Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: И. С. Глушатов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

Задача: Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Дана последовательность длины N из целых чисел 1, 2, 3. Необходимо найти минимальное количество обменов элементов последовательности, в результате которых последовательность стала бы отсортированной.

1 Описание

Жадные алгоритмы - алгоритмы, предполагающие принятие локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. В общем случае жадные алгоритмы могут не находить глобального оптимума, однако в некоторых задачах позволяют это сделать.

Для моей задачи алгоритм выглядит так:

Во время ввода масива считаем количество единичек, двоек и троек. Таким образом мы можем разбить мысленно массив на три блока, где в первом должны находиться только единицы, во втором только двойки, в третьем - тройки. Причем длина первого блока будет равняться количеству единичек и т.д.

Далее мы проходимся по массиву слева направо. Если мы в первом блоке встречаем единицу, то идем дальше, если встречаем двойку, значит существует единица, которая не на своем месте. Пытаемся ее найти во втором блоке, так как в таком случае при обмене сразу два элемента встанут на свои места и нам понадобиться сделать лишь одно действие. Если же во втором блоке не нашлось единицы, то ищем в третьем блоке, где она точно есть. Если в первом блоке мы встретили тройку, то шаги аналогичные, только сначала пытаемся найти единицу в третьем блоке, а потом во втором блоке.

Когда мы оказываемся во втором блоке, все единицы уже на своем месте, следовательно, встретив тройку, достаточно лишь найти двойку в третьем блоке и обменять. На самом деле можно в этом случае не обменивать (и не искать) тройку и двойку, а просто посчитать количество троек во втором блоке и прибывить к ответу, однако я придержался концепции, что в итоге массив должен быть тоже отсортирован, хотя этого и не требуется.

Сложность такого алгоритма $O(n^2)$, так как нам надо пройтись по всем числам из первого и второго блоков, которых, если считать, что количество элементов равное, $\frac{2n}{3}$ и для каждого элемента, стоящего не на своем месте за O(n) искать.

2 Исходный код

```
int main() {

  size_t n = 0;
  cin >> n;

  short* numbers = new short[n];
  size_t one_two_three[3] = {0, 0, 0};

  for (size_t i = 0; i < n; i++) {
    cin >> numbers[i];
    one_two_three[numbers[i]-1]++;
  }

  size_t result = 0;
  for (size_t i = 0; i < n; i++) {

    if (i < one_two_three[0]) {

      if (numbers[i] == 2) {
        for (size_t j = one_two_three[0]; j < n; j++) {
          if (numbers[j] == 1) {
            numbers[i] = 1;
            numbers[j] = 2;
            result++;
            break;
          }
        }
      } else if (numbers[i] == 3) {
        for (size_t j = one_two_three[0] + one_two_three[1]; j < n; j++) {
          if (numbers[j] == 1) {
            numbers[i] = 1;
            numbers[j] = 3;
            result++;
            break;
          }
        }

        if (numbers[i] == 3) {
          for (size_t j = one_two_three[0]; j < one_two_three[0] + one_two_three[1]; j
              ++) {
            if (numbers[j] == 1) {
              numbers[i] = 1;
              numbers[j] = 3;
              result++;
              break;
            }
          }
        }
      }

    } else if (i < one_two_three[0] + one_two_three[1]) {

      if (numbers[i] == 3) {
        for (size_t j = one_two_three[0] + one_two_three[1]; j < n; j++) {
          if (numbers[j] == 2) {
            numbers[i] = 2;
            numbers[j] = 3;
            result++;
            break;
```

```
59          }
60        }
61      }
62
63    }
64
65  }
66
67  cout << result << endl;
68
69  delete[] numbers;
70  return 0;
71 }
```

3 Консоль

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab8$ cat tests/test1.txt
141
1 1 3 2 3 3 2 1 2 2 2 3 1 1 1 1 3 1 3 1 2 2 1 3 3 3 2 1 2 2 1 3 1 3 2 2 3 1
1 2 2 1 2 3 3 1 1 1 2 1 1 2 2 2 2 3 2 3 3 3 2 2 1 2 3 2 2 2 1 3 1 2 1 2 3 1
1 1 1 1 2 3 3 1 1 3 3 1 1 3 3 3 3 2 3 2 2 1 3 1 2 2 1 2 3 1 2 3 2 1 3 1 3 2
2 2 2 1 1 2 2 3 3 2 1 2 1 2 2 2 2 3 1 1 1 1 3 3 1 1 2
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab8$ ./main.out <tests/test1.txt
47
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab8$ ./main.out
3
3 2 1
1
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab8$
```

4 Тест производительности

Для тестов я использвал утилиту gnuplot для построения графиков зависимости времени работы программы от количества чисел. Так же для сравнения использовал библиотеку chrono для замера времени.

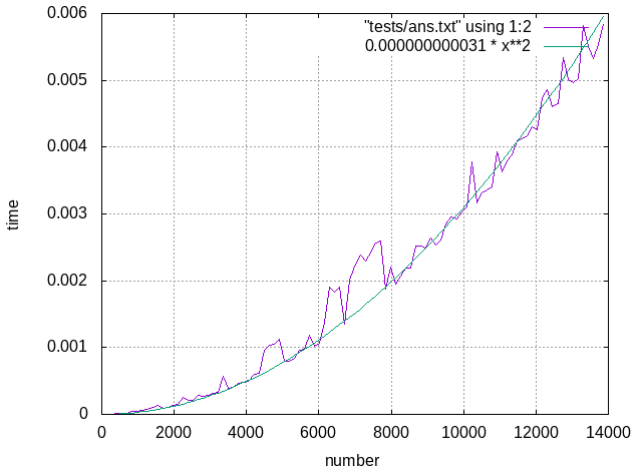

Рис. 1: Графики работы жадного алгоритма и сопоставление с квадратичной функцией

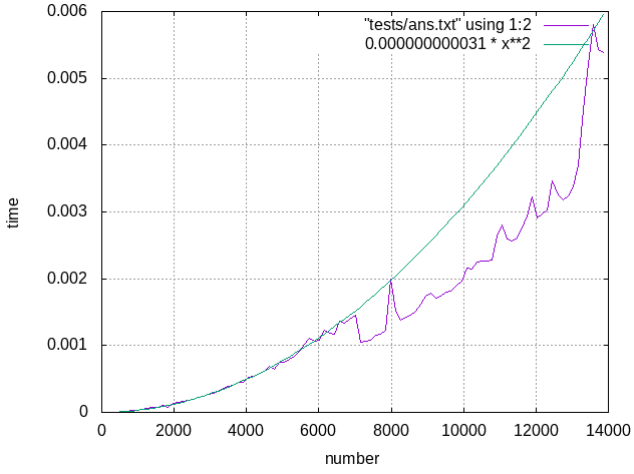

Рис. 2: Графики работы жадного алгоритма без обмена во втором блоке и сопоставление с квадратичной функцией из прошлого примера

Как видно сложность алгоритмов одинаковая и в худшем случае второй приближается к первому.

5 Выводы

В ходе восьмой лабораторной работы я познакомился с жадными алгоритмами. В процессе выполнения задания особых проблем не было.

Список литературы

[1] *Поисковик - Google.*
URL: `https://www.google.com/`

[2] *Сайт с подробной документацией библиотек C++*
URL: `https://en.cppreference.com/`