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: В. П. Будникова
Преподаватель: А. А. Кухтичев
Группа: М8О-209Б-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: Карманная сортировка.

Тип ключа: `Вещественные числа в промежутке [-100, 100].`

Тип значения: `Числа от 0 до 264 - 1.`

Формат входных данных: На каждой непустой строке входного файла располагается пара «ключ-значение», в которой ключ указан согласно заданию, затем следует знак табуляции и указано соответствующее значение.

Формат результата: Выходные данные состоят из тех же строк, что и входные, за исключением пустых и порядка следования.

1 Описание

Как сказано в [1]: «Карманная сортировка - алгоритм сортировки, в котором сортируемые элементы распределяются между конечным числом отдельных блоков (карманов, корзин) так, чтобы все элементы в каждом следующем по порядку блоке были всегда больше (или меньше), чем в предыдущем. Каждый блок затем сортируется отдельно, либо рекурсивно тем же методом, либо другим. Затем элементы помещаются обратно в массив.». «Предполагается, что входные числа подчиняются закону распределения» [2]. При входе числа(ключи) распределяются по карманам, для этого требуется вспомогательный массив(или вектор), каждое поле которого - это карман(вектор). На входе каждое число(ключ) попадает в соответсвующий карман, так как предполагается что числа распределены равномерно, в один карман попадает не очень много элементов, поэтому на сортировку каждого кармана нам не потребуется много времени. Благодаря такому распределению, время карманной сортировки в среднем случае будет равно O(n). В данной лабораторной работе при сортировке карманов используется [3]: «Сортировка вставками - алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов». После сортировка каждого кармана числа последовательно из каждого кармана записываются обратно в исходной вектор(или выдаются на экран).

2 Исходный код

Так как в данной лабораторной работе запрещается использовать стандартный шаблон языка C++ - std:vector<T>, реализуем «свой» класс вектор, также реализуем добавление элемента в вектор(PushBack), перегрузим оператор «[ ]», для удобного доступа к элементам вектора и реализуем метод нахождения длины вектора(SizeVector). Так как вектор - динамический объект, то при добавлении нового элемента, если длина вектора совпадает с его вместимостью, т.е. для нового элемента не хватает места, будем увеличивать каждый раз вместимость вектора в 2 раза(перевыделяя соответствующую память). На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $TPair$, в которой будем хранить ключ и значение. Количество карманов в векторе для сортировки равно количеству пар, послупивших на вход плюс 1. При считывании пары «ключ-значение», элемент будет попадать определённую ячейку этого вектора. Так как номера ячеек принаджедат отрезку - [0,n+1], где n - количество пар, поступивших на вход, то для нахождения индекса кармана, в который попадет элемент, будем "сдвигать"все ключи на 100 и делить получившееся числе на размер кармана. Размер кармана получается путем деления длины отрезка(по условию задачи отрезок - [-100,100], следовательно его длина - 200) на количество элементов. При реализации деструктора вектора требуется освободить память, аналогично память освобождается при переопределении вместимости вектора. Также реализован конструктор вектора, на вход которого подается число типа unsigned int, которое является размером вектора, при этом также выделяется соответствующая память. При сортировке мы проходимся по данному вектору, и , если длина его кармана больше 1(если длина равна 1, то карман считается отсортированным), то сортируем этот карман с помощью сортировки вставками. После сортировки проходим по вектору и последовательно из карманов выдаем элементы(пары «ключ-значение») на экран.

```
1
2  #include "stdio.h"
3  namespace NVector
4  {
5      template<class T>
6      class TVector {
7          public:
8              TVector() {
9                  size = 0;
10                 capacity = 0;
11                 data = 0;
12             }
13             ~TVector() {
14                 delete[] data;
15             }
16             TVector(unsigned long max) {
17                 data = new T[max];
18                 size = 0;
19                 capacity = max;
20             }
21             void PushBack(const T elem);
22             T &operator[](unsigned int ind);
23             unsigned int SizeVect();
24         private:
25             unsigned int size;
26             unsigned int capacity;
27             T * data;
28     };
29     template<class T>
30     void TVector<T>::PushBack(const T elem) {
31         if (capacity == 0){
32             data = new T[4];
33             capacity = 4;
34         } else {
35             if (size == capacity) {
36                 T * data1 = new T[capacity << 1];
37                 capacity <<= 1;
38                 for (int i = 0; i < size; ++i) {
39                     data1[i] = data[i];
40                 }
41                 delete[] data;
42                 data = data1;
43             }
44         }
45         data[size] = elem;
46         size ++;
47     }
48     template<class T>
49     T &TVector<T>::operator[](unsigned int ind) {
```

```
50          return data[ind];
51      }
52      template<class T>
53      unsigned int TVector<T>::SizeVect() {
54          return size;
55      }
56  }
57  struct TPair {
58      double key;
59      unsigned long long value;
60  };
61  int main() {
62      NVector::TVector<TPair> elems;
63      TPair p;
64      while (scanf("\%lf \%llu", &p.key, &p.value) > 0) {
65          elems.PushBack(p);
66      }
67      if (elems.SizeVect() == 0) {
68          return 0;
69      }
70      NVector::TVector<NVector::TVector<TPair>> bucket(elems.SizeVect() + 1);
71      double tempsize = 200 / (double)elems.SizeVect();
72      for (unsigned int i = 0; i < elems.SizeVect(); ++i) {
73          bucket[(int)((elems[i].key + 100) / tempsize)].PushBack(elems[i]);
74      }
75      TPair temp;
76      for (unsigned int i = 0; i < elems.SizeVect() + 1; ++i) {
77          if (bucket[i].SizeVect() >= 2) {
78              for (unsigned int j = 1; j < bucket[i].SizeVect(); ++j) {
79                  double key = bucket[i][j].key;
80                  temp = bucket[i][j];
81                  int k = j - 1;
82                  while (k >= 0 && bucket[i][k].key > key) {
83                      bucket[i][k + 1] = bucket[i][k];
84                      k--;
85                  }
86                  bucket[i][k + 1] = temp;
87              }
88          }
89      }
90      for (unsigned int i = 0; i < elems.SizeVect() + 1; ++i) {
91          if (bucket[i].SizeVect() != 0) {
92              for (unsigned int j = 0; j < bucket[i].SizeVect(); ++j) {
93                  printf("\%f \%llu\n", bucket[i][j].key, bucket[i][j].value );
94              }
95          }
96      }
97      return 0;
98  }
```

