Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: Ф. А. Иванов
Преподаватель: И. Н. Симахин
Группа: М8О-208Б-19

Дата: 21.04.2022
Оценка:
Подпись:

Москва, 2022

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.

Формат входных данных: Число N на первой строке и N чисел на второй строке.
Формат результата: Минимальное количество обменов.

1 Описание

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.

В данной задаче нам нужно, применяя жадный алгоритм, найти минимальное количество обменов элементов последовательности, в результате которых последовательность стала бы отсортированной.

В условии указано, что в последовательности всего 3 числа: 1, 2, 3. Соответственно, это наталкивает на мысль, разбить последовательность на 3 блока: блок единиц, блок двоек, блок троек.

При считывании последовательности я сразу подсчитываю количество единиц, двоек и троек. Теперь, поэлементно просматриваем последовательность и опеределяем, если в блоке единиц присутствует инородное число (двойка или тройка), то применим следующую методологию поиска единицы в других блоках.

Если встретилась двойка в блоке единиц: идем в область двоек и начинаем просматривать до конца последовательности, пока не встретим единицу, определяем позицию, меняю местами и продолжаю далее. У меня может возникнуть ситуация когда при обмене я поставлю двойку в область троек, но она заменется на последующем этапе.

Если встретилась тройка в блоке единиц: идем в конец последовательности и просматриваем ее в обратном порядке на поиск единицы, определяем позицию и меняем местами. Аналогичная ситуация может и возникнуть здесь, что при обмене цифра окажется не в своем блоке. Эта проблема решается на последующем этапе.

Когда все единицы отсортированы, мы переходим в облась двоек и это являемтся конечным этапом, т.к. тройки уже будут отсортированы после прохода этого этапа, а единицы были отсортированы на предыдущем этапе. Посик осуществляется по аналогии с предыдущем этапом. Если встретилась тройка, .то идем в область троек и ищем там двойку, меняем.

Сложность алгоритма $O(k*n)$,где k $<$ n. Сложность можно апроксимировать до $O(n)$.

2 Исходный код

```
#include <iostream>
#include <vector>
using namespace std;
long getPositionSwap(long first, long currentPosition, vector<long> &digits)
{
    if (digits.empty() || currentPosition < 0 || currentPosition >= digits.size()){
        return 0;
    }

    while (digits[currentPosition] != first && currentPosition + 1 < digits.size()) {
        ++currentPosition;
    }
    return currentPosition;
}
long getPosisionSwapReverse(long first, long currentPosition, vector<long> &digits) {
    if (digits.empty() || currentPosition < 0 || currentPosition >= digits.size()){
        return 0;
    }

    while (digits[currentPosition] != first && currentPosition - 1 > -1) {
        --currentPosition;
    }
    return currentPosition;
}
int main() {
    const int COUNT_NUMBERS = 3;
    vector<long> count(COUNT_NUMBERS);
    long n, answer = 0;
    cin >> n;
    vector<long> digits(n);

    for (long i = 0; i < n; ++i) {
        cin >> digits[i];
        ++(count[digits[i]- 1]);
    }
    long position_1_in_2 = getPositionSwap(1,count[0], digits);
    long position_1_in_3 = getPosisionSwapReverse(1, n - 1, digits);
    long position_2_in_3 = getPositionSwap(2, count[0] + count[1], digits);
    for (long i = 0; i < n; ++i) {
        if (i < count[0] && digits[i] != 1){
            if (digits[i] == 2) {
                ++answer;
                swap(digits[i], digits[position_1_in_2]);
                position_1_in_2 = getPositionSwap(1,count[0], digits);
            } else {
                ++answer;
                swap(digits[i], digits[position_1_in_3]);
```

```
48                position_1_in_3 = getPosisionSwapReverse(1, n - 1, digits);
49            }
50        } else if (i >= count[0] && i < count[0] + count[1] && digits[i] != 2) {
51            ++answer;
52            swap(digits[i], digits[position_2_in_3]);
53            position_2_in_3 = getPositionSwap(2, count[0] + count[1], digits);
54        }
55    }
56    cout << answer << endl;
57    return 0;
58 }
```

