Лабораторная работа № 1 по курсу дискретного анализа: сортировка за линейное время

Выполнил студент группы М8О-208Б-20 Примаченко Александр.

Условие

Кратко описывается задача:

1. Требуется написать программу на языке C++, которая сортирует входные данные, подающиеся пользователю парой “ключ-значение”.
2. Вариант задания: 8-2. Карманная сортировка. ключами служат числа от 0 до $2^{64} – 1$, значениями: строки произвольной длины. Ключ и значение при вводе разделены знаком табуляции.

Метод решения

Для хранения пар ключ-значения я использовал вектор пар uint64_t и string. В него записывались вводимые пары, затем производится проверка на пустоту – если вектор пуст, программа завершает свою работу, иначе начинается сортировка вектора. Обычно карманная сортировка использует в себе какую-либо другую – я выбрал сортировку вставкой.
Идея карманной сортировки заключается в следующем: исходные значения разбиваются на равные интервалы (их количество совпадает с количеством элементов массива), затем осуществляется проход по массиву, все элементы помещаются в «карманы», затем карманы сортируются «второй» сортировкой – как упоминалось выше – я для этого выбрал сортировку вставкой.
Сравниваются между собой элементы при сортировке вполне очевидным образом – по полю .first, то есть по ключу.

Описание программы

Вся программа содержится в единственном файле main.cpp. Для реализации карманной сортировки были написаны следующие вспомогательные процедуры:

```
void Swap(pair<uint64_t, string> &first, pair<uint64_t, string>
&second) {
```

```
    pair<uint64_t, string> tmp = first;
    first = second;
    second = tmp;
} - меняет местами два элемента вектора

void InsertionSort(vector<pair<uint64_t, string>> &array) {
    for (int i = 1; i < array.size(); i++)
        for (int j = i; j > 0; --j)
            if (array[j - 1].first > array[j].first) {
                Swap(array[j - 1], array[j]);
            }
} - сортировка вставкой
```

Сама карманная сортировка содержится здесь:

```
void BucketSort(vector<pair<uint64_t, string>> &array)
```

Первым делом необходимо создать вектор векторов, который будет хранить интервалы, в которые мы позже поместим все члены исходного вектора, а затем ищем минимальный и максимальный элементы:

```
vector<vector<pair<uint64_t, string>>> buckets(array.size());
uint64_t max_elem = array[0].first;
uint64_t min_elem = array[0].first;
for (const auto &p: array) {
    if (max_elem < p.first)
        max_elem = p.first;
    if (min_elem > p.first)
        min_elem = p.first;
}
```

Находим длину интервала:

```
const long double interval = (long double)(max_elem - min_elem + 1) /
array.size();
```

Размещаем элементы по интервалам:

```
for (int i = 0; i < array.size(); ++i) {
    auto a = (uint64_t)((array[i].first - min_elem) / interval);
    if (a == array.size())
        a--;
    buckets[a].push_back(array[i]);
}
```

Теперь, когда карманы заполнены, необходимо их отсортировать:

```
for (int i = 0; i < array.size(); ++i) {
    InsertionSort(buckets[i]);
}
```

Заполняем исходный вектор отсортированными значениями (последняя строка служила для отладки и необходимой не является):

```
  int k = 0;
    for (int i = 0; i < buckets.size(); ++i) {
        for (int j = 0; j < buckets[i].size(); ++j) {
            array[k++] = buckets[i][j];
        }
    }
    assert(k == array.size());
```

В main в начале создаётся вектор, в который и будут помещаться пары, а также переменные для чтения ключей и значений:

```
vector<pair<uint64_t, string>> array;
    uint64_t key;
    string value;
```

Затем происходит непосредственно считывание, и если хотя бы один элемент считался – запускается сортировка, отсортированный вектор выводится на экран:

```
while (cin >> key >> value) {
    ++count;
    array.emplace_back(make_pair(key, value));
}
unsigned int start_time = clock();
if(count > 0) {
    BucketSort(array);
}
for (const auto &p: array) {
    std::cout << p.first << "\t" << p.second << endl;
}
```

Исходный код:

```
#include <iostream>
#include <vector>

struct Pair {
    unsigned long long key{};
    char value[64]{};
};

void swap(Pair &first, Pair &second) {
    Pair tmp = first;
    first = second;
    second = tmp;
}

void insertionSort(std::vector<Pair> &vec) {
    for (int i = 1; i < vec.size(); i++)
        for (int j = i; j > 0; --j)
            if (vec[j - 1].key > vec[j].key)
                swap(vec[j - 1], vec[j]);
```

```
}

void bucketSort(std::vector<Pair> &vec) {
    std::vector<std::vector<Pair>> buckets(vec.size());
    unsigned long long maxKey = vec[0].key;
    unsigned long long minKey = vec[0].key;
    for (const auto &v: vec) {
        if (maxKey < v.key) maxKey = v.key;
        if (minKey > v.key) minKey = v.key;
    }
    const long double interval = (long double) (maxKey - minKey + 1) /
vec.size();
    for (int i = 0; i < vec.size(); ++i) {
        auto j = (unsigned long long) ((vec[i].key - minKey) /
interval);
        if (j == vec.size())
            j--;
        buckets[j].push_back(vec[i]);
    }
    for (int i = 0; i < vec.size(); ++i)
        insertionSort(buckets[i]);
    int position = 0;
    for (auto &bucket : buckets) {
        for (auto &pair : bucket) {
            vec[position++] = pair;
        }
    }
}

int main() {
    std::vector<Pair> vec;
    unsigned long long k;
    std::string v;
    while (std::cin >> k >> v) {
        Pair pair;
        pair.key = k;
        for (int i = 0; i < 64; i++) {
            pair.value[i] = v[i];
        }
        vec.push_back(pair);
    }
    if (!vec.empty()) {
        bucketSort(vec);
        for (Pair &p: vec)
            std::cout << p.key << '\t' << p.value << '\n';
    }
    return 0;
}
```

Дневник отладки

Первой ошибкой при отладке была ошибка компиляции, которая была исправлена с первого раза и быстро – программа была отправлена в некорректном виде.

Вторая ошибка заключалась в том, что при выводе я разделял ключ и значение пробелом, а не табуляцией, поэтому при фактически правильной сортировке ответ засчитывался за неверный.

Третья ошибка, на исправление которой было потрачено больше всего времени – Runtime Error. Как выяснил с шестого раза, проблема была в том, что программа вылетала при пустом вводе.

Тест производительности

1) n = 10, time = 2ms
2) n = 100, time = 14ms
3) n = 1000, time = 156ms
4) n = 10000, time = 1600ms

Если гарантируется, что ввод будет корректным – программа не имеет никаких недочётов. Но в противном случае она может отработать некорректно.

Выводы

Карманная сортировка хорошо работает для сортировки любого количества входных данных, главное, чтобы числа редко повторялись, так как если вероятность повторения высока – алгоритм может сильно деградировать – много элементов будет попадать в один карман, сложность может приблизиться к значению сложности сортировки вставкой O(N^2). В случае данной лабораторной работы эта вероятность высокой не является, поэтому сложность является линейной O(N).