Московский авиационный институт (национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Дискретный анализ»

Студент: Л. В. Короткевич
Преподаватель: Н. С. Капралов
Группа: М8О-208Б

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: Поразрядная сортировка.

Вариант ключа: Даты в формате DD.MM.YYYY, например 1.1.1, 1.9.2009, 01.09.2009, 31.12.2009.

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

1 Описание

Для выполнения поставленной задачи необходимо реализовать поразрядную сортировку. Множеством объектов для сортировки являются пары «ключ-значение», для которых в последующем будет описана структура Item. Хранить их удобней всего будет в векторе.

Ввод и вывод осуществляется с помощью функций стандартного ввода/вывода scanf/printf. Почему не cin/cout? Во-первых, так удобней разбивать входные строки на отдельные числа: день, месяц, год. Во-вторых, использование cin/cout приведет к превышению ограничения по времени исполнения программы.

При сортировке сравнение объектов будет производиться поразрядно: сначала сравнивается разряд дней нашей строки-ключа, потом месяцев, после - годов. На каждом этапе элементы группируются относительно результатов сравнения предыдущего этапа. Такой метод поразрядной сортировки, «от младшего к старшему», называется LSD (Least Significant Digit) Radix Sort.

В данной разновидности не используются сравнения и обмены, алгоритм линеен. Вычислительная сложность - O(n). Основной особенностью алгоритма является высокая эффективность для сильно перемешанных или случайных наборов данных. На почти отсортированных наборах имеет смысл применять другие алгоритмы, так как выигрыш будет не так значителен. Плохо работает на малых массивах размером с пару сотен элементов.

2 Исходный код

Как было сказано выше, для хранения объектов будет использоваться вектор. Его реализацию я описал в отдельном файле **Vector.hpp**:

```
1  #pragma once
2
3  template <class T>
4  class TVector
5  {
6  public:
7    TVector();
8    TVector(const TVector &v);
9    TVector(const size_t n, const T &val = T());
10   ~TVector();
11
12   void PushBack(const T &val);
13   void Erase(const size_t index);
14   void Resize(const size_t n, const T &val = T());
15   void Clear();
16   size_t Size() const;
17   bool Empty() const;
18
19   T &operator[](const size_t i);
20   TVector &operator=(const TVector &v);
21
22 private:
23   T *begin;
24   size_t size;
25   size_t cap;
26
27   void Copy(const TVector &v);
28 };
29
30 template <class T>
31 TVector<T>::TVector()
32 {
33   begin = new T[1];
34   size = 0;
35   cap = 1;
36 }
37
38 template <class T>
39 TVector<T>::TVector(const TVector &v)
40 {
41   begin = new T[1];
42   size = 0;
43   cap = 1;
44
```

```
45    if (this != &v)
46      Copy(v);
47  }
48
49  template <class T>
50  TVector<T>::TVector(const size_t n, const T &val)
51  {
52    begin = new T[n + 1];
53
54    for (size_t i = 0; i < n; ++i)
55      begin[i] = val;
56
57    size = n;
58    cap = n + 1;
59  }
60
61  template <class T>
62  TVector<T>::~TVector()
63  {
64    delete[] begin;
65  }
66
67  template <class T>
68  void TVector<T>::PushBack(const T &val)
69  {
70    if (size == cap)
71    {
72      size_t oldSize = size;
73
74      Resize(size * 2);
75
76      size = oldSize;
77    }
78
79    begin[size++] = val;
80  }
81
82  template <class T>
83  void TVector<T>::Erase(const size_t index)
84  {
85    for (size_t i = index; i < size - 1; ++i)
86      begin[i] = begin[i + 1];
87
88    size--;
89  }
90
91  template <class T>
92  void TVector<T>::Resize(const size_t n, const T &val)
93  {
```

```
94    const size_t copySize = n < size ? n : size;
95    T *_buffer = new T[n + 1];
96
97    for (size_t i = 0; i < copySize; ++i)
98      _buffer[i] = begin[i];
99
100   for (size_t i = copySize; i < n; ++i)
101     _buffer[i] = val;
102
103   delete[] begin;
104
105   begin = _buffer;
106   size = n;
107   cap = n + 1;
108 }
109
110 template <class T>
111 void TVector<T>::Clear()
112 {
113   delete[] begin;
114
115   begin = new T[1];
116   size = 0;
117   cap = 1;
118 }
119
120 template <class T>
121 size_t TVector<T>::Size() const
122 {
123   return size;
124 }
125
126 template <class T>
127 bool TVector<T>::Empty() const
128 {
129   return size == 0;
130 }
131
132 template <class T>
133 T &TVector<T>::operator[](const size_t i)
134 {
135   return begin[i];
136 }
137
138 template <class T>
139 TVector<T> &TVector<T>::operator=(const TVector &v)
140 {
141   if (this != &v)
142     Copy(v);
```

```
143
144   return *this;
145 }
146
147 template <class T>
148 void TVector<T>::Copy(const TVector &v)
149 {
150   const size_t n = v.Size();
151
152   delete[] begin;
153
154   begin = new T[n + 1];
155
156   for (size_t i = 0; i < n; ++i)
157     begin[i] = v.begin[i];
158
159   size = n;
160   cap = n + 1;
161 }
```

