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: П. Ф. Гришин
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

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` $2^{64} - 1$.

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

1 Описание

Опишу реализацию алгоритма поразрядной сортировки.

Основная идея поразрядной сортировки заключается в том, что сначала производится сортировка по младшему разряду, после по предпоследнему и т.д. до тех пор, пока входные данные не будут отсортированы по всем разрядам [1].

Как следует из описания, данная сортировка подходит для таких данных, в которых можно выделить поразрядную структуру - это не только числа, но и, например, строки. Чтобы сортировка получилась действительно линейной, в пределах одного разряда данные нужно сортировать так же с использованием сортировки линейной сложности - например, с помощью сортировки подсчётом. Основная её идея заключается в том, чтобы для каждого элемента x определить количество элементов, которые меньше x. С помощью этой информации можно разместить элемент x в нужной позиции выходной последовательности[1].

2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение». Ключ и значение будем хранить так: в векторе map будем хранить строки (значения), а в векторе $data$ - пары (ключ формата $unsigned\ long\ long$ и int). Это сделано для того, чтобы избежать копирования строк при сортировке. int будет использоваться, как индекс для получения конкретной строки.

После ввода значений непосредственно сортировка выполняется в цикле по всем разрядам (их коичество определяется максимальной длиной введённого ключа). В пределах разряда для сортировки элементов применяется сортировка подсчётом. Опишем этот процесс подробнее.

Сортировка по разряду n производится во время очередной итерации. В её пределах создаётся вектор векторов $count$ на 10 элементов и в каждую корзину записываются пары чисел. Пара < $\overline{x_n, x_{n-1}, ..., x_1, x_0}$, a_i> на k разряде, где $0 \leq k \leq n$, запишется в ячейку под номером x_k.Например, когда будем рассматривать разряд единиц, пара <134,5> запишется в корзину под номером 4.

После того, как все пары из вектора $count$ были распределены по корзинам, начинаем перезаписывать вектор с исходными данными.

Продолжаем проходится по разрядам чисел, пока не дотигнем максимального разряда n.

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int MaxDigit(unsigned long long maximum) {
7      int count = 0;
8      while (maximum > 0) {
9          maximum /= 10;
10         count++;
11     }
12     return count;
13 }
14
15 void RadixSort(vector<pair<unsigned long long, int>> &vec, unsigned long long maximum)
       {
16     int n = MaxDigit(maximum);
17     unsigned long long power = 1;
18     for (int i = 0; i < n; i++) {
19         vector<vector<pair<unsigned long long, int>>> count(10);
20         for (pair<unsigned long long, int> pair_var: vec) {
21             count[(pair_var.first / power) % 10].push_back(pair_var);
22         }
23         int k = 0;
```

