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: К. А. Калугин
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

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: `Числа от 0 до 65535.`

Вариант значения: `строки переменной длины (до 2048 символов).`

1 Описание

Суть поразрядной сортировки заключается в том, что нам необходимо взять младшие разряды чисел и отсортировать числа по ним (сортировка должны быть устойчивой), после чего отсортировать получившийся набор снова, уже по следующему разряду. Такую сортировку необходимо продолжать до тех пор, пока не закончаться разряды в самом большом числе.

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $TData$, в которой будем хранить ключ в виде ссылки на строку и значение. Далее создадим цикл, который будет выполнятся, пока на вход подаются строки. Сами будут записываться в динамически выделяющуюся память, а хранить в созданном нами динамическом массиве *active* мы будем лишь ключи и ссылки на память. В случае, если место в нем заканчивается (изначально размер *active*'а - всего два элемента), мы создаем новый массив в два раза большего размера и переносим в него все наши даннные. После считывания всех строк вызывается функция $Sort$. В ней мы создаем массив *counter* размера 65536, инициализируем его нулями, после чего делаем побитовое «И» с числом, состоящим из 16 единиц, сдвинутых, в зависимости от номера итерации на 0, 16, 32 или 48 бит. Это позволяет нам разбить ключи на разряды по 16 бит каждый, и как следствие, существенно ускорить работу программы. После того, как мы получаем разряд очередного числа, мы увеличиваем значение соответствующего элемента массива count на 1. Когда строки кончаются, цикл проходит по массива со второго до последнего элемента, увеличивая значение каждой клетки на значение предыдущей. Это сделано для того, чтобы реализовать устойчивую сортировку подсчетом. И наконец, в последнем цикле, строки расставляются в нужном порядке, основываясь на данных из массива *counter*. Этот алгоритм повторяется 4 раза (по количеству 16-битных разрядов в 64-битном числе), после чего на вывод поступает уже отсортированный массив. В последних строчках программы реализуется простейший вывод всех элементов массива и очистка памяти.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  const int M_BIT = 65536;
6  const unsigned long long int MASK = (1 << 16) - 1;
7
8  struct TData {
9      unsigned long long int key;
10     string* data;
11 };
12
13 TData* Sort (TData* active, int count) {
14     unsigned long long int counter [M_BIT];
15     TData *result = new TData [count];
16     TData* buffer;
17     for (int N = 0; N < 4; N ++) {
18         for (int i = 0; i < M_BIT; i ++) {
19             counter [i] = 0;
20         }
```

```
21          for (int i = 0; i < count; i ++) {
22              counter[((active [i].key & (MASK << N * 16)) >> N * 16)] ++;
23          }
24          for (int i = 1; i < M_BIT; i ++) {
25              counter [i] += counter [i - 1];
26          }
27
28          for (int i = count - 1; i >= 0; i --) {
29              counter[((active [i].key & (MASK << N * 16)) >> N * 16)] --;
30              result [counter[((active [i].key & (MASK << N * 16)) >> N * 16)]] = active
                    [i];
31          }
32          buffer = active;
33          active = result;
34          result = buffer;
35      }
36
37      delete [] result;
38      return (active);
39  }
40
41
42  int main() {
43      ios::sync_with_stdio(false);
44      cin.tie(0);
45      cout.tie(0);
46      TData line;
47      int count = 0;
48      int mCount = 2;
49      TData* active = new TData [mCount];
50      while (cin >> line.key) {
51          string* buffer = new string;
52          cin >> *buffer;
53          line.data = buffer;
54          if (count == mCount - 1) {
55              mCount *= 2;
56              TData* extra = new TData [mCount];
57              for (int i = 0; i < count; i ++) {
58                  extra [i] = active [i];
59              }
60              delete [] active;
61              active = extra;
62          }
63          active[count].key = line.key;
64          active[count].data = line.data;
65          count ++;
66      }
67
68      active = Sort (active, count);
```

```
69
70
71
72      for (int i = 0; i < count ; i++) {
73          cout << active [i].key << "\t";
74          cout << *(active [i].data) << "\n";
75          delete active [i].data;
76      }
77      delete [] active;
78      return 0;
79  }
```

3 Консоль

```
35 ddd
7 hlkjl
1 bbb
2 ccc
0 aaa
9999 fghk
85412 hjljkl
999999 gkhjk
22 hjkhk
333 gjkhl'';' 0  xGfxrxGGxrxMMMMfrrrG
0 xGfxrxGGxrxMMMMfrr
18446744073709551615  xGfxrxGGxrxMMMMfrrr
18446744073709551615  xGfxrxGGxrxMMMMfr
34 hjghj

0 aaa
0 xGfxrxGGxrxMMMMfrrrG
0 xGfxrxGGxrxMMMMfrr
1 bbb
2 ccc
7 hlkjl
22 hjkhk
34 hjghj
35 ddd
333 gjkhl'';' 9999 fghk
85412 hjljkl
```

```
999999 gkhjk
18446744073709551615 xGfxrxGGxrxMMMMfrrr
18446744073709551615 xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Для проверки производительности сравним быстродействие нашей сортировки и стандартной сортировки $std :: stable\_sort$. Наборы тестов состоят из 10^5, 10^6, 10^7 строк, каждая из которых является числом от 0 до $10^{64} - 1$ и случайной строчкой-значением.

```
C:\Users\Кирилл\Desktop>g++ main.cpp -o slow.exe & g++ gen.cpp -o fast.exe
& g++ TEST.cpp -o TEST.exe & TEST.exe|slow.exe & TEST.exe|fast.exe
0.0300
0.0110
C:\Users\Кирилл\Desktop>g++ main.cpp -o slow.exe & g++ gen.cpp -o fast.exe
& g++ TEST.cpp -o TEST.exe & TEST.exe|slow.exe & TEST.exe|fast.exe
0.4140
0.0580
C:\Users\Кирилл\Desktop>g++ main.cpp -o slow.exe & g++ gen.cpp -o fast.exe
& g++ TEST.cpp -o TEST.exe & TEST.exe|slow.exe & TEST.exe|fast.exe
4.0700
0.4800
```

Как видно, поразрядная сортировка существенно быстрее, чем $std :: stable\_sort$, причем чем больше тестов обрабатывает программа, тем больше разрыв. Это происходит из-за того, что наша сортировка работает за O(n), тогда как скорость стандартной - O(n*log(n)).

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился пользоваться такими алгоритмами сортировки, как *radix sort* и сортировка подсчетом. Также я научился работать в условиях ограниченной доступной для программы памяти и времени работы, а значит, как следствие - оптимизировать алгоритм работы программы.

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] *Поразрядная сортировка — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Поразрядная_сортировка` (дата обращения: 05.11.2020).

[3] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).