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. О. Дубинин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

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: Телефонные номера, с кодами стран и городов в формате +<код страны>-<код города>-телефон.

Вариант значения: Строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. Числа сортируются сортировкой подсчётом по каждому разряду. Согласно [2] Существует два варианта: least significant digit (LSD) и most significant digit (MSD). При LSD сортировке, сначала сортируются младшие разряды, затем старшие. В [1] указано: «Сначала производиться сортировка по младшей цифре».

Для чисел удобнее использовать разряды не в десятичном представлении, а несколько битов двоичного представления, так как для получения цифры числа в десятичном представлении используется ресурсоёмкая операция целочисленного деления на степени 10,а для двоичного лёгкие операции логического сдвига.

Согласно тому всё же[1] и материалам лекций для n b-битовых чисел и цифр из r битов алгоритм поразрядной сортировки выполнит сортировку за время $O(\frac{b}{r}(n+2^r))$. Выбирать r нужно в зависимости от $\log_2 n$. Если $b < \log_2 n$, то $r = b$, иначе $r = \log_2 n$.

Непосредственно про реализацию сортировки:

Сперва создается временный вспомогательный массив B равный исходному, затем массив чисел C размеров 2^r в котором будет проходить сортировка посчётом. Число разбивается на непересекающиеся части по r бит и сортируется посчётом по ним, начиная с младшего разряда.

Конкретнее про сортировку подсчетом.

Совершается проход по исходному массиву, для каждой цифры значение элемента C на позиции со значением этой цифры увеличивается на 1. Затем элементы массива C последовательно суммируются, таким образом в массиве C значение по индексу цифры равно количеству строго меньших чем этот индекс элементов. Идем по изначальному массиву назад(для того, чтобы сохранить устойчивость): для j элемента этого массива в $C[B[j]-1]$ содержится корректный индекс этого элемента в массиве (ведь индексами ниже располагаются элементы строго меньше его). Уменьшаем значение $C[B[j]]$ на 1 чтобы следующий элемент с таким же значением не записался поверх этого.

Копируем в вспомогательный массив отсортированный по текущему разряду массив и сортируем по следующему.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $TData$, в которой будем хранить ссылку на ключ, ключ в целочисленном виде и ссылку на значение.

main.cpp	
int main()	Основная функция. В ней происходит ввод данных и управление векторами

Далее необходимо реализовать класс векторов. В векторе будет храниться массив структур $TData$, который будет динамически расширяться при необходимости.

TVector.h, TVector.cpp	
TVector(long long size, long long capacity)	Конструктор. Создает пустой вектор длины аргумента функции size (с capacity равным аргументу функции capacity)
long long Size()	Возвращает размер вектора
void Print()	Выводит вектор
void PushBack(TData elem)	Вставляет элемент в конец вектора
TData &operator[ ](size_t ix)	Оператор, который возращает элемент под индексом ix TData из массива структур TData вектора
TVector &operator(TVector &vector)	Оператор присваивание одного вектора к другому.
∼ TVector()	Деструктор вектора. Уничтожает массив TData

Листинг класса.

```
1  struct TData{
2      long keyInNum;
3      char *key;
4      char *val;
5  };
6
7  class TVector {
8  private:
9      TData* arr;
10     long long vec_size;
11     long long capacity;
12 public:
13     TVector(long long size, long long capacity);
```

```
    long long Size();
    void Print();
    void PushBack(TData elem);
    TData &operator[](size_t ix);
    TVector &operator= (TVector &vector);
    ~TVector();
};
```

Подробнее выделим функции сортировки(и их вспомогательные функции), поскольку это ключевая часть лабораторной работы

```
int Log(size_t a, size_t b) {
    return (int)(log(b) / log(a));
}

int GetDigit(long n, int r, int numSys, int i) { //    r
    return (n >> (r * i)) & (numSys - 1);
}

int GetSortPar(long n) { //-
    if (PHONES_BITS < Log(2, n)) {
        return PHONES_BITS;
    } else {
        return (int)Log(2, n);
    }
}

void CountingSort(TVector &vec, int index, int r, int numSys) {
    long *tmp_array = new long[numSys];

    //vector of result
    TVector result(vec.Size(), vec.Size());

    for(int i=0; i<numSys; ++i) {
        tmp_array[i] = 0;
    }

    //run of data array
    for (int indexI = 0; indexI < vec.Size(); ++indexI) {
        tmp_array[GetDigit(vec[indexI].keyInNum,r,numSys,index)]++;
    }

    //adding priviously
    for(int i=1; i<=numSys; ++i) {
        tmp_array[i] += tmp_array[i-1];
    }

    for (int j = vec.Size()-1; j >= 0 ; --j) {
        TData tmp_data = vec[j];
        result[tmp_array[GetDigit(vec[j].keyInNum,r,numSys,index)]-1] = tmp_data;
```

```
40          tmp_array[GetDigit(vec[j].keyInNum,r,numSys,index)]--;
41      }
42
43      vec = result;
44
45      delete[] tmp_array;
46
47  }
48
49
50
51  void RadixSort(TVector &vec) {
52
53      if (vec.Size() < 2) {
54          return;
55      }
56
57      int r = GetSortPar(vec.Size()); //
58      int numSys = pow(2, r); //
59      int phoneDigits = ceil((float)PHONES_BITS / r); //
60
61
62      for (int index = 0; index < phoneDigits ; ++index) {
63          CountingSort(vec,index, r, numSys);
64
65      }
66
67  }
```

