Лабораторная работа № 8 по курсу: Дискретный анализ

Выполнил студент группы М8О-30ХБ-17 МАИ *Лопатин Александр.*

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++, соответствующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Вариант 5: Оптимальная сортировка чисел.
Дана последовательность длины N из целых чисел 1, 2, 3. Необходимо найти минимальное количество обменов элементов последовательности, в результате которых последовательность стала бы отсортированной.

Входные данные: число N на первой строке и N чисел на второй строке.

Выходные данные: минимальное количество обменов.

Информация

Суть жадных алгоритмов состоит в принятии локально-оптимальных решений в надежде, что конечное решение так же окажется оптимальным. Но не всегда применим жадный алгоритм, чтобы доказать корректность его применения, нужно доказать что исследуемый объект является матроидом.

Матроидом называется пара множеств E, I, состоящая из конечного множества E, называемого базовым множеством матроида, и множества его подмножеств I, называемого множеством независимых множеств матроида, причем I удовлетворяет свойствам:

- Множество I не пусто. Даже если исходное множество E было пусто – $E = 0$, то I будет состоять из одного элемента - множества, содержащего пустое $I = \{\{0\}\}$.
- Любое подмножество любого элемента множества I также будет элементом этого множества.
- Если $X, Y \in I$, причем $|X| = |Y| + 1$, тогда существует элемент $x \in X \backslash Y$, такой что $Y \cup \{x\} \in I$.

Этапы построения алгоритма решения подзадач:

- Описать структуру оптимального решения.
- Составить рекурсивное решение для нахождения оптимального решения.

- Вычисление значения, соответствующего оптимальному решению, методом восходящего анализа.
- Непосредственное нахождение оптимального решения из полученной на предыдущих этапах информации.

Метод решения

В результате сортировки наша последовательность примет вид: $1, 1, ..., 2, 2, ..., 3, 3, ...$, причем вычислить индексы начала каждого из блока единиц, двоек или троек можно подсчетом единиц, двоек и троек. Разделим неотсортированную последовательность на такие же блоки. Заметим, что если в блоке, соответсвующем i, есть элементы i, то менять их не нужно, они уже стоят на своем месте. Если же в блоке i есть элемент j, не равный i, то находим в блоке j элемент i и меняем их. В результате данного обмена мы стали ближе к отсортированной последовательности, так как два элемента встали на свои места, причем мы сделали наименьшее возможное количество обменов. Если же в блоке j нет элемента i, то это значит, что элемент i есть в блоке k, не равном i и j. Меняем их местами. Встал на свое место только один элемент, чтобы встали на свое места элементы k и j, нужен еще один обмен. За два обмена встали на свои места 3 элемента. Однако в данном случае одного обмена никак не хватит, пример: 2 3 1. Таким образом, алгоритм принимает на каждом шаге локально-оптимальное решение, в надежде на то, что конечное решение будет оптимальным, и оно, как доказано выше по принципу жадного выбора, будет оптимальным.

Исходный код

lab8.cpp

```
#include <iostream>
#include <fstream>
using namespace std;

int main(int argc, char *argv[]) {
    ifstream inFile(argv[1]);
    int indices[] = {0, 0, 0, 0};
    inFile >> indices[3];
    int* arr = new int[indices[3]];
    for( int i = 0; i < indices[3]; i++) {
        inFile >> arr[i];
        indices[arr[i]-1]++;
    }
    inFile.close();
    indices[2] = indices[1] + indices[0];
    indices[1] = indices[0];
    indices[0] = 0;
    int res = 0;
    for( int i = 0; i < indices[3]; i++) {
```

```
        if(i < indices[arr[i]-1] || i >= indices[arr[i]]) {
            int lastNotSorted = -1;
            for ( int j = indices[arr[i]-1]; j < indices[arr[i]]; j++ ) {
                if(j < indices[arr[j]-1] || j >= indices[arr[j]]) {
                    lastNotSorted = j;
                }
                if(i >= indices[arr[j]-1] && i < indices[arr[j]]) {
                    int tmp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = tmp;
                    res++;
                    break;
                }
            }
            if(i < indices[arr[i]-1] || i >= indices[arr[i]]) {
                int tmp = arr[i];
                arr[i] = arr[lastNotSorted];
                arr[lastNotSorted] = tmp;
                res++;
            }
        }
    }
    ofstream outFile(argv[2]);
    outFile << res;
    outFile.close();
    delete[] arr;
}
```

Из-за наличия вложенных циклов, сложность по времени оценивается $O(n^2)$

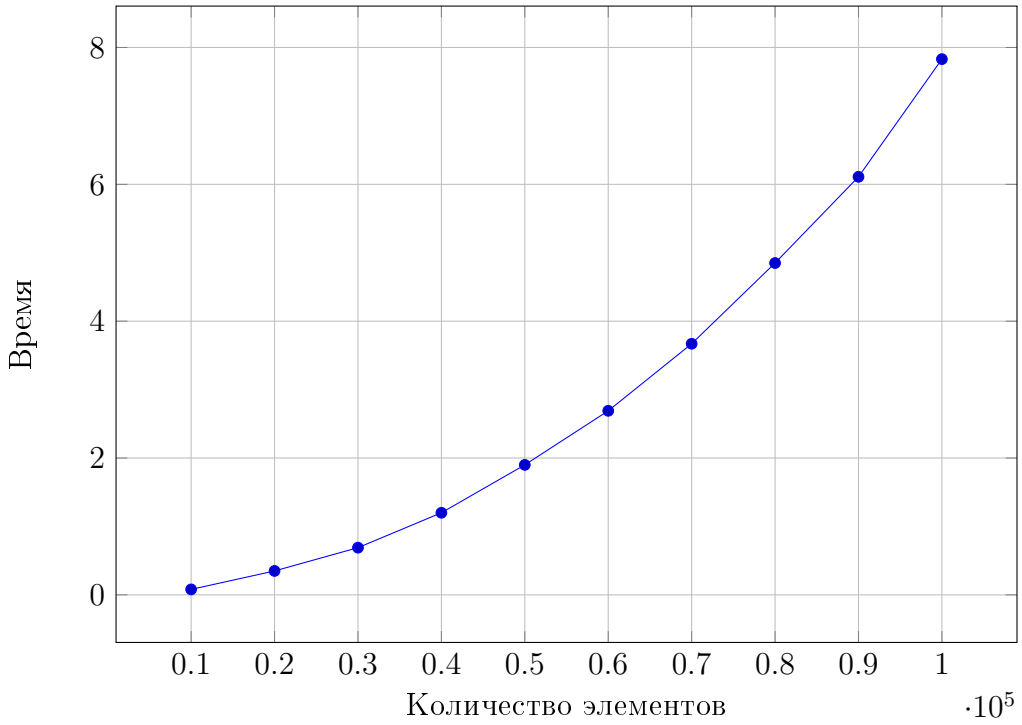
Генератор тестов

tests.py

```
import random

for i in range(10000, 110000, 10000):
    file = open('tests/test'+str(i)+'.txt','w')
    file.write(str((i))+'\n')
    for j in range(1, i+1):
        file.write(str((random.randint(1, 3)))+' ')
    file.close()
```

Тест производительности



По графику видно, что сложность близка к квадратичной.

Выводы

Жадный подход далеко не всегда дает оптимальное решение, но во многих случаях получаемое решение оказывается достаточно близким к оптимальному, а для некоторых задач известны и оптимальные жадные алгоримы (построение оптимального префиксного кода по Хаффману, построение остовного дерева максимального веса, планирование вычислений на процессоре с минимизацией ожидания и другие).