```
        for (vector<pair<unsigned long long, int>> bucket: count) {
            for (pair<unsigned long long, int> pair_var: bucket) {
                vec[k++] = pair_var;
            }
        }
        power *= 10;
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    vector<pair<unsigned long long, int>> data;
    pair<unsigned long long, int> pair1;
    unsigned long long maximum = 0;
    vector<string> map;
    string line;
    int i = 0;
    while (cin >> pair1.first >> line) {
        maximum = max(maximum, pair1.first);
        map.push_back(line);
        pair1.second = i++;
        data.push_back(pair1);
    }
    RadixSort(data, maximum);
    for (pair<unsigned long long, int> pair_var: data) {
        cout << pair_var.first << "\t" << map[pair_var.second] << "\n";
    }
    return 0;
}
```

3 Консоль

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ make
Consolidate compiler generated dependencies of target Lab1
[ 50%] Building CXX object CMakeFiles/Lab1.dir/main.cpp.o
[100%] Linking CXX executable Lab1
[100%] Built target Lab1
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ cat test1.txt
4717633170819744722 xNCDE
9822309815890042053 npvThypBTRcQSxZorPvGEQHEUJWTWDaOLNRSdoyJlMwkbHlV
3952217684878621983 CQgXUYTVMUkIDAutECTJQ
3915403387641033293 ACsNzuYIJcfCpXxNnTIgEkKHjYSAOGcBBTmLmlkUflocr
13658203420764998726 YUGqciK
4963525392010493422 SIPoVdkBCzNF
6807515352472165033 lspGXYSPbJiyOSeR
14310008278242022627 IfWqIOQ
1473305144600239241 RCAtp
5334614247217967216 FlMNVO
13281804333106179212 ShrOZ
212708620671791667 YKBLAwqPS
16517187918995857731 FKtXwZGoOEDJ
6925710470920587974 hovGYMIXLmUGUWE
13623314078058231029 nDCYBNyaYLwmqRlUVb
12506355131614209192 UqIIbcNFXt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test1.txt
212708620671791667 YKBLAwqPS
1473305144600239241 RCAtp
3915403387641033293 ACsNzuYIJcfCpXxNnTIgEkKHjYSAOGcBBTmLmlkUflocr
3952217684878621983 CQgXUYTVMUkIDAutECTJQ
4717633170819744722 xNCDE
4963525392010493422 SIPoVdkBCzNF
5334614247217967216 FlMNVO
6807515352472165033 lspGXYSPbJiyOSeR
6925710470920587974 hovGYMIXLmUGUWE
9822309815890042053 npvThypBTRcQSxZorPvGEQHEUJWTWDaOLNRSdoyJlMwkbHlV
12506355131614209192 UqIIbcNFXt
13281804333106179212 ShrOZ
13623314078058231029 nDCYBNyaYLwmqRlUVb
13658203420764998726 YUGqciK
14310008278242022627 IfWqIOQ
```

```
16517187918995857731 FKtXwZGoOEDJ
Runtime: 0ms
```

4 Тесты

В целях экономии места, был убран вывод, что немного уменьшило время выполнения кода. Так как в программе нет ветвлений и программа сортирует правильно, то следует выполнить тесты на проверку времени выполнения.

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test.txt
100000 test.txt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test.txt
Runtime: 177ms
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test2.txt
1000000 test2.txt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test2.txt
Runtime: 1627ms
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test3.txt
10000000 test3.txt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test3.txt
Runtime: 15135ms
```

Из тестов можно сделать вывод, что программа работает достаточно быстро.

Проанализируем эффективность программы в "худшем случае". $O(n)$ - время на считывание ввода, где n - количество входных данных.

Теперь переходим к поразрядной сортировке. Время на копирование элементов в вектор $count$ - $O(n)$. Время на перезапись исходного вектора - $O(n)$. Повторять эти действия мы будем d раз, где d - количество разрядов максимального ключа. Поиск количества разрядов займет $O(d)$.

Соединяя всё вместе получим - $O(n+d+d(n+n)) = O(n+d+dn)$, где $O(dn)$ - время на поразрядную сортировку.

Теперь сравним эту поразрядную сортировку со $stable\_sort$ из стандартной библиотеки C++.

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test.txt
100000 test.txt
```

```
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test.txt
Runtime: 43ms
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test2.txt
1000000 test2.txt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test2.txt
Runtime: 481ms
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ wc -l
test3.txt
10000000 test3.txt
gpavel@gpavel-HP-Pavilion-Gaming-Laptop-17-cd1xxx:~/Desktop/DA1/Lab1$ ./Lab1
<test3.txt
Runtime: 5289ms
```

Как можно увидеть реализованная поразрядная сортировка работает медленнее, чем сортировка из стандартной библиотеки STL, хоть алгоритмическая сложность лучше, чем у $stable\_sort$ $O(n \log n)$. Возможно, это связано с тем, что поразрядная имеет довольно большое значение констант, что преводит к тому, что сортировка работает медленее, чем ожидалось.

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я познакомился с сортировками линейной сложности (сортировкой подсчётом и поразрядной сортировкой) и научился их реализовывать.

Довольно интересным для меня оказалось доаказательство того, что реализованная мной сортировка является линейной. Довольно долго я думал над упрощением программы, так как чекер отказывался принимать лабораторную работу. Оказалось, что копирование строк - это довольно затратная по времени операция, но с числами компьютер справляется быстро. Этот факт, помог мне ускорить программу.

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.))