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: П. А. Харьков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата: 14.10.2020
Оценка:
Подпись:

Москва, 2020

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.
Поразрядную сортировку можно реализовать, к примеру, используя сортировку подсчетом, необходимо лишь разбить элементы на «блоки» и сортировать по ним. Идеей сортировки подсчетом является нахождение для каждого элемента количества элементов, меньших по значению. Пусть мы разбили n чисел на блоки от 0 до k, тогда сложность сортировки подсчетом является равной $O(n + k)$, а значит сложность поразрядной сортировки является $O(d(n + k)$, где d - количество «блоков» в сортирующихся элементах. Так как мы сами фиксируем k и d, то сложность получается линейно зависящей от n.
Согласно [1] сложность поразрядной сортировки можно определить более точно - $O((b/r) * (n + 2^r))$, где b - количество байт во всем числе, r - количество байт в «блоке». Тогда можно заметить зависимость, что при

- $b > log(n)$, выгоднее всего использовать $r = log(n)$, так как при использовании большего r, 2^r будет превышать n.
- $b \leq log(n)$, выгоднее использовать r, как можно более приближенное к b.

Так как нам даны MD5-суммы, размер которых равен 128 битам, то r, желательнее всего, сделать равным 128 битам. Но из-за ограничений по выделению памяти в компьютерных системах, каждый «блок» MD5-суммы будет хранить 16 бит. То есть сложность моей поразрядной сортировки будет равна $O(8 * (n + 2^8))$.

2 Исходный код

До тех пор, пока программа получает данные из стандартного потока ввода, она считывает пару значений: MD5-сумму и число от 0 до $2^{64} - 1$ и добавляет в вектор эту пару. Затем программа сортирует вектор, с помощью поразрядной сортировки и выводит значения вектора в стандартный поток вывода, разделяя элементы с помощью табуляции.

Для того, чтобы поразрядная сортировка была достаточно быстрой при большом количестве входных данных, MD5-сумма разделена на блоки, в каждом определено целое десятичное число равное значению нескольких последовательно идущих разрядов MD5-суммы. В данное случае я разбиваю MD5-сумму на 8 блоков, в каждом из которых находится десятичное значение 4 разрядов. Листинг считывания и обработки MD5-суммы:

```
1  std::istream& operator>> (std::istream &in, TMd5& md5){
2      std::string str; // Считывается md5-сумма
3      if (!(in >> str)){
4          return in;
5      }
6
7      for (int block = 0; block < MD5_BLOCKS; ++block){
8          unsigned short blockVal = 0; // Значение блока
9          for (int i = 0; i < BLOCK_LEN; ++i){
10             blockVal = blockVal << BITS; // Сдвигается текущее значение блока на 4 бита
                   (столько необходимо для 16-ного числа)
11             char c = str[block * BLOCK_LEN + i];
12             // К значению блока прибавляется значение следующего разряда
13             if ('0' <= c && c <= '9'){
14                 blockVal += c - '0';
15             }else{
16                 blockVal += c - 'a' + DECIMAL;
17             }
18         }
19         md5.Blocks[block] = blockVal;
20     }
21     return in;
22 }
```

Функция поразрядной сортировки получает на вход ссылку на вектор (v) и ничего не возвращает. Сортировка работает следующим способом:

```
1  using TUll = unsigned long long;
2  using TVector_pair = NVector::TVector<TPair<NMd5::TMd5, TUll>>;
3  const size_t ONE = 1;
```

```
4  const size_t MAX_BLOCK_VAL = ONE << (NMd5::BLOCK_LEN * NMd5::BITS); // Максимально
       возможное число в каждом блоке md5
5
6  void RadixSort(TVector_pair& v){
7      NVector::TVector<TUll> count(MAX_BLOCK_VAL); // Дополнительный вектор
8      TVector_pair secv(v.Size()); // Вектор, в который записываем промежуточные
           результаты сортировки.
9
10     for(int block = NMd5::MD5_BLOCKS - 1; block >= 0; --block)
11     {
12         for(size_t i = 0; i < MAX_BLOCK_VAL; ++i){
13             count[i] = 0;
14         }
15
16         for(size_t i = 0; i < v.Size(); ++i){
17             TUll blockVal = v[i].Key[block];
18             ++count[blockVal];
19         }
20         // Теперь в count[blockVal] хранится количество элементов, где значение блока
               равняется blockVal
21
22         for(size_t i = 1; i < MAX_BLOCK_VAL; ++i){
23             count[i] += count[i-1];
24         }
25         // Теперь в count[i] хранится количество элементов, которые меньше i. (Позиция
               в векторе для элемента, содержащего этот блок)
26
27         for(long long i = v.Size() - 1; i >= 0; --i){
28             TUll blockVal = v[i].Key[block];
29             TUll pos = count[blockVal] - 1;
30             secv[pos] = v[i];
31             --count[blockVal];
32         }
33         // Теперь вектор secv - отсортированный по блоку вектор v и для дальнейшей
               работы необходимо поменять их местами.
34         v.Swap(secv);
35     }
36 }
```

md5.cpp	
unsigned short operator[] (const int i) const	Перегрузка оператора для удобного доступа к блокам MD5-суммы
friend std::istream& operator» (std::istream &in, TMd5& md5)	Дружественная перегрузка оператора для считывания md5-суммы.
bool operator< (const TMd5&)	Перегрузка оператора сравнения для работы сортировки стандартной библиотеки.