Также была описана структура для работы с объектами:

Item.hpp:

```
1  #pragma once
2
3  struct TKey
4  {
5      int date[3] = {-1}; // days, months, years separately
6      char str[11] = {'\0'}; // DD.MM.YYYY
7  };
8
9  struct TItem
10 {
11     TKey key; // key
12     unsigned long long val = 0; // value
13
14     TItem() {}
15     TItem(const TKey &k, const unsigned long long &v)
16     {
17         val = v;
18         memcpy(key.date, k.date, 3 * sizeof(int));
19         strcpy(key.str, k.str);
20     }
21 };
```

Наконец, файл **main.cpp**, в котором я описал инициализацию, считывание и сортировку объектов, а также их последующий вывод:

```cpp
#include <iostream>
#include <cstdio>
#include <string.h>

#include "Vector.hpp"
#include "Item.hpp"

int main()
{
    TVector<TItem> vec;
    TKey k;
    unsigned long long v;

    int sup[3] = {0}; // max value of day, month and year got in input
    while (scanf("%s %llu", k.str, &v) > 0)
    {
        sscanf(k.str, "%d.%d.%d", &k.date[0], &k.date[1], &k.date[2]); // parsing key (string) into date array
        vec.PushBack(TItem(k, v));
        sup[0] = std::max(sup[0], k.date[0]);
        sup[1] = std::max(sup[1], k.date[1]);
        sup[2] = std::max(sup[2], k.date[2]);
    }

    int n = vec.Size();
    TVector<TItem> res(n); // gonna be sorted vector

    for (int R = 0; R < 3; ++R) // R = 0 means counting sort by days, 1 - by months, 2 - by years
    {
        // cnt: number of occurrences of days or months or years (depends on R)
        TVector<int> cnt(sup[R] + 1, 0);
        for (int i = 0; i < n; ++i)
        {
            cnt[vec[i].key.date[R]]++;
        }
        // counting prefix-sums of cnt
        for (int i = 1; i < sup[R] + 1; ++i)
        {
            cnt[i] += cnt[i - 1];
        }
        // getting result sorted by days or months or years (depends on R)
        for (int i = n - 1; i >= 0; i--)
        {
            res[cnt[vec[i].key.date[R]] - 1] = vec[i];
            --cnt[vec[i].key.date[R]];
        }
```

```
46          for (int i = 0; i < n; i++)
47          {
48              vec[i] = res[i];
49          }
50      }
51
52      for (int i = 0; i < n; ++i)
53      {
54          printf("%s\t%llu\n", vec[i].key.str, vec[i].val);
55      }
56
57      return 0;
58  }
```

