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. А. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата: 05.10.2020
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: Сортировка подсчётом.

Вариант ключа: `почтовые индексы`.

Вариант значения: `строки переменной длины (до 2048 символов)`.

1 Описание

В задаче нужно реализовать стабильный вариант сортировки подсчётом. Идея заключается в том, чтобы для каждого элемента массива x определить сколько чисел меньше x встречаются в исходном массиве и расставить по этим значениям x в отсортированном порядке. Для этого сначала посчитаем число вхождений каждого ключа в массив $count$, затем вычислим префиксные суммы этого массива. Получим, что $count_i$ – число элементов, ключ которых меньше либо равен i. Проходя по исходному массиву с конца, будем ставить x на место $count[x.key] - 1$ и уменьшать $count[x.key]$, чтобы поставить следующий элемент с таким же ключом левее нашего x.

Стоит также отметить, что элемент x представлен в программе в виде структуры со значениями key – ключ элемента x и val – значение элемента x, соответствующий ключу key.

2 Исходный код

Для хранения данных типа "ключ-значение"нужно создать вектор. Так как по заданию нам запрещено пользоваться библиотеками **STL**, создадим свой шаблонный вектор $TVector$. Сначала его обернем в наш *namespace*, чтобы случайно не использовать одинаковые обозначения. Реализуем в нем стандартные методы вектора: $Assign()$, $PushBack()$, $GetSize()$, $Empty()$. Также для удобства получения данных и их изменения перегрузим оператор [], сделав в нем защиту выхода за пределы массива. В случае этой ошибки, программа выведет предупреждение и выведет последний элемент вектора.

Также для удобства создания моего вектора, я сделал различные конструкторы, которые могут выделить необходимое количество памяти, и, если надо, заполнить нужными значениями.

Можно заметить, что для определения размерности вектора, я использую 2 переменные. Так как выделение новой памяти занимает продолжительное время, я пытаюсь минимизировать частоту отправлений запросов, на выделение памяти. Поэтому выделение памяти у меня происходит путем увеличения действующей памяти в 2 раза. За это отвечает переменная *capasity*. *size* в свою очередь показывает количество заполненных данных в векторе.

Теперь перейдем в исполнительный файл *main.cpp*.

На каждой непустой строке входного файла располагается пара "ключ-значение поэтому создадим новую структуру $TPair$, в которой будем хранить ключ в int и значение в $char[LEN]$. Значение хранится так, потому что на чеккере возникала неизвестная мне ошибка. Перепробовав все способы решения проблемы, тип значения поменялся, как следствие. Так как на память данное изменение не влияет очень сильно, я решил оставить как есть.

Далее идет реализация $CountingSort$. Создается 2 массива. *count* (его смысл был описан выше) и *result*, куда записывается результат сортировки.

