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. Д. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22

Дата: 21.03.2024
Оценка:
Подпись:

Москва, 2024

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: `телефонные номера, с кодами стран и городов в формате +<код страны> <код города> телефон.`

Вариант значения: `Строки фиксированной длины 64 символа, во входных данных могут встретиться строки меньшей длины, при этом строка дополняется до 64-х нулевыми символами, которые не выводятся на экран.`

1 Описание

Требуется написать реализацию алгоритма поразрядной сортировки. Для этого необходимо также реализовать алгоритм сортировки подсчетом.

Алгоритм звучит так: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца. [1].

Сортировку каждого разряда будем производить с помощью сортировки подсчетом.

2 Исходный код

Определим несколько констант: MAX_LENGTH_OF_NUMBER - максимальная длина номера телефона (тип: const size_t); MAX_LENGTH_OF_VALUE - максиимальная длина значения (тип: const size_t); DEFAULT - начальная вместимость вектора (тип: const size_t).

Теперь опишем шаблон класса вектора. Будем хранить три поля: buffer - в нем будут храниться элементы вектора (тип: T*); size - будет хранить в себе информацию о количестве элементов в векторе (тип: size_t); capacity - будет хранить в себе информацию о вместимости вектора (тип: size_t). У шаблона есть несколько методов: TVector() (стандартный конструктор) - создает, выделяет память под вектор и инициализирует необходимые поля; void Push_back(const T value) - добавляет элементы в конец вектора, при необходимости расширяет буфер; size_t Size() - возвращает значение поля size; T operator[](size_t idx) (перегруженный оператор []) - возврает ссылку на эелемент стоящий на idx'ом индексе в буффере; void Expand() - расширяет буффер в два раза; TVector() (деструктор) - очищает буфер во избежание утечек памяти.

На каждой непустой строке файла располагается пара ключ-значение. Ключ представлен в виде номера телефона, а значение в виде строки фиксированной длины 64 литеры. Создадим класс для удобного хранения номера телефона. В классе будем хранить несколько полей: null_counter - счетчик нулей перед номером, он нам пригодится в алгоритме сортировки (тип: uint_8t); full_number - массив для хранения самого номера телефона (тип: char[MAX_LENGTH_OF_NUMBER]); idx - индекс, необходимый для поиска значения в массиве значений принадлежащего этому ключу (тип: size_t). Теперь считываем две строки, разделенные символом табуляции: номер телефона и значения. Сохраняем длину номера в переменную, для дальнейшего определения максимальной длины номера. Далее определяем необходимое количество нулей перед номером для вводимого номера, также определяем индекс для значения ключа в массиве значений. А теперь помещаем структуру номера телефона и значение в два разных вектора. Теперь можно начинать сортировать вектор.

В начале поразрядной сортировки определим массив для хранения результата сортировки подсчетом для каждого разряда. Далее, начиная с конца номера, начинаем проходить по разрядам номера и применять к этим разрядам сортировку подсчетом, перед этим проверив, не являются ли текуций разряд номера символом дефиса, если нет, то начинаем сортировку подсчетом, в противном случае продолжаем проход по номеру дальше.

В сортировке подсчетом создаем массив для хранения префиксных сумм и инициализируем его нулями. Далее инкриментируем значения в массиве префиксных сумм, индекс которых соответствует значениям в текущем разряде номеров. Далее, проходя весь массив от начала до конца, прибавляем к текущему элементу массива

предыдущий. Таким образом получаем массив префиксных сумм. Далее, начиная с конца вектора, распределяем по результирующему массиву на места, соответствующие значению в массиве префиксных сумм, стоящему на индексе, равному значению текущего разряда номера. Это значение в массифе префиксных сумм декриментируем. Результирующий массив копируем в вектор.

