***Лабораторная работа 4.***

***Построение кодов Хаффмана***

***Задание.***

1. Дан текстовый файл размером не менее *5 кбайт*. Построить для данного текста *коды Хаффмана* (см. матермал лекции). Написать программу для кодировки и раскодировки заданного файла. Указать размеры файла до и после сжатия алгоритмом *Хаффмана*.   
   Для наглядности текстовый файл можно создать самостоятельно и меньшего размера.
2. Письменно подробно ответить на вопросы.
3. Решить задачу [**3. Longest Substring Without Repeating Characters**](https://leetcode.com/problems/longest-substring-without-repeating-characters/) на ресурсе *LeetCode*. Описание задачи смотри ниже.

***Вопросы.***

1. Что такое *сжатие без потерь*?
2. Опишите алгоритм построения кодов *Шеннона-Фано* для сжатия данных.
3. Что такое *префиксные коды*, являются ли в данных алгоритмах коды *префиксными* и для чего они используются?
4. Благодаря каким принципам происходит *сжатие* данных в указанных алгоритмах?
5. Укажите недостатки указанных кодов, средние коэффициенты сжатия для указанных алгоритмов.

***Метод скользящего окна***

В методе *скользящего окна* создается окно на некоторой коллекции, которое удовлетворяет ограничениям задачи. Окно нестатично. В случае нарушений ограничения задачи, оно начинает стабилизироваться, чтобы удовлетворять условиям задачи, увеличивая или уменьшая свой размер.

Есть много задач, решаемых *методом скользящего окна*. Рассмотрим одну из них.

[**3. Longest Substring Without Repeating Characters**](https://leetcode.com/problems/longest-substring-without-repeating-characters/)

Given a string s, find the length of the **longest** **substring** without repeating characters.

Требуется найти длину наибольшей подстроки, не содержащей повторяющиеся символы.

*Идея алгоритма*

В нашей задаче *скользящее окно* будет содержать подстроку с неповторяющимися (уникальными) символами. Окно имеет *3* параметра: левую границу окна – *left,* правую границу окна – *right* и ширина окна – *temp = right- left+1*.

Для учета всех встречающихся символов создается *булев* массив (можно использовать другую структуру данных, например множество) размером *128* *,* где *индексу массива ссответствует код символа по таблице ASCII.* До построения окна все значения элементов массива инициализируем *false.*

Строим окно, поместив границы окна на начало строки, *temp = 1.* Значение ячейки булевого массива, соответствующей первому символу меняем на *true.* Расширяем окно, увеличивая каждый раз *right* на *1 и записываем присутствие символа, на который указывает right, в булев массив,* пока не встретим символ, уже имеющийся в окне ( значение соответствующей ячейки булева мвссива уже равно *true*).

В этом случае переносим левую границу (на каждом шаге на *1* и меняя значение проходимