcfbjuly
41622 pkabowgux
44021 egaeqezha
44235 yborezsfg
51997 jyjntrrgf
64044 xelwhtsro
64981 mrwlghlcs
```

6 Проверка на утечки памяти

Тестирование проводилось на файле в 100 000 строк, на системе Arch Linux, с помощью утилиты Valgrind.

```
[Cp окт 04: 05:53] snowden@arch ~/Projects/DA/da1 [master] $ make
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c TString.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c Model.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c TVector.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -c main.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -O2 -o CounterSort TString.o Model.o TVector.o main.o -lm
[Cp окт 04: 05:54] snowden@arch ~/Projects/DA/da1 [master] $
valgrind ./CounterSort < tests/01.t > tmp
==1417== Memcheck, a memory error detector
==1417== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1417== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1417== Command: ./CounterSort
==1417==
==1417==
==1417== HEAP SUMMARY:
==1417== in use at exit: 0 bytes in 0 blocks
==1417== total heap usage: 1,517,400 allocs, 1,517,400 frees, 9,380,563 bytes
allocated
==1417==
==1417== All heap blocks were freed -- no leaks are possible
==1417==
==1417== For counts of detected and suppressed errors, rerun with: -v
==1417== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[Cp окт 04: 05:54] snowden@arch ~/Projects/DA/da1 [master] $ wc tests/01.t
100000 200000 1083130 tests/01.t
```

7 Тест производительности

Чтобы увидеть разницу были созданы файлы размером 10^3 , 10^5 , 10^6 , $10 * 10^6$ входных строк и был создан файл бенчмарка, с реализацией сортировки подсчетом и QSort из STD. Исходный код лежит в «дополнительных файлах» выше.

```
$ CounterSort make bench
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
-c benchmark.cpp
g++ -g -std=c++11 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long -O2
-o benchmark benchmark.o -lm
$ CounterSort wc tests/01.t
1000    2000    15797 tests/01.t
$ CounterSort ./benchmark < tests/01.t
Std SortTime = 0.000080
Counting SortTime = 0.000241

$ CounterSort wc tests/02.t
100000  200000  1083096 tests/02.t
$ CounterSort ./benchmark < tests/02.t
Std SortTime = 0.010439
Counting SortTime = 0.005754

$ CounterSort wc tests/03.t
100000  200000  1083096 tests/03.t
$ CounterSort ./benchmark < tests/03.t
Std SortTime = 0.095891
Counting SortTime = 0.050415

$ CounterSort wc tests/04.t
10000000 20000000 108302115 tests/04.t
Std SortTime = 0.984407
Counting SortTime = 0.854098
```

8 Дневник отладки

1. 29.09.17; 0:07; Сел за написание лабораторной. Написал TString, TPair, TVector. Долго искал информацию по грамотному разделению шаблона на несколько файлов;
2. 1.10.17; 0:49; Первый провал на системе тестирования, убрал gm из makefile;
3. 1.10.17; 1:04; Неверный ответ на первом тесте, программа теряла где-то одно значение. Забыл убрать отладочную строчку с удалением корневого элемента;
4. 1.10.17; 23:24; Превышение временного порога на 10-м тесте. На самом деле я даже не удивился. Изменять размер вектора при каждом добавлении элемента само по себе не очень решение. Исправлял введением доп. переменной в вектор, которая считала сколько всего элементов реально заполнены и в случае её равенства размеру массива, увеличивал размер в 2 раза;
5. 2.10.17; 2:37; Решил дописать кучу никому не нужных методов и оберток, чтобы не нарушать инкапсуляцию;
6. 2.10.17; 4:55; Превышение временного порога на 12-м тесте. Поправил начальный размер вектора с 1-го элемента до 32-х;
7. 2.10.17; 4:59; Программа получила статус «Ок» на системе автоматического тестирования;
8. 2.10.17; 18:52; Новая версия программы с исправлениями утечек памяти провалила по времени 12-й тест, но это лишь потому что я решил попробовать заслать с начальным массивом в 8 элементов. Также запускал на разных машинках, дописывал код;
9. 2.10.17; 21:41; Решил самоутвердиться и заслал финальную версию программы, которую тестировал пару-тройку часов;
10. 3.10.17; 5:57; Финальная отправка на проверку, перед отправкой отчёта. Внесены мелкие правки по кодстайлу;
11. 3.10.17; 6:27; Засылаю отчёт;
12. 6.10.17; 14:54; Исправил отчёт. Нашёл ошибку в бенчмарке - он показывал не правильное время. Исправил замеры времени, также исправил нарушение инкапсуляции. Пришлось отказаться от шаблона в виде TVector.h+TVector.hpp и написать лишь одну его реализацию, которая зависит только от TPair. Также метод сортировки был перенесён в TVector.cpp.

9 Выводы

Сортировка подсчётом применима, если сортируемые числа имеют диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством. Само написание программы не вызвало особых затруднений. Куда больше сил и времени было потрачено на всевозможные тесты и приведение кода в презентабельный вид.

Весьма много времени ушло на сравнение моей сортировки с другими. Для себя я выяснил, что тестировать все на железе i7 4/8 + SSD (1400мб/1200мб чтение/запись в сек) глупая затея, ибо чтобы увидеть хоть какую-то разницу нужно создавать входные файлы $> 10^{18}$ строк. Также написание лабораторной мне показалось чем-то схожим с хакатоном, которые я регулярно посещаю: быстро и красиво написать что-то почти не реально. Написанный за ночь код, нужно доводить еще пару тройку дней до совершенства, проводя уйму времени за тестами.

На вооружение забрал себе Radix, включающий в себя сортировку подсчётом. Поэтому что все проведённые мною тесты показали лучшие результаты именно на нем.

Список литературы

- [1] *GitHub одного из семинаристов.*
URL: <https://github.com/toshunster/da> (дата обращения: 29.09.2017).
- [2] *Шаблоны функций в C++.*
URL: <http://cppstudio.com/post/5165/> (дата обращения: 30.09.2017).
- [3] *Шаблоны функций в C++.*
URL: <http://cppstudio.com/post/673/> (дата обращения: 1.10.2017).
- [4] *Функция `strcmp`.*
URL: <http://cppstudio.com/post/686/> (дата обращения: 1.10.2017).
- [5] *Сортировка подсчётом — Википедия.*
URL: http://ru.wikipedia.org/wiki/Сортировка_подсчётом (дата обращения: 16.12.2013).
- [6] 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.))
- [7] *Bitbucket лектора.*
URL: <https://bitbucket.org/nkmakarov/da4students/> (дата обращения: 2.10.2017).