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

Выполнил студент группы 08-208 МАИ *Каширин Кирилл.*

Условие

Кратко описывается задача:

1. Требуется разработать программу, осуществляющую ввод пар «ключ-значение», их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности.

2. Вариант задания: Сортировка подсчётом. Тип ключа: числа от 0 до 65535. Тип значения: строки переменной длины (до 2048 символов).

Метод решения

Для сортировки подсчетом создадим вспомогательный массив entries[0...k], который изначально заполним нулями (k = максимальный элемент исходного массива + 1). В вектор entries будет хранится количество элементов равных индексу массива entries. После этого посчитаем префиксную сумму в массиве entries. Каждый элемент вектора entries обозначает количество элементов меньших или равных индексу элемента. Чтобы алгоритм был устойчив, проходим с конца исходного массива, где значение исходного элемента равно индексу вектора entries. В каждой итерации цикла, декрементируем элемент на 1 в векторе entries (это будет являться индексом выходного массива) и кладём в найденный индекс значение элемента исходного массива. Декрементация позволяет правильно(не нарушая их начальный порядок) расположить элементы с одинаковым ключом. Проход исходного массива с конца позволяет использовать сортировку подсчетом не только для обычных чисел, но и для объектов.

Описание программы

Вся программа написана в файле main.cpp. В нем реализован класс TPair, который содержит в себе объект "ключ-значение функцию CountingSort, в которой реализован алгоритм сортировки подсчетом.

Исходный код

```
#include<iostream>
#include<vector>
#include<string>
#include<ctime>
```

```
class TPair {
    private:
        unsigned short localKey;
        std::string localValue;
    public:
        TPair(unsigned short key, std::string value) {
            localKey = key;
            localValue = value;
        }
        int GetKey() {
            return localKey;
        }
        std::string GetValue() {
            return localValue;
        }
};

std::vector<int> CountingSort(std::vector<TPair>& input, int max) {
    std::vector<int> entries(max+1,0);
    for (int i = 0; i < input.size(); ++i) {
        ++entries[input[i].GetKey()];
    }
    for (int i = 1; i < entries.size(); ++i) {
        entries[i] += entries[i-1];
    }
    std::vector<int> output(input.size());
    for (int i = input.size() - 1; i >= 0; --i) {
        --entries[input[i].GetKey()];
        output[entries[input[i].GetKey()]] = i;
    }
    return output;
 }

int main() {
    unsigned int start_time =  clock();
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::vector<TPair> input;
    unsigned short key;
    std::string value;
    int max = 0;
```

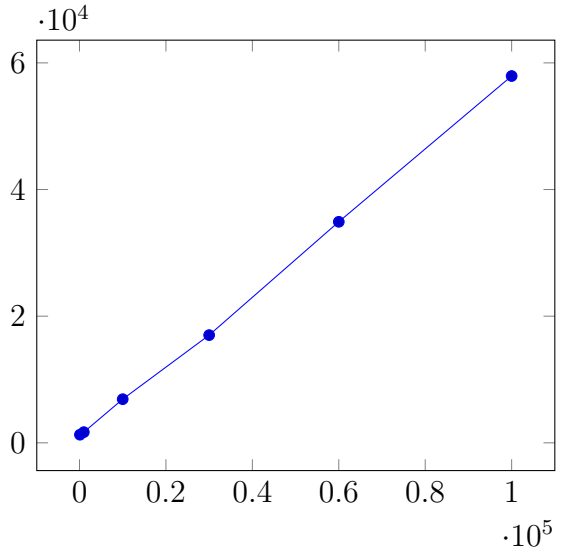
```
    while(std::cin >> key >> value) {
        input.push_back({key, value});
        if (key > max) {
            max = key;
        }
    }
    std::vector<int> output(input.size());
    output = CountingSort(input, max);
    for (int i = 0; i < output.size();i++) {
         std::cout << input[output[i]].GetKey() << '\t'
        << input[output[i]].GetValue() << "\n";
    }
}
```

Дневник отладки

Был превышен лимит при использовании памяти, поэтому, помимо обращение через указатели исходного массива в функции было принято решение в выходном массиве хранить не элемент, а индекс иходного массива, где хранился нужный элемент. При выводе, зная индекс, я обращался к исходному массиву и выводил элемент.

Тест производительности

Для теста производительности я создал шесть файлов, в которых сгенерировал 100, 1000, 10000, 30000, 60000 100000 входных данных.
Получились следующие результаты:
100 входных данных - 1282 ms,
1000 входных данных - 1688 ms,
10000 входных данных - 6893 ms,
30000 входных данных - 17010 ms,
60000 входных данных - 35904 ms,
100000 входных данных - 57924 ms
По результатам теста я построил график зависимости времени работы программы (ось y) от количества входных данных (ось x):



В результате, график получился линейный, а следовательно рост времени работы при увеличении объема входных данных согласуется с заявленной сложностью сортировки подсчетом - O(n+k).

Недочёты

Недостатком программы является ограниченность использования. На входных данных, заявленных на чекере (ключ от 0 до 65535), программа будет работать правильно, но если взять, например, входные данные с отрицательными значениями правильность работы достигаться не будет.

Выводы

В данной лабораторной работе мною был реализован алогоритм сортировки подсчётом. Эта сортировка часто применяется тогда, когда в последовательности есть множество повторяющихся значений. Также этот алгоритм целесообразно использовать тогда, когда в сортируемой последовательности диапазаон возможных значений мал по сравнению с количеством уникальных значений последовательности.

Одним из минусов данной сортировки является тот факт, что сортировать подсчётом в основном можно только целочисленные значения. Вещественные, строковые значения не получится отсортировать. Ещё одним недостатком данной сортировки является требование дополнительной памяти O(k), где k - максимальный элемент в исходной последовательности. Дополнительная память в некоторых случаях может быть большой. Именно поэтому область применения алгоритма сортировки подсчетом ограничена, поскольку неуместное использование этой сортировки может привести к нерациональному расходу памяти.