Далее, после прохождения всего алгоритма сортировки печатаем на экран получившийся вектор.

```
#include <iostream>
#include <string>
#include <cstring>
#include <memory>

const size_t MAX_LENGTH_OF_NUMBER = 16;
const size_t MAX_LENGTH_OF_VALUE = 64;
const size_t DEFAULT = 1000000;

class TPhoneNumber
{
public:
    uint8_t null_counter;
    char full_number[MAX_LENGTH_OF_NUMBER];
    size_t idx;
};

namespace NVector
{
    template <class T>
    class TVector
    {
    private:
        std::size_t size;
        std::size_t capacity;

        void Expand()
        {
            T *new_buffer = new T[capacity * 2];
            capacity *= 2;
            for (std::size_t i = 0; i < size; ++i)
            {
                new_buffer[i] = std::move(buffer[i]);
            }
            delete[] buffer;
            buffer = new_buffer;
        }

    public:
        T *buffer;
```

```
41          TVector()
42          {
43              buffer = new T[DEFAULT];
44              size = 0;
45              capacity = DEFAULT;
46          }
47
48          void Push_back(const T& value)
49          {
50              if (size == capacity)
51              {
52                  Expand();
53              }
54              buffer[size] = value;
55              ++size;
56          }
57
58          size_t Size()
59          {
60              return size;
61          }
62
63          T& operator[](std::size_t idx)
64          {
65              return buffer[idx];
66          }
67
68          ~TVector()
69          {
70              delete[] buffer;
71          }
72      };
73  }
74
75  void CountingSort(NVector::TVector<TPhoneNumber>& arr, uint8_t& idx, TPhoneNumber *
        result)
76  {
77      // Temporary array for digits
78      int tmp[10] = {0};
79      // Counting of quantity digits in numbers
80      for (size_t i = 0; i < arr.Size(); ++i)
81      {
82          tmp[(idx <= arr[i].null_counter ? '0' : arr[i].full_number[idx - arr[i].
                null_counter]) - '0']++;
83      }
84      // Counting of prefix sums
85      for (size_t i = 1; i < 10; ++i)
86      {
87          tmp[i] += tmp[i - 1];
```

```
88      }
89      // Destribution elements in result array
90      for (size_t i = arr.Size(); i > 0; --i)
91      {
92          --tmp[(idx <= arr[i - 1].null_counter ? '0' : arr[i - 1].full_number[idx - arr[
                i - 1].null_counter]) - '0'];
93          result[tmp[(idx <= arr[i - 1].null_counter ? '0' : arr[i - 1].full_number[idx -
                 arr[i - 1].null_counter]) - '0']] = arr[i - 1];
94      }
95      // Copying of sorted elements in result array
96      for (size_t i = 0; i < arr.Size(); ++i)
97      {
98          arr[i] = std::move(result[i]);
99      }
100 }
101
102 void RadixSort(NVector::TVector<TPhoneNumber>& arr, uint8_t& max_len)
103 {
104     // Allocating counting sort result array
105     TPhoneNumber *result = new TPhoneNumber[arr.Size()];
106     for (uint8_t i = max_len - 1; i > 0; --i)
107     {
108         // '-' symbols in 4th and 8th idx's
109         if ((i == 8) || (i == 4))
110         {
111             continue;
112         }
113         CountingSort(arr, i, result);
114     }
115 }
116
117 int main()
118 {
119     std::ios::sync_with_stdio(false);
120     std::cin.tie(0);
121     std::cout.tie(0);
122
123     TPhoneNumber number;
124     char value[MAX_LENGTH_OF_VALUE];
125
126     NVector::TVector<TPhoneNumber> arr;
127     NVector::TVector<std::shared_ptr<std::string>> valarr;
128     uint8_t max_len = 0;
129     // Reading elements
130     while (scanf("%s", number.full_number) != EOF)
131     {
132         scanf("%s", value);
133         // Counting of max length of number
134         max_len = (strlen(number.full_number) > max_len ? strlen(number.full_number) :
```

```
            max_len);
        // Null digits counter forward number
        number.null_counter = MAX_LENGTH_OF_NUMBER - strlen(number.full_number);
        // Idx for value array
        number.idx = arr.Size();
        // Filling of value array
        valarr.Push_back(std::make_shared<std::string>(std::string(value)));
        // Filling of key array
        arr.Push_back(number);
    }
    // Sorting elements
    RadixSort(arr, max_len);
    // Printing elements
    for (size_t i = 0; i < arr.Size(); ++i)
    {
        printf("%s\t%s\n", arr[i].full_number, valarr[arr[i].idx]->c_str());
    }
    return 0;
}
```

