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. Инютин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

Дата:
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: `Числа от 0 до 65535.`

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

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 вхождение каждого ключа в массив $count$, а 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.

2 Исходный код

Входные данные представляют собой пару «ключ-значение», поэтому создадим структуру *TItem* для хранения пары. Так как размер входных данных (количество пар) не задано, то пары будем хранить в векторе, добавляя по одной паре в конец контейнера. Затем применим сортировку, реализуя алгоритм, описанный выше.

Первый этап написания кода - ввод и вывод данных. Для этого нужно реализовать свой вектор. Чтобы не делать основной файл кода сильно большим, вынесем вектор в отдельный файл «vector.hpp». Теперь опишем структуру *TItem* и создадим вектор пар.

Второй этап - сама сортировка. Функция *CountingSort* будет принимать на вход вектор пар и размер сортируемого блока. Результат сортировки будет сохранён в исходном векторе.

Наконец, написание генератора тестов, отладка, тестирование и бенчмарк программы. Вводить тест 10^6 довольно сложно, поэтому разумнее перенаправить ввод программы в файл, в который мы заранее напишем сгенерированный тест.

```
1  /*
2   * main.cpp
3   */
4
5  #include "stdio.h"
6  #include "vector.hpp"
7
8  struct TItem {
9    unsigned short Key;
10   unsigned long long Value;
11 };
12
13 void CountingSort(NMystd::TVector<TItem> &data, int n) {
14   int maxKey = 0;
15   for (int i = 0; i < n; ++i) {
16     if (data[i].Key > maxKey) {
17       maxKey = data[i].Key;
18     }
19   }
20   NMystd::TVector<int> count(maxKey + 1);
21   for (int i = 0; i < n; ++i) {
22     count[data[i].Key]++;
23   }
24
25   for (int i = 1; i < maxKey + 1; ++i) {
26     count[i] = count[i] + count[i - 1];
27   }
28   NMystd::TVector<TItem> res(n);
29   for (int i = n - 1; i >= 0; --i) {
```

```
30      res[count[data[i].Key] - 1] = data[i];
31      count[data[i].Key]--;
32    }
33    for (int i = 0; i < n; ++i) {
34      data[i] = res[i];
35    }
36  }
37
38  signed main() {
39    //freopen("input.txt", "r", stdin);
40    //freopen("output.txt", "w", stdout);
41
42    NMystd::TVector<TItem> a;
43    TItem cur;
44    while (scanf("%hu%llu", &cur.Key, &cur.Value) > 0) {
45      a.PushBack(cur);
46    }
47    int n = a.Size();
48    CountingSort(a, n);
49    for (int i = 0; i < n; ++i) {
50      printf("%hu %llu\n", a[i].Key, a[i].Value);
51    }
52    return 0;
53  }
```

```
/*
 * vector.hpp
 */

#ifndef VECTOR_HPP
#define VECTOR_HPP

namespace NMystd {
  template<class T>
  class TVector {
    private:
      unsigned int Capacity;
      unsigned int MaxSize;
      T* Data;
    public:
      void Assert(const unsigned int &n, T elem);
      void PushBack(T elem);
      unsigned int Size();
      TVector();
      TVector(const unsigned int &n);
      TVector(const unsigned int &n, T elem);
      ~TVector();
      T& operator[](const unsigned int &index);
  };

  template<class T>
  void TVector<T>::Assert(const unsigned int &n, T elem) {
    for (int i = 0; i < n; ++i) {
      Data[i] = elem;
    }
  }

  template<class T>
  void TVector<T>::PushBack(T elem) {
    if (Data == 0) {
      MaxSize = 1;
      Data = new T[MaxSize];
    } else if (Capacity == MaxSize) {
      MaxSize = MaxSize * 2;
      T* newData = new T[MaxSize];
      for (int i = 0; i < Capacity; ++i) {
        newData[i] = Data[i];
      }
      delete[] Data;
      Data = newData;
    }
    Data[Capacity] = elem;
    Capacity++;
  };
```

```
50
51    template<class T>
52    unsigned int TVector<T>::Size() {
53      return Capacity;
54    }
55
56    template<class T>
57    TVector<T>::TVector() {
58      Capacity = 0;
59      MaxSize = 0;
60      Data = 0;
61    }
62
63    template<class T>
64    TVector<T>::TVector(const unsigned int &n) {
65      Capacity = n;
66      MaxSize = n;
67      Data = new T[Capacity];
68      Assert(n, T());
69    }
70
71    template<class T>
72    TVector<T>::TVector(const unsigned int &n, T elem) {
73      Capacity = n;
74      MaxSize = n;
75      Data = new T[Capacity];
76      Assert(n, elem);
77    }
78
79    template<class T>
80    TVector<T>::~TVector() {
81      delete[] Data;
82    }
83
84    template<class T>
85    T& TVector<T>::operator[](const unsigned int &index) {
86      return Data[index];
87    }
88  }
89
90  #endif // VECTOR_HPP
```

3 Консоль

```
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ cat input.txt
5 9
5 10
1 1
1 2
2 4
3 6
4 8
1 3
2 5
3 7
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ cat input.txt | ./solution
1 1
1 2
1 3
2 4
2 5
3 6
3 7
4 8
5 9
5 10
```

4 Тест производительности

Чтобы провести тест производительности сравним время выполнения нашей сортироваки подсчётом и готовой сортировки *std::stable_sort* . Первый тест состоит из 10^4 пар «ключ-значение», второй из 10^5 и третий из 10^6 строк.

```
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ make benchmark
g++ -std=c++14 -Werror benchmark.cpp -o benchmark
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ cat input1.txt | ./benchmark
CountingSort 1.621 ms
std::stable_sort 4.870 ms
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ cat input2.txt | ./benchmark
CountingSort 6.648 ms
std::stable_sort 64.408 ms
engineerxl@engineerxl-VirtualBox:~/DA/lab1$ cat input3.txt | ./benchmark
CountingSort 57.126 ms
std::stable_sort 753.869 ms
```

Видно, что сортировка подсчётом быстрее в 3, 10, 13 раз, так как сложность подсчёта $O(n)$, а сложность готовой сортировки $O(n * \log n)$

5 Выводы

В ходе выполнения лабораторной работы по курсу «Дискретный анализ», я научился работать с шаблонными классами в языке *C++* на примере своей реализации структуры данных *std::vector*, узнал новые алгоритмы линейной сортировки и их стабильные версии, реализовал сортировку подсчётом согласно своему варианту и поразрядную сортировку для себя. В ходе отладки выяснил, что *iostream* работает медленнее, чем *stdio.h*, что сильно сказывалось на общем времени работы программы во время тестирования.

Список литературы

[1] Кормен Т., Лейзерсон Ч., Ривест Р. *Алгоритмы: построение и анализ. — 2-е изд.* — М.: Издательский дом «Вильямс», 2007. — С. 224—226.

[2] *Сортировка подсчётом — Викиконспекты.*
URL: `http://www.neerc.ifmo.ru/wiki/index.php?title=Сортировка_подсчётом`
(дата обращения: 24.09.2020).