3 Консоль

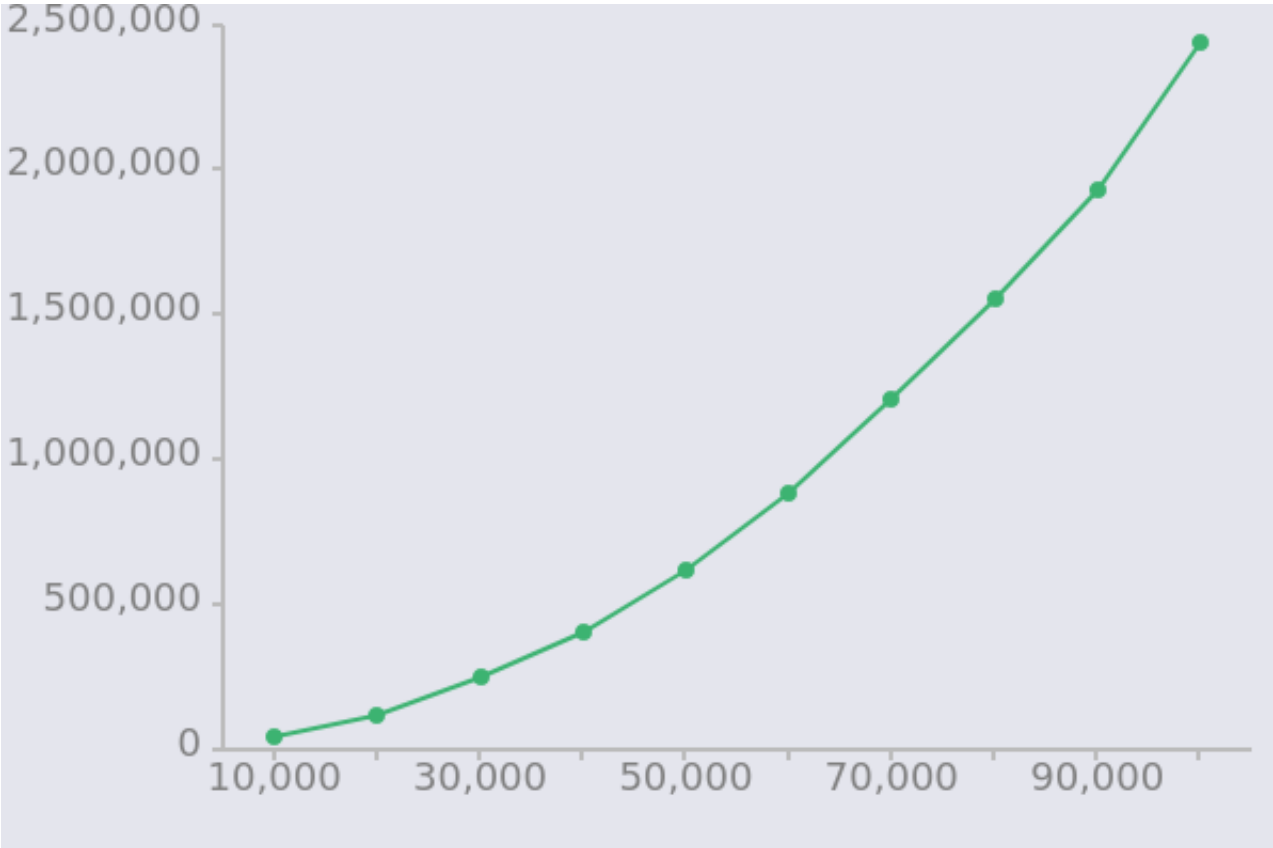
```
orion@orion-laptop:~/University/DA/Lab_08$ g++ main.cpp -o main
orion@orion-laptop:~/University/DA/Lab_08$ ./main
3
3 2 1
1
orion@orion-laptop:~/University/DA/Lab_08$ ./main
9
2 1 2 1 3 2 2 3 1
3
orion@orion-laptop:~/University/DA/Lab_08$ ./main
15
3 3 3 3 1 3 2 2 1 2 1 3 3 2 1
6
orion@orion-laptop:~/University/DA/Lab_08$
```

4 Тест производительности

Для докозательства приближенной линейной сложности алгоритма построим график зависимости времени работы от длины строки.

```
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 10000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 47087
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 20000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 115321
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 30000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 250996
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 40000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 404096
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 50000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 620638
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 60000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 883863
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 70000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 1209271
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 80000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 1558519
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 90000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
```

```
Время работы: 1933921
orion@orion-laptop:~/University/DA/Lab_08$ python3 creater.py
Количество символов строки: 100000
orion@orion-laptop:~/University/DA/Lab_08$ ./main <test
Время работы: 2442291
orion@orion-laptop:~/University/DA/Lab_08$
```



5 Выводы

Выполнив данную лабораторную работу, я познакомился с жадными алгоритмами. Это довольно простой и единственный метод решения задач на оптимизацию, который может быть полезен там, где не справляется динамической программирование. Жадные алгоритмы часто используются на практике. Однако, если приходится работать с огромными данными, то вычислительнгой мощности может не хватать. В этом случае применяются приближенные жадные алгоритмы, которые работают гораздо быстрее, но дают приблизительный ответ вместо точного.

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.))