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. Т. Бахарев
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: `MD5-суммы (32-разрядные шестнадцатиричные числа).`

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

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки.

Идея сортировки заключается в разбиении сортируемых элементов на разряды. Затем выполняется сортировка для каждого разряда. Алгоритм для сортировки разрядов может быть выбран любым, но для максимальной эффективности нужно использовать сортировки за линейное время. При этом, для строк подходит версия MSD(Most Significant Digit) - сортировка начинается от самого старшего разряда. Для чисел же нужно использовать LSD-версию(Least Significant Digit). Для выполнения данной лабораторной работы была выбрана устойчивая сортировка подсчетом. В данной версии, сортировка начинается от самого младшего разряда.

Свойства сортировки подсчетом:

– Не является сортировкой сравнением: ни одна пара элементов не сравнивается друг с другом

– Линейная (вернее, $O(k+n)$, но при $k = O(n)$ время выполнения $O(n)$)

– Устойчивая (стабильная)

– Не используются обмены(swap)

– Требует дополнительную память под массивы C и B размером k и n соответственно

Теорема 1 (О времени работы программы)

Для n b-битовых чисел и натурального числа $r \leq b$ (цифры из r битов) алгоритм Radix-Sort выполнит сортировку за время $\Theta(\frac{b}{r}(n+2^r))$

Тем самым, для $b < [log(n)]$, то асимптотически оптимален перебор $r = b$, иначе $r = [log(n)]$

r – количество разрядов в самом длинном ключе. b – разрядность : количество значений разряда ключа

Описание работы программы

Сначала происходит ввод данных. Структура $TElement$ имеет 2 поля - для ключа и значения. Так как заранее неизвестно, сколько элементов будет обрабатываться, то исходный массив динамически расширяется. Как только встречается символ EOF(все данные введены), ввод считается завершенным и массив передается функции *sort*, которая сортирует данные. Так же, в функцию передается количество введеных элементов. Затем происходит печать того же массива, но уже отсортированного. Функция *sort* работает следующим образом: Нам заранее известен алфавит входных данных – 16-ричные числа. Известна и длина каждой строки - 32. Создадим временный массив A с типом $TElement$ и таким же размером как и исходный массив данных. Начинаем сортировку с конца строк. На каждом шагу(а всего их 32) заполняем массив C из 16 элементов. Так мы узнаем сколько раз какой элемент алфавита был

встречен. Далее начинаем использовать массив A, который является как-бы "мгновенным снимком"исходного массива на данной итерации. Он нужен чтобы никакие данные при перемещении элементов не пропали. Благодаря массиву C, мы знаем какой элемент должен находится на определенном месте. Осталось только перезаписать данные в исходном массиве и начать следующую итерацию.

2 Исходный код

1. *main.cpp*

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
typedef struct TElement TElement;
struct TElement
{
    char Buffer[33];
    unsigned long long int n;
};
void sort(TElement*, int& amount_of_elems);
int main(int argc, char *argv[])
{
    int size_of_array = 1; // default value
    int amount_of_elems = 0;
    TElement* array = new TElement[size_of_array]; // array of elems
    while(1)
    {
        char ch = getchar();
        if(ch == EOF || ch == '\0')
            break;
        else
            ungetc(ch, stdin);
        if(amount_of_elems == size_of_array)
        {
            TElement* tmp_buffer = new TElement[size_of_array];
            memcpy(tmp_buffer, array, size_of_array*sizeof(TElement));
            array = new TElement[size_of_array * 2];
            memcpy(array, tmp_buffer, size_of_array*sizeof(TElement));
            size_of_array *= 2;
            delete[] tmp_buffer;
        }
        scanf("%s", array[amount_of_elems].Buffer);
        scanf("%llu", &array[amount_of_elems].n);
        ++amount_of_elems;
        getchar();
    }
    sort(array, amount_of_elems);
    for(int i = 0; i < amount_of_elems; ++i)
    {
        printf("%s\t", array[i].Buffer);
        printf("%llu\n", array[i].n);
    }
    delete[] array;
    return 0;
}
```

```
47
48 void sort(TElement* array, int& amount_of_elems)
49 {
50     const int radix = 16;
51     const int strLen = 32;
52     TElement* A = new TElement [amount_of_elems];
53     for(int digit = strLen - 1; digit >= 0; --digit)
54     {
55         int C[radix] = {0};
56         for(int i = 0; i < amount_of_elems; ++i)
57         {
58             A[i] = array[i];
59             int c = array[i].Buffer[digit];
60             if (c >= '0' && c <= '9') c = c - '0';
61             else c = c - 'a' + 10; // According to ASCII table
62             ++C[c];
63         }
64         for(int i = 1; i < radix; ++i)
65             C[i] = C[i] + C[i - 1];
66         for(int i = amount_of_elems - 1; i >= 0; --i)
67         {
68             TElement val = A[i];
69             int c = A[i].Buffer[digit];
70             if (c >= '0' && c <= '9') c = c - '0';
71             else c = c - 'a' + 10; // According to ASCII table
72             int position = C[c] - 1;
73             array[position] = val;
74             --C[c];
75         }
76     }
77     delete[] A;
78 }
```

2. *testgenerator.py*

```
1  import random
2  import string
3
4  MAX_KEY_LEN = 33
5  MAX_VALUE_LEN = 8
6
7  def generate_random_key():
8      return "".join( [ random.choice( string.hexdigits.lower() ) for _ in range( 1,
           MAX_KEY_LEN ) ] )
9
10 def generate_random_value():
11     return "".join( [ random.choice( string.digits ) for _ in range( 1, MAX_VALUE_LEN )
           ] )
12
13 if __name__ == "__main__":
```