main.cpp	
void CountingSort(NVector::TVector<TPhoneNumber>& arr, uint8_t& idx, TPhoneNumber *result)	Функция сортировки подсчётом
void RadixSort(NVector::TVector<TPhoneNumber>& arr, uint8_t& max_len)	Функция поразрядной сортировки
int main()	Входная точка программы

```
class TPhoneNumber
{
public:
    uint8_t null_counter;
    char full_number[MAX_LENGTH_OF_NUMBER];
    size_t idx;
};

template <class T>
class TVector
{
private:
    std::size_t size;
    std::size_t capacity;

    void Expand();

public:
    T *buffer;
    TVector();
```

```
21
22      void Push_back(const T& value);
23
24      size_t Size();
25
26      T& operator[](std::size_t idx);
27
28      ~TVector();
29  };
```

3 Консоль

```
lexasy@MSI:$ g++ main.cpp
lexasy@MSI:$ cat test
+7-495-1123212  n399tann9nnt3
+375-123-1234567        n399tann
+7-495-1123212  bugaga
+375-123-1234567        yahooo
lexasy@MSI:$ ./a.out <test
+7-495-1123212  n399tann9nnt3
+7-495-1123212  bugaga
+375-123-1234567        n399tann
+375-123-1234567        yahooo
```

4 Тест производительности

Тест производительности представляет из себя следующее: сортировка 100000 рандомно сгенерированных пар ключ-значение с помощью самостоятельно реализованной сортировки подсчетом и стандартной устойчивой сортировкой.

```
lexasy@MSI:~/Desktop/Prog/DA_labs/lab1$ g++ main.cpp
lexasy@MSI:~/Desktop/Prog/DA_labs/lab1$ ./a.out <tests/01.t >tmp
lexasy@MSI:~/Desktop/Prog/DA_labs/lab1$ cat tmp | grep "time"
time: 21218ms
lexasy@MSI:$ g++ stl.cpp
lexasy@MSI:$ ./a.out <tests/01.t >tmp
lexasy@MSI:1$ cat tmp | grep "time"
time: 58896ms
```

Как видно, самодельная сортировка выигрывает у стандартной больше чем в два раза. Это объясняется просто: сложность std::stable_sort - $O(n \log n)$, а сложность написанной нами сортировки - $O(n * k)$, где n - количество сортируемых элементов, а k - максимальная длина номера. Допустим мы возьмем $k = 16$ и $n = 100000$ как в этом тесте. $\log 100000 > 16$, следовательно при одинаковых n, выигрыш написанной нами сортировки по времени на лицо.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать поразрядную сортировку в паре с сортировкой подсчетом. Также я узнал, что та сортировка подсчетом, которую я реализовывал еще в школе, является неправильной, и она может сортировать только числа. А сортировка подсчетом, которую я реализовал в данной лабораторной работе, может сортировать сложные структуры. Также я еще раз напомнил себе как реализовывать структуру вектора. Во время выполнения лабораторной работы я столкнулся со сложностью в виде большого расхода памяти. Из-за этого пришлось избавиться от копирований значений. Это дало невероятное уменьшение сложности по памяти. В будущем я буду за этим пристально следить и не допускать ненужных копирований.

Список литературы

[1] https://ru.wikipedia.org/wiki/Поразрядная_сортировка (дата обращения: 20.03.2024)