3 Консоль

```
[leo@pc solution]$ make
g++    -c -o main.o main.cpp
g++ -std=c++14 -O2 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic main.o -o solution
[leo@pc solution]$ cat test01
30.3.6071       450587263
4.12.6809       894803200
4.3.8884        271119703
3.4.5432        1480599065
2.9.7465        1458018124
4.1.9912        2017513080
30.10.6043      1857878275
21.11.4279      206941075
4.4.7337        1252396845
25.3.8044       1101879030
[leo@pc solution]$ ./solution <test01
21.11.4279      206941075
3.4.5432        1480599065
30.10.6043      1857878275
30.3.6071       450587263
4.12.6809       894803200
4.4.7337        1252396845
2.9.7465        1458018124
25.3.8044       1101879030
4.3.8884        271119703
4.1.9912        2017513080
```

```
[leo@pc solution]$ cat test02
1.8.4742        1615060245
19.11.2483      1172615916
30.12.8292      484413915
2.12.2543       1338332487
11.7.3382       1316124383
[leo@pc solution]$ ./solution <test02
19.11.2483      1172615916
2.12.2543       1338332487
11.7.3382       1316124383
1.8.4742        1615060245
30.12.8292      484413915
[leo@pc solution]$ cat test03
9.2.2815        1685520877
24.3.8654       743133012
10.6.8117       773777756
2.2.2413        499234118
13.11.3122      507741609
17.7.8896       1412683711
16.12.8219      1578901332
15.8.1228       258756370
12.9.8245       9492636
22.12.1647      1708693402
[leo@pc solution]$ ./solution <test03
15.8.1228       258756370
22.12.1647      1708693402
2.2.2413        499234118
9.2.2815        1685520877
13.11.3122      507741609
10.6.8117       773777756
16.12.8219      1578901332
12.9.8245       9492636
24.3.8654       743133012
17.7.8896       1412683711
```