std::ostream& operator« (std::ostream &out, const TMd5& md5)	Перегрузка оператора для вывода значения MD5-суммы.
pair.hpp	
std::istream& operator»(std::istream& in, TPair<K, V>& p)	Перегрузка для считывания ключа и значения.
std::ostream& operator«(std::ostream& out, const TPair<K, V>& p)	Перегрузка для вывода ключа и значения через табуляцию.
bool operator< (TPair<K, V> lhs, TPair<K, V> rhs)	Перегрузка оператора для работы сортировки стандартной библиотеки.
vector.hpp	
TVector()	Конструктор вектора.
TVector()	Деструктор вектора.
explicit TVector(TUll cap)	Конструктор для создания вектора с необходимым размером.
void PushBack(T)	Функция, добавляющая элемент в конец вектора.
T PopBack()	Функция, возвращающая значение последнего элемента вектора и удаляющая этот элемент.
void Resize(TUll)	Функция, изменяющая размер вектора.
T& operator[] (TUll)	Перегрузка оператора для удобной работы с элементами вектора.
TVector& operator= (const TVector&)	Перегрузка оператора для копирования вектора.
TUll Size()	Функция, возвращающая размер вектора
T* Begin()	Функция, возвращающая указатель на первый элемент.
T* End()	Функция, возвращающая указатель за последний элемент.
void Swap(TVector<T>&)	Функция, меняющая местами значения векторов без копирования.

Структуры без реализаций методов:

```
1 template <typename K, typename V>
2 struct TPair {
3     K Key;
4     V Value;
5     TPair(){}
6     TPair(K tmpK, V tmpV):
7             Key(tmpK), Value(tmpV){}
```

```
8
9  };
10
11 class TMd5{
12 private:
13     unsigned short Blocks[MD5_BLOCKS];
14 public:
15     unsigned short operator[] (const int i) const;
16     friend std::istream& operator>> (std::istream &in, TMd5& md5);
17     bool operator< (const TMd5&);
18 };
19
20 template <typename T>
21 class TVector
22 {
23 private:
24     T* Vbuf;
25     TUll Vsize;
26     TUll Vcapacity;
27 public:
28     TVector():
29         Vbuf(new T[MIN_CAP]),
30         Vsize(0),
31         Vcapacity(MIN_CAP) {};
32     ~TVector();
33
34     explicit TVector(TUll cap);
35     void PushBack(T);
36     T PopBack();
37     void Resize(TUll);
38     T& operator[] (TUll);
39     TVector& operator= (const TVector&);
40     TUll Size();
41     T* Begin();
42     T* End();
43     void Swap(TVector<T>&);
44 };
```

3 Консоль

```
p.kharkov$ make
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable -c sort.cpp
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable -c md5.cpp
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable md5.o sort.o main.cpp
-o solution
p.kharkov$ cat test1
cb118f1c5002dd7770de064bbabe395f        3380207781261273855
e9fcc36eb8bc39b710d297c065e3dfd1        6081758355532154161
c6d6f48c5aae832d628c00d9c358e603        15589438396786576246
da7fe5f785f2513c5ca1a33152a11bf9        1909228697236021787
069983206a11f199be5fd6be58b7e241        9092327366733038542
p.kharkov$ ./solution <test1
069983206a11f199be5fd6be58b7e241        9092327366733038542
c6d6f48c5aae832d628c00d9c358e603        15589438396786576246
cb118f1c5002dd7770de064bbabe395f        3380207781261273855
da7fe5f785f2513c5ca1a33152a11bf9        1909228697236021787
e9fcc36eb8bc39b710d297c065e3dfd1        6081758355532154161
```

4 Тест производительности

Тест производительности представляет из себя следующее: поразрядная сортировка сравнивается со стабильной сортировкой из стандартной библиотеки, но время на подсчет численного значения в каждом блоке md5-суммы не учитывается. Тест состоит из $2 * 10^7$ образцов.

```
p.kharkov$ make benchmark
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable -c sort.cpp
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable -c md5.cpp
g++ -std=c++14 -pedantic -Wall -Wextra -Wno-unused-variable md5.o sort.o
benchmark.cpp -o benchmark
p.kharkov$ ./benchmark <../tests/2000000.t
Count of lines is 2000000
Radix sort time: 1266ms
STL stable sort time: 1617ms
```

Как видно, метод поразрядной сортировки выиграл у стабильной сортировки из стандартной библиотеки, из-за того, что сложность поразрядной сортировки - $O(n)$, а сложность стабильной сортировки - $O(nlog(n))$. Так как сложность моей поразрядной сортировки равна $O(2^3 * (n + 2^{16}))$, то уже при $n \sim 2^{20}$ поразрядная сортировка начинает выигрывать по скорости у быстрой сортировки.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился писать свой вектор, используя шаблоны и саму сортировку. Я понял, что при разных входных данных, лучше использовать разные виды сортировок. К примеру, мне кажется, использовать поразрядную сортировку на MD5-суммах выгодно при большом количестве входных данных, так как при маленьких, быстрая сортировка работает быстрее. Также я научился пользоваться valgrind для выявления утечек памяти и некоторых других возможных ошибках. Это очень полезная утилита, которая необходима, мне кажется, для проверки любой программы на работоспособность. Еще я узнал, что в конструкторе класса нельзя использовать указатель *this*, так как он ещё не определен, что вызывало большое количество непонятных для меня ошибок. А также научился писать по иному, непривычному для меня, стилю, так как я всегда старался писать в одном и том же. Мне кажется, что такая «смена обстановки» пригодится ещё не раз, так как в разных компаниях разные стили, это поможет мне быстрее привыкнуть к новому.

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] Страуструп, Бьярне. *Язык программирования C++. Краткий курс, 2-е изд.* — Пер. с англ. - СПб.: ООО "Диалектика 2019. - 320 с.: ил. - Парал. тит. англ.