Ниже есть функция, обернутая в оболочку предпроцесса, сделанная исключительно в отладочных целях. И, наконец, дальше идет функция *main*, в которой написаны ввод, вызов сортировки и вывод.

```
1  #include "TVector.h"
2  //#define DEBUG
3  //#define TEST_ENTER
4  const int LEN = 2049;
5
6  struct TPair{
7      int key;
8      char val[LEN];
9      TPair(){
10         key = 0;
```

```
11      }
12      TPair(int n, char elem[LEN]){
13          strcpy(val,elem);
14          key = n;
15      }
16      ~TPair(){}
17  };
18
19  void CountingSort(NVector::TVector<TPair> & data, const unsigned int & maxNum){
20      NVector::TVector<int> count(maxNum + 1);
21      NVector::TVector<TPair> result(data.GetSize());
22
23          count.Assign(0);
24          for (int j = 0; j < data.GetSize(); ++j) {
25              count[data[j].key]++;
26          }
27
28          for (int j = 1; j <= maxNum; ++j) {
29              count[j] = count[j] + count[j - 1];
30          }
31
32          for(int j = data.GetSize() - 1; j >= 0; --j){
33              result[count[data[j].key] - 1] = data[j];
34              count[data[j].key]--;
35          }
36
37          for (int j = 0; j < data.GetSize(); ++j) {
38              data[j] = result[j];
39          }
40  }
41
42  #ifdef TEST_ENTER
43  void EnterKey(const TPair &data){
44      for (int i = 0; i < 6; ++i) {
45          std::cout << data.key[i];
46      }
47  }
48  #endif // TEST_ENTER
49
50  int main() {
51      NVector::TVector<TPair> values;
52
53      int key;
54      char val[LEN];
55      int maxNum = -1;
56      while(scanf("%d %s", &key, val) > 0){
57          TPair initialVal;
58          initialVal.key = key;
59          strcpy(initialVal.val,val);
```

```
        values.PushBack(initialVal);

        maxNum = std::max(maxNum, key);
    }
#ifdef DEBUG
    for (int i = 0; i < values.GetSize(); ++i) {
        for (auto j : values[i].key){
            std::cout << j;
        }
        std::cout << std::endl;
    }
#endif // DEBUG
    if(!values.Empty()){
        CountingSort(values, maxNum);
        for (int i = 0; i < values.GetSize(); ++i) {
            if(values[i].key == 0){
                printf("00000");
            }
            else{
                for (int j = 0; j < 6 - 1 - log10(values[i].key); ++j) {
                    printf("0");
                }
            }
            printf("%d %s\n", values[i].key, values[i].val);
        }
    }
    return 0;
}
```

```
//TVector.h

#ifndef DA_LAB1_T_VECTOR_H
#define DA_LAB1_T_VECTOR_H
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <algorithm>
#include <iostream>

namespace NVector {
    template<class T>
    class TVector {
    private:
        unsigned int size;
        unsigned int capacity;
        T *data;

    public:
        void Assign(const T elem){
            for (unsigned int i = 0; i < size; ++i){
```

```
22              data[i] = elem;
23          }
24      }
25
26      TVector() {
27          size = 0;
28          capacity = 1;
29      };
30
31      TVector(const unsigned int n) /*: TVector()*/ {
32          size = n;
33          capacity = n;
34          data = new T[capacity];
35      };
36
37      TVector(const unsigned int n, T elem)/*: TVector()*/{
38          size = n;
39          capacity = n;
40          data = new T[capacity];
41          Assign(elem);
42      }
43
44      unsigned int GetSize(){
45          return size;
46      }
47
48      void PushBack(const T &elem){
49          if(capacity == 1){
50              capacity *= 2;
51              data = new T[capacity];
52              data[size] = elem;
53              size++;
54          }
55
56          else if(capacity <= size){
57              capacity *= 2;
58              T* newData = new T[capacity];
59              for (int i = 0; i < size; ++i) {
60                  newData[i] = data[i];
61              }
62              delete[] data;
63              data = newData;
64              data[size] = elem;
65              size++;
66          }
67
68          else{
69              data[size] = elem;
70              size++;
```

```
71              }
72          }
73
74          bool Empty(){
75              return !size;
76          }
77
78          T &operator[](const unsigned int &iterator){
79              if(iterator >= size) {
80                  std::cout << "CAUTION: size was reached\niterator is " << iterator <<
                        std::endl;
81                  return data[size - 1];
82              }
83              return data[iterator];
84          }
85
86          ~TVector(){
87              if(!this->Empty()) {
88                  delete[] data;
89              }
90          }
91      };
92  }
93  #endif //DA_LAB1_T_VECTOR_H
```

3 Консоль

```
whitewolf@WhitewolfsPC:/mnt/d/Projects/c++/DA-labs/DA lab1$ cat test.txt
923400 end
000000 begin
923399 middle1
923399 middle2
123456 middle3
whitewolf@WhitewolfsPC:/mnt/d/Projects/c++/DA-labs/DA lab1$ cat test.txt |
./a.out
000000 begin
123456 middle3
923399 middle1
923399 middle2
923400 end
```

4 Тест производительности

Тест производительности представляет из себя следующее: выполняются 2 типа сортировок. Первый – сортировка подсчетом. Второй – стабильная сортировка. В тесте находится 10^6 различных сгенерированных данных. Ввод данных при учете времени не учитывается.

```
my Counting sort 15.625

stable sort  31.25
```

Из теста видно, что сортировка подсчетом работает быстрее почти в 2 раза. Это связано с тем, что сортировка подсчетом работает за $O(n)$, а стабильная за $O(n * log(n))$

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился создавать свой собственный вектор, используя шаблоны классов; научился реализовывать новую для меня сортировку подсчетом; научился писать генераторы тестов и бенчмарки для тестирования времени работы той или иной функции.

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: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008