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: Т. А. Бердикин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б
Дата:
Оценка:
Подпись:

Москва, 2019

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` $10^6 - 1$`, почтовые индексы.`

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

1 Описание

Основная идея сортировки подсчётом, алгоритма со списком. Этот вариант используется, когда на вход подается массив структур данных, который следует отсортировать по ключам (назовём их key, а максимальный ключ пусть будет k). Нужно создать вспомогательный массив $C[0..k-1]$, каждый $C[i]$ в дальнейшем будет содержать список элементов из входного массива. Затем последовательно прочитать элементы входного массива A, каждый $A[i]$ добавить в список $C[A[i].key]$. В заключении пройти по массиву C, для каждого $j \in [0..k-1]$ в массив A последовательно записывать элементы списка $C[j]$. Алгоритм устойчив - это значит, что он не будет менять относительный порядок сортируемых элементов, имеющих одинаковые ключи. Wiki.

2 Исходный код

Входные данные состоят из пар: ключ, значение, где ключ - это почтовый индекс, а значение - это 64-битное число в десятичном представлении. Так как длинна нашего ключа ничем не зафиксирована, то необходимо его привести к какому-то единому стандарту. Этим стандартом является число длиной в 6 символов,так как это максимальная длинна почтового индекса. Значит, что? Если во входных данных индекс имеет менее шести знаков, чтобы было красиво, допишем незначащие нули спереди с помощью функций $setw$ и $setfill$ из библиотеки $< iomanip >$. Создадим класс $TInsex$ в котором будут храниться ключи и значения, функция Set, которая поможет нам в создании пар ключ-значение, функция $Show$, выводящая элементы массива. Также дружественной функцией туда записана сортировка подсчётом. Теперь самое интересное - функция $main$. Узнал, что существенно может ускорить программу рассинхронизация потоков с помощью строки $ios :: sync_with_stdio(false);$. Дальше всё просто: создаём переменные ключа и значения, потом вектор с лаконичным названием a, а затем переменную tmp, которая и будет парой, объединяющей ключ и значение. Вводим ключи и значения через, неожиданно, $scanf$, потому что это быстрее. Неоднократно проверено с помощью *chrono*, о котором подробнее в разделе «Тест производительности». Скорее всего, это связано с тем, что cin и cout должны синхронизироваться со стандартной библиотекой Си. Затем применяем функцию Set, а затем получившуюся пару помещаем в вектор. Повторяем ввод и нижеописанные действия, пока не введём всё, что нам нужно. Затем всё сортируем и выводим.

```
1  #include <iostream>
2  #include <iomanip>
3  #include <algorithm>
4  #include <cassert>
5
6  using namespace std;
7
8  const int SIZE_ARRAY = 1000000;
9
10 template <typename T>
11 class TVector {
12 public:
13     using value_type = T;
14     using iterator = value_type*;
15     using const_iterator = const value_type*;
16
17     TVector():
18         already_used_(0), storage_size_(0), storage_(nullptr)
19     {
20     }
21
22     TVector(int size, const value_type& default_value = value_type()):
23         TVector()
24     {
25         assert(size >= 0);
26
27         if (size == 0)
28             return;
29
30         already_used_ = size;
31         storage_size_ = size;
32         storage_ = new value_type[size];
33
34         fill(storage_, storage_ + already_used_, default_value);
35     }
36
37     int Size() const
```

```
38      {
39          return already_used_;
40      }
41
42      /*bool empty() const
43      {
44          return Size() == 0;
45      }
46
47      iterator begin() const
48      {
49          return storage_;
50      }
51
52      iterator end() const
53      {
54          if (storage_)
55              return storage_ + already_used_;
56
57          return nullptr;
58      }*/
59
60      friend void Swap(TVector& lhs, TVector& rhs)
61      {
62          using std::swap;
63
64          swap(lhs.already_used_, rhs.already_used_);
65          swap(lhs.storage_size_, rhs.storage_size_);
66          swap(lhs.storage_, rhs.storage_);
67      }
68
69      TVector& operator=(TVector other)
70      {
71          Swap(*this, other);
72          return *this;
73      }
74
75      TVector(const TVector& other):
76          TVector()
77      {
78          TVector next(other.storage_size_);
79          next.already_used_ = other.already_used_;
80
81          if (other.storage_ )
82              copy(other.storage_, other.storage_ + other.storage_size_,
83                       next.storage_);
84
85          Swap(*this, next);
86      }
87
88      ~TVector()
89      {
90          delete[] storage_;
91
92          storage_size_ = 0;
93          already_used_ = 0;
94          storage_ = nullptr;
95      }
96
97      void PushBack(const value_type& value)
98      {
99          if (already_used_ < storage_size_) {
```

```
100             storage_[already_used_] = value;
101             ++already_used_;
102             return;
103         }
104 
105         int next_size = 1;
106         if (storage_size_)
107             next_size = storage_size_ * 2;
108 
109         TVector next(next_size);
110         next.already_used_ = already_used_;
111 
112         if (storage_ )
113             std::copy(storage_, storage_ + storage_size_, next.storage_);
114 
115         next.PushBack(value);
116         Swap(*this, next);
117     }
118 
119     const value_type& At(int TInsex) const
120     {
121         if (TInsex < 0 || TInsex > already_used_)
122             throw std::out_of_range("You are doing this wrong!");
123 
124         return storage_[TInsex];
125     }
126 
127     value_type& At(int TInsex)
128     {
129         const value_type& elem = const_cast<const TVector*>(this)-> At(TInsex);
130         return const_cast<value_type&>(elem);
131     }
132 
133     const value_type& operator[](int TInsex) const
134     {
135         return At(TInsex);
136     }
137 
138     value_type& operator[](int TInsex)
139     {
140         return At(TInsex);
141     }
142 
143 private:
144     int already_used_;
145     int storage_size_;
146     value_type* storage_;
147 };
148 
149 typedef unsigned long long long_t;
150 
151 class TInsex{
152 public:
153     long_t key, elem;
154     TInsex(int k, long_t e) : key(k), elem(e) {}
155     TInsex() : key(), elem() {}
156     friend TInsex* CountingSort(TInsex* obj, long_t max, size_t n);
157     void Show(){
158       cout << setw(6) << setfill('0') << key << " " << elem << '\n';
159     }
160     TInsex Set(long_t k, long_t e){
161       elem = e;
```

```
      key = k;
      return *this;
    }
  };

void CountingSort(TVector <TInsex>& a){
  int *c = new int[SIZE_ARRAY]{0};
  TVector <TInsex> out(a.Size());

  for (size_t i = 0; i < a.Size(); ++i){
    c[a[i].key]++;
    //c[a[i].key] = a[i].elem;
  }
  for (int i = 1; i < SIZE_ARRAY; ++i){
    c[i] += c[i - 1];
  }
  for (int i = a.Size() - 1; i >= 0; i--){
    out[--c[a[i].key]] = a[i];
  }
  Swap(out, a);
  delete[] c;
}

int main(){
    ios::sync_with_stdio(false);
    long_t e, k;
    TVector <TInsex> a;
    TInsex tmp;

    while (scanf("%llu%llu", &k, &e) == 2) {
      tmp.Set(k, e);
      a.PushBack(tmp);
    }

    CountingSort(a);

    for (int i = 0; i < a.Size(); i++){
      a[i].Show();
    }
    return 0;
}
```