3 Консоль

```
Lera:lb valeriabudnikova$ g++ -std=c++14 -pedantic lab11.cpp -o lab1
Lera:lb valeriabudnikova$ cat test0.txt
-48.909065 17544704456574172030
87.798131 9381235158324630206
73.295752 2270329736173411256
62.211664 5417349800616700252
75.491021 7354860280882021016
43.690995 6781511321330931577
68.921311 15366352371649901594
-37.578724 7314883924904900233
-35.146131 12819139162760241371
17.264407 13953212136721859399
19.976025 16320110372271155572
96.998241 7242514149444554261
99.155696 16134476325468815823
45.898096 2971230429791814369
-46.275508 15774253782884946386
-18.927859 2972582474069404232
-93.681717 10091883519186326617
-14.994226 10641803510538946514
Lera:lb valeriabudnikova$ ./lab1 <test0.txt
-93.681717 10091883519186326617
-48.909065 17544704456574172030
-46.275508 15774253782884946386
-37.578724 7314883924904900233
-35.146131 12819139162760241371
-18.927859 2972582474069404232
-14.994226 10641803510538946514
17.264407 13953212136721859399
19.976025 16320110372271155572
43.690995 6781511321330931577
45.898096 2971230429791814369
62.211664 5417349800616700252
68.921311 15366352371649901594
73.295752 2270329736173411256
75.491021 7354860280882021016
87.798131 9381235158324630206
96.998241 7242514149444554261
99.155696 16134476325468815823
```

4 Тест производительности

Сравним время работы карманной сортировки с временем работы Стабильной сортировки из стандартной библиотеки C++(stable_sort). Проверим на 200 элементах:

```
Lera:lb valeriabudnikova$ ./timelab1 <test0.txt
Карманная: 1.6e-05
Стабильная из std: 9.4e-05
```

На 200 элементах Карманная выигрывает у Стабильной, практически каждый карман будет содержать только один элемент, следовательно сложность такого алгоритма будет O(n), так как нам необходимо пройтись только один раз по вектору, чтобы его выдать.

Проверим на 200 элементах, которые распределены не равномерно(для этого сгенерируем такие числа, чтобы они лежали, например, в отрезке [99.933,99.99]:

```
Lera:lb valeriabudnikova$ ./timelab1 <test0.txt
Карманная: 0.000308
Стабильная из std: 0.000123
```

Карманная сортировка проиграла Стабильной, так как числа распределены не равномерно, следовательно ммного чисел попадает в один и тот же карман, а сортировка карманов(Сортировка вставками) в худшем случае имеет сложность $O(n^2)$

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научилась реализовывать собственный вектор, использовать шаблоны, что оказалось очень удобным, так как при создании вектора, состоящего из векторов, в свою очередь состоящих из пар «ключ-значение» можно было абстрагироваться от конкретных типов, выделения дополнительной памяти под них, реализаций операций над ними. Также использования шаблонов даёт возможность использовать этот класс в других программах, где будет необходим вектор, при этом не задумываться его типе. Я узнала преимущества объектов заголовочного файла iostream - cin и cout и функций scanf(), printf(). У cin и cout есть преимущество - можно перегружать операторы « и », следовательно можно выдавать различные объекты, при этом не задумываясь о их типе, плюс это улучшает читаемость кода. При очень больших входных данных cin и cout работают немного медленней, чем scanf() и printf(), я узнала, что причиной этому является то, что по умолчанию iostream использует систему буферизации stdio, следовательно при использовании cin и cout тратится время на синхронизацию с буфером stdio. Позже, поисков информацию, я узнала, что возможно избежать этого, используя: std::ios::sync_with_stdio(false); [4]: sync_with_stdio - «устанавливает, синхронизируются ли стандартные потоки C ++ со стандартными потоками C после каждой операции ввода / вывода. По умолчанию все восемь стандартных потоков C ++ синхронизируются с соответствующими потоками C.» Также в этой лабораторной работе я научилась реализовывать Карманную сортировку, узнала от чего зависит скорость сортировки. Для себя я сделала вывод, что перед тем, как что-либо сортировать, нужно проанализировать входные данные и потом уже думать, какой сортировкой лучше воспользоваться.

Список литературы

[1] *Карманная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Блочная_сортировка` (дата обращения: 06.10.2020).

[2] 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.))

[3] *Сортировка вставками — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Сортировка_вставками` (дата обращения: 06.10.2020).

[4] *sync_ with_ stdio*
URL: `https://en.cppreference.com/w/cpp/io/ios_base/sync_with_stdio` (дата обращения: 06.10.2020).