```
14      for num in range(1, 2):
15          values = list()
16          output_filename = "tests/{:02d}.t".format( num )
17          with open( output_filename, 'w') as output:
18              for _ in range( 10000000 ):
19                  key = generate_random_key()
20                  value = generate_random_value()
21                  values.append( (key, value) )
22                  output.write( "{}\t{}\n".format(key, value) )
23          output_filename = "tests/{:02d}.a".format( num )
24          with open( output_filename, 'w') as output:
25              values = sorted( values, key=lambda x: x[0] )
26              for value in values:
27                  output.write( "{}\t{}\n".format(value[0], value[1]) )
```

3 Консоль

```
alex$ g++ -std=c++11 -o da_1 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm main.cpp sort.cpp
alex$ cat 01.t
00000000000000000000000000000000 13207862122685464576
ffffffffffffffffffffffffffffffff 7670388314707853312
00000000000000000000000000000000 4588010303972900864
ffffffffffffffffffffffffffffffff 12992997081104908288
alex$ ./da_1 <01.t
00000000000000000000000000000000 13207862122685464576
00000000000000000000000000000000 4588010303972900864
ffffffffffffffffffffffffffffffff 7670388314707853312
ffffffffffffffffffffffffffffffff 12992997081104908288
```

4 Тест производительности

Тест производительности представляет из себя следующее: Производится замер времени для работы поразрядной сортировки и сравнивается со временем работы сортировки std::sort().

Тесты производятся на основании сгенерированых текстовых файлов при помощи скрипта на Python. В каждом файле имеется от 1 до 10^{17} входных элементов.

```
alex$ bash benchmark.sh
g++ -std=c++11 -o da_1 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-long
-lm main.cpp sort.cpp
Time for Radix Sort

real 0m20.225s
user 0m19.748s
sys 0m0.472s
Time for std::sort()

real 0m28.851s
user 0m27.836s
sys 0m1.000s
alex$
```

Ниже представлен листинг теста производительности: *benchmark.cpp*

```
1  #include <iostream>
2  #include <cstdio>
3  #include <cstdlib>
4  #include <cstring>
5  #include <vector>
6  #include <algorithm>
7  class TElement
8  {
9  public:
10     TElement(){}
11     ~TElement(){}
12     friend std::istream& operator>>(std::istream&, const TElement&);
13     friend std::ostream& operator<<(std::ostream&, const TElement&);
14     bool operator >(const TElement&);
15     bool operator <(const TElement&);
16     bool operator ==(const TElement&);
17     unsigned long long int* GetN();
18     char* GetBuff();
19 private:
20     char Buffer[33];
```

```
21      unsigned long long int N;
22  };
23  bool TElement::operator<(const TElement& n1)
24  {
25      if(strncmp(n1.Buffer, this->Buffer, 33) < 0) return true;
26      else return false;
27  }
28  bool TElement::operator>(const TElement& n1)
29  {
30      if(strncmp(n1.Buffer, this->Buffer, 33) > 0) return true;
31      else return false;
32  }
33  bool TElement::operator==(const TElement& n1)
34  {
35      if(strncmp(n1.Buffer, this->Buffer, 33) == 0) return true;
36      else return false;
37  }
38  std::istream& operator>>(std::istream& input, TElement& n)
39  {
40      char* Buff = n.GetBuff();
41      input>>Buff;
42      input.get();
43      auto N = n.GetN();
44      input>>*N;
45      //input.get(); Commented for \0 recognition
46      return input;
47  }
48  std::ostream& operator<<(std::ostream& output, const TElement& n)
49  {
50      output<<n.Buffer;
51      output<<"\t";
52      output<<n.N;
53      return output;
54  }
55  unsigned long long int* TElement::GetN()
56  {
57      return &N;
58  }
59  char* TElement::GetBuff()
60  {
61      return Buffer;
62  }
63  int main()
64  {
65      std::vector<TElement> array;
66      while(!std::cin.eof())
67      {
68          //std::cin.unget();
69          TElement tmp;
```

```
70          std::cin>>tmp;
71          array.push_back(tmp);
72          // std::cout<<"OK\n";
73      }
74      std::sort(array.begin(), array.end());
75      for(int i = 0; i < array.size() - 1; ++i)
76      {
77          std::cout<<array[i]<<std::endl;
78      }
79  }
```

Как видно, поразрядная сортировка выиграла по времени у быстрой сортировки(std::sort()). Это связано с тем, что сложность первой сортировки - линейная, а у std::sort() она равна $O(NlogN)$. На небольших данных разница во времени не ощущается, но при работе с внушительными объемами, поразрядная сортировка значительно лучше и производительнее. Стоит заметить, что если заменить сортировку подсчетом(в нашем случае эта сортировка являеется внутренней для поразрядной) на другую, которая будет работать ассимптотически дольше, то и вся сортировка в целом будет работать дольше.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать и применять на практике эффективные сортировки за линейное время. Те алгоритмы, которые я знал до этого, значительно проигрывают по времени и по памяти. Хорошо и то, что сортировки за линейное время достаточно просты в реализации. Увидеть преимущество данных алгоритмов можно только при больших обьемах данных. Для повседневной жизни вполне может сойти и быстрая сортировка. А для особенных случаев, когда входные данные слишком большие, уместно использовать сортировки за линейное время.

Также, я научился писать бенчмарки для оценки времени работы программы и определять сложность работы алгоритмов, а это, на мой взгяд, очень ценные знания.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2017).