3 Консоль

```
tim@wrangler:~/Desktop/da/1$ g++ main.cpp
tim@wrangler:~/Desktop/da/1$ cat test1.txt
534095 345092598342
533553 539080594648
354253 532555523555
525 5324534
098432 3542098
43444 423904
1 1
525 5487369
tim@wrangler:~/Desktop/da/1$ ./a.out < test1.txt
000001    1
000525    5324534
000525    5487369
043444    423904
098432    3542098
354253    532555523555
533553    539080594648
534095    345092598342
```

4 Тест производительности

В тесте производительности пришлось реализовать целых две программы, в каждой из которых генерируются ключи для сортировки, количество которых, в свою очередь, задаётся пользователем. Для замера времени сортировки используются функции из библиотеки $< chrono >$. Они замеряют время до начала сортировки и сразу после конца, вычитают одно из другого и выводят результат в миллисекундах. Забавно, что время считается от 1.01.1970. Сравнивал я, собственно, сортировку подсчётом и $std :: sort$, то бишь, сортировку Хоара.

```
tim@wrangler:~/Desktop/da/1$ ./count
10
The time: 6 ms
tim@wrangler:~/Desktop/da/1$ ./count
100
The time: 7 ms
tim@wrangler:~/Desktop/da/1$ ./count
1000
The time: 8 ms
tim@wrangler:~/Desktop/da/1$ ./count
10000
The time: 12 ms
tim@wrangler:~/Desktop/da/1$ ./count
100000
The time: 28 ms
tim@wrangler:~/Desktop/da/1$ ./count
1000000
The time: 82 ms
tim@wrangler:~/Desktop/da/1$ ./count
10000000
The time: 838 ms
tim@wrangler:~/Desktop/da/1$ ./count
100000000
The time: 10692 ms
tim@wrangler:~/Desktop/da/1$ ./std
10
The time: 0 ms
tim@wrangler:~/Desktop/da/1$ ./std
100
The time: 0 ms
tim@wrangler:~/Desktop/da/1$ ./std
1000
The time: 0 ms
tim@wrangler:~/Desktop/da/1$ ./std
10000
The time: 10 ms
tim@wrangler:~/Desktop/da/1$ ./std
100000
The time: 29 ms
tim@wrangler:~/Desktop/da/1$ ./std
1000000
```

```
The time: 334 ms
tim@wrangler:~/Desktop/da/1$ ./std
10000000
The time: 3828 ms
tim@wrangler:~/Desktop/da/1$ ./std
100000000
The time: 41909 ms
```

Из результатов могу выделить то, что при этих тестах, работает быстрее сортировка подсчётом, чем Хоара. Видно, что сложность соритровки Хоара $O(nlogn)$, а сортировки подсчётом - $O(n)$. Сответственно, при большом количестве тестов сортировка подсчётом работает быстрее, чем сортировка Хоара, а при малом - наоборот, что видно при тестах.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился работать со строками, выделяя нужное и преобразуя их в однотипные значения, и вставлять их в вектор. Также я реализовал сортировку подсчетом. Кроме того я сравнил её с сортировкой Хоара, вызывая встроенную функцию std::sort.

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`