3 Консоль

```
art@mars:~/study/semester_3/DA/lab_1/lab1$ ls
lab1  lab1.cpp  lab1.tar  Makefile  report  test  tmp  TVector.cpp  TVector.h
art@mars:~/study/semester_3/DA/lab_1/lab1$ cat test
+7-495-1123212 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
+7-495-1123212 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
+375-123-1234567 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
+375-123-1234567 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aa
art@mars:~/study/semester_3/DA/lab_1/lab1$ make
g++ -std=c++11 -g -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm -c lab1.cpp
g++ -std=c++11 -g -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm -c TVector.cpp
g++ -std=c++11 -g -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm -o lab1 lab1.o TVector.o
art@mars:~/study/semester_3/DA/lab_1/lab1$ ./lab1 <test
+7-495-1123212 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naatt
+7-495-1123212 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naa
+375-123-1234567 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3naat
+375-123-1234567 n399tann9nnt3ttnaaan9nann93na9t3a3t9999na3aan9antt3tn93aat3na
art@mars:~/study/semester_3/DA/lab_1/lab1$
```

4 Тест производительности

Поскольку поразрядная сортировка устойчива (стабильна), то имеет смысл сравнивать её с std::stable_sort, а не с std::sort.

Тест производительности представляет из себя следующее: сортировка данных с помощью поразрядной сортировки сравнивается с стандартной библиотечной std::stable_sort. Тесты состоят из 1 тысячи, 1 миллиона и 15 миллионов строк случайно сгенерированных пар входных данных. Тесты генерируются отдельной программой, написанной на языке Python.

```
art@mars:~/study/semester_3/DA/lab_1/lab1$ ls
benchmark      lab1      lab1.o    Makefile  test  TVector.cpp  TVector.o
benchmark.cpp  lab1.cpp  lab1.tar  report    tmp   TVector.h
art@mars:~/study/semester_3/DA/lab_1/lab1$ g++ -std=c++11 -g -pedantic -Wall
-Werror -Wno-sign-compare -Wno-long-long -lm benchmark.cpp TVector.cpp -o benchmark
art@mars:~/study/semester_3/DA/lab_1/lab1$ ./benchmark <../tests/1k.t
Sorting: radix
Time of working 0.475ms.

Sorting: std::stable_sort
Time of working 0.398ms.

art@mars:~/study/semester_3/DA/lab_1/lab1$ ./benchmark <../tests/1kk.t
Sorting: radix
Time of working 521.863ms.

Sorting: std::stable_sort
Time of working 641.192ms.

art@mars:~/study/semester_3/DA/lab_1/lab1$ ./benchmark
../tests/15kk.t
Sorting: radix
Time of working 8611.44ms.

Sorting: std::stable_sort
Time of working 11757.1ms.
```

Листинг генератора тестов.

```
1 # test_generator.py
```

```
import random
import string

MAX_KEY_VALUE = 100
MAX_VALUE_LEN = 10

def generate_random_value():
    return "".join( [ random.choice( string.ascii_lowercase ) for _ in range( 1,
        MAX_VALUE_LEN ) ] )

def generate_random_telephone_number():
    seq = (str(random.randint(1, 999)), str(random.randint(1, 999)), str(random.randint
        (1, 9999999999)));
    return "+" + "-".join(seq)

if __name__ == "__main__":
    output_filename = "tests/15kk.t"
    with open( output_filename, 'w') as output:
        for _ in range( 0, 15000000 ):
            key = generate_random_telephone_number()
            value = generate_random_value()
            values.append( (key, value) )
            output.write( "{}\t{}\n".format(key, value) )
    output_filename = "tests/1k.t"
    with open( output_filename, 'w') as output:
        for _ in range( 0, 1000 ):
            key = generate_random_telephone_number()
            value = generate_random_value()
            values.append( (key, value) )
            output.write( "{}\t{}\n".format(key, value) )
    output_filename = "tests/1kk.t"
    with open( output_filename, 'w') as output:
        for _ in range( 0, 1000000 ):
            key = generate_random_telephone_number()
            value = generate_random_value()
            values.append( (key, value) )
            output.write( "{}\t{}\n".format(key, value) )
```

В reference[4] указано,что std::stable_sort производит до $N * \log_2(N)$ сравнений элементов, если предоставлено достаточно оперативной памяти. Поразрядная сортировка выполняется за линейное время ($O(N)$). Теоретические расчёты расходятся с реальными данными по нескольким причинам:

1. В процессе сортировки создаётся временный массив и для каждого разряда (нескольких битов) происходит копирование из него в исходный. Поскольку нам не важны значения в вспомогательном массиве, то можно поочерёдно использовать как вспомогательный текущий и временный.

2. Реализация класса векторов для этой лабораторной далека от идеала и не вы-

держивает сравнения с уже реализованным в stl. Как пример можно привести использование шаблонов, оно убирает необходимость писать отдельный класс для каждого типа значений.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я изучил сортировки за линейное время, осознал преимущества и недостатки отсутствия сравнений в сортировке и научился реализовывать эти сортировки. Я изучил работу и опробовал на практике ввод и вывод данных в C++, методы std::istream. Научился обходиться без stl и реализовать его элементы самостоятельно.

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. — 1328 с. (ISBN 878-5-8459-1794-2 (рус.))

[2] *Поразрядная сортировка — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 07.10.2017).

[3] ГОСТ Р 7.05-2008

[4] *cplusplus.com* std::stable_sort

[5] *cplusplus.com* Random Access Iterator