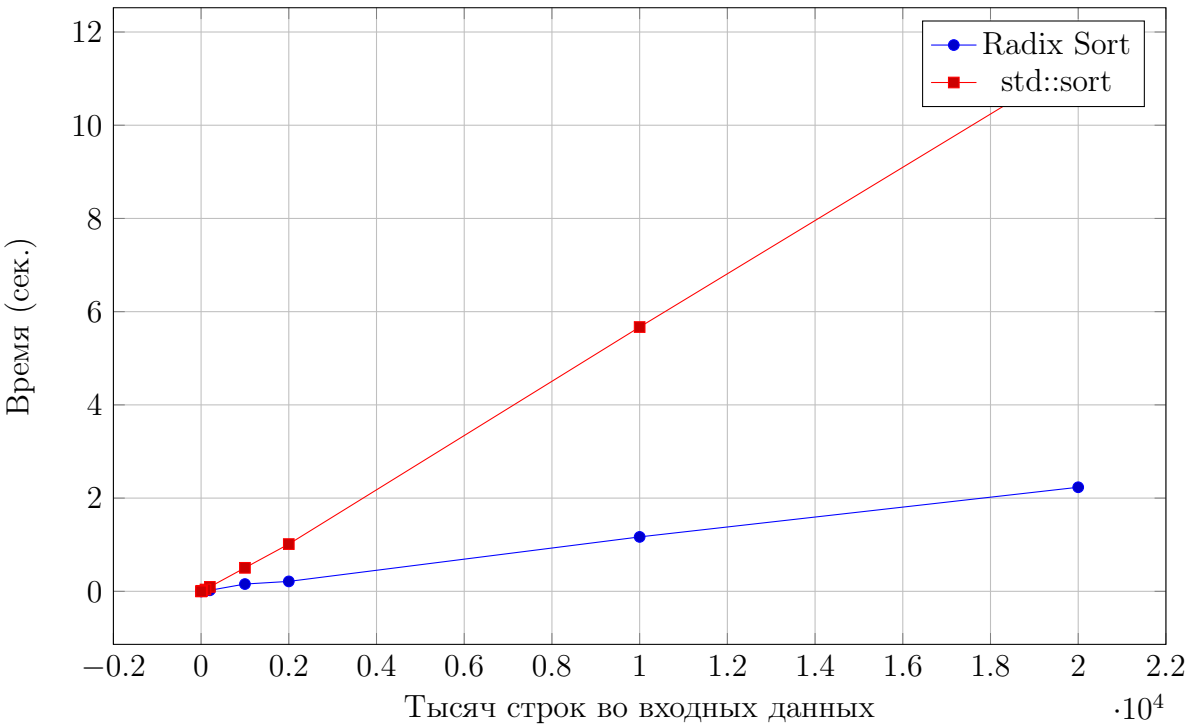
4 Тест производительности

Программа для геренации наборов входных данных:

```
1  #include <bits/stdc++.h>
2
3  #define ull unsigned long long
4
5  using namespace std;
6
7  ull rand(ull a, ull b)
8  {
9      return a + rand() % (b - a + 1);
10 }
11
12 int main(int argc, char* argv[])
13 {
14     srand(time(NULL));
15
16     ull t = rand(atoi(argv[2]), atoi(argv[2]));
17     while (t--)
18     {
19         cout << rand(1, 31) << "." << rand(1, 12) << "." << rand(1e3, 9999) << '\t' <<
               rand(1, ULLONG_MAX) << '\n';
20     }
21 }
```

Для измерения производительности я посчитал время работы написанной мной поразрядной сортировки и std::sort на одинаковых входных данных. Замерение времени работы производилось с помощью библиотеки std::chrono, позволяющей фиксировать временные моменты: начала, конца сортировки, например, как в моем случае. Остается вычесть первое из второго и получить искомое время работы той или иной сортировки.

```
[leo@pc benchmark]$ wc -l test* | head -n 5 | tail -n 5
1000 test1e3
10000 test1e4
100000 test1e5
1000000 test1e6
10000000 test1e7
[leo@pc benchmark]$ ./prog_both <test1e3
radix sec: 0.000154777
quick sec: 0.000230654
[leo@pc benchmark]$ ./prog_both <test1e5
radix sec: 0.0105862
quick sec: 0.0398631
```



Кол-во строк	std::sort	Radix Sort
20000000	11.3825	2.23262
10000000	5.66991	1.16744
2000000	1.01335	0.212634
1000000	0.504287	0.155962
200000	0.0948895	0.0219994
100000	0.0389095	0.0104155
20000	0.0106709	0.00549445
10000	0.00300164	0.00113038
2000	0.000495543	0.000234246
1000	0.000231345	0.000148544

Как видно по результатам тестирования, на все большего размера входных данных поразрядная сортировка все пуще выигрывает у стандартной. Наблюдательно, что даже на маленьких тестах поразрядная сортировка эффективней быстрой из <strdlib.h>.

5 Выводы

По мере выполнения данной лабораторной работы я освежил в памяти написание классов, шаблонов. Вспомнил про то, как писать многофайловые проекты. В момент, когда возникла необходимость спарсить ключ-строку, я пошел кропотливо изучать стандартные функции для работы с C-style строками; уверен, в будущем они мне еще не раз пригодятся. Также я закрепил у себя в памяти реализацию поразрядной сортировки, сортировки подсчетом.

Благодаря же суровому чекеру я отрекся от упущений в виде объявления неиспользуемых переменных, утечек памяти и других недочетов.

Также удовольствие мне принесло тестирование программы: построение графиков, анализ результатов на разного размера входных данных. Благо, до написания лабораторной я уже вовсю пользовался тестированием программ в условиях контестов, что крайне выручало меня, если я не мог найти ошибку в коде.