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. А. Земсков
Преподаватель: С. А. Михайлова
Группа: М8О-201Б

Дата: 22.03.2025
Оценка:
Подпись:

Москва, 2025

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: `Сортировка подсчётом.`

Вариант ключа: `Почтовые индексы.`

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

1 Описание

Основная идея сортировки подсчётом заключается в том, чтобы для каждого входного элемента x определить количество элементов, которые меньше x [1].

Сортировка подсчётом работает следующим образом:

1. Находим максимальный и минимальный элемент в массиве k
2. Создаём вспомогательный массив *count* размером $max - min + 1$, инициализированный нулями
3. Подсчитываем количество каждого элемента в исходном массиве
4. Вычисляем префиксные суммы массива *count*
5. Строим отсортированный массив, используя информацию о позициях из *count*

Ниже представлен псевдокод алгоритма стабильной сортировки подсчетом [2]:

```
function complexCountingSort(A: int[n], B: int[n]):
    for i = 0 to k - 1
        P[i] = 0;
    for i = 0 to length[A] - 1
        P[A[i].key] = P[A[i].key] + 1;
    carry = 0;
    for i = 0 to k - 1
        temporary = P[i];
        P[i] = carry;
        carry = carry + temporary;
    for i = 0 to length[A] - 1
        B[P[A[i].key]] = A[i];
        P[A[i].key] = P[A[i].key] + 1;
```

2 Исходный код

Сортировка подсчётом реализована для пар строк, где первый элемент пары интерпретируется как целочисленный ключ. Программа считывает входные данные построчно, разделяя каждую строку по символу табуляции на ключ и значение. Алгоритм определяет минимальный и максимальный ключи, создаёт массив подсчёта частот встречаемости каждого ключа, вычисляет префиксные суммы для определения позиций элементов, затем строит отсортированный массив, сохраняя стабильность сортировки через обратный проход по исходным данным. Результат выводится в том же формате , сохраняя исходные строковые представления ключей. Особенность реализации - обработка строковых ключей с их временным преобразованием в числа для сортировки.

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

using namespace std;

void countSort(vector<pair<string, string>> &v)
{
  if (v.size() == 0)
    return;

  int min = stoi(v[0].first);
  int max = stoi(v[0].first);

  for (pair<string, string> &p : v)
  {
    int key = stoi(p.first);
    if (key < min)
      min = key;
    if (key > max)
      max = key;
  }

  vector<int> count(max - min + 1, 0);

  for (pair<string, string> &p : v)
  {
    int key = stoi(p.first);
    count[key - min]++;
  }

  for (int i = 1; i < count.size(); i++)
  {
    count[i] += count[i - 1];
```

```
36    }
37
38    vector<pair<string, string>> res(v.size());
39
40    for (int i = v.size() - 1; i >= 0; i--)
41    {
42      int key = stoi(v[i].first);
43      count[key - min]--;
44      res[count[key - min]] = v[i];
45    }
46
47    v = res;
48  }
49
50  int main()
51  {
52    vector<pair<string, string>> v;
53    string line;
54
55    while (getline(cin, line))
56    {
57      size_t tabPos = line.find('\t');
58      if (tabPos == string::npos)
59        continue;
60
61      string key = line.substr(0, tabPos);
62      string value = line.substr(tabPos + 1);
63
64      v.push_back({key, value});
65    }
66
67    countSort(v);
68
69    for (pair<string, string> &p : v)
70    {
71      cout << p.first << "\t" << p.second << endl;
72    }
73
74    return 0;
75  }
```

3 Консоль

```
g++ -std=c++20 -o main main.cpp
./main
000000 xGfxrxGGxrxMMMMfrrrG
999999 xGfxrxGGxrxMMMMfrrr
000000 xGfxrxGGxrxMMMMfrr
999999 xGfxrxGGxrxMMMMfr

000000 xGfxrxGGxrxMMMMfrrrG
000000 xGfxrxGGxrxMMMMfrr
999999 xGfxrxGGxrxMMMMfrrr
999999 xGfxrxGGxrxMMMMfr
```

4 Тест производительности

Тестирование проводилось на 4×10^6 элементов со строковыми ключами от "000000"до "999999". Сортировка подсчётом выполнилась за 1323 мс, стандартная std::sort — за 8364 мс. Разница объясняется фундаментальным различием в сложности алгоритмов. Сортировка подсчётом имеет линейную сложность O(n + k), где n — количество элементов 4×10^6, k — диапазон ключей 10^6. Стандартная сортировка основана на сравнениях и имеет сложность O(n log n).

Преимущество сортировки подсчётом проявляется именно при ограниченном диапазоне ключей, когда k не слишком велико относительно n. Алгоритм эффективно использует информацию о структуре данных, подсчитывая количество каждого ключа. Однако он требует O(k) дополнительной памяти для хранения массива частот. Стандартная сортировка, будучи универсальным алгоритмом, не использует специфику данных и потому менее эффективен в данном случае, но сохраняет преимущество при работе с произвольными ключами или когда диапазон значений слишком велик. Разница в 6.3 раза соответствует теоретическим ожиданиям для данных параметров.

```
zmskv@MacBook-Air-Pavel benchmark % make test
Testing count sort...
time: 1323.02ms
Testing std::sort...
time: 8364.7ms
```

5 Выводы

В ходе лабораторной работы я узнал, что сортировка подсчётом — это алгоритм, эффективный для данных с ограниченным диапазоном ключей. На практике я научился адаптировать её не только для числовых значений, но и для строковых ключей, таких как почтовые индексы, преобразуя их в числовой формат. Особенно полезным оказалось понимание того, как выбор алгоритма влияет на производительность: для 4×10^6 элементов сортировка подсчётом оказалась в 6 раз быстрее стандартной std::sort. Также я освоил методику объективного сравнения алгоритмов через замеры времени выполнения. Эти знания важны для оптимизации обработки больших данных, где критична скорость работы. Главный вывод — эффективность алгоритма сильно зависит от специфики данных, и иногда кастомные реализации превосходят стандартные решения.

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] *Сортировка подсчётом — ITMO Wiki.*
URL: `https://neerc.ifmo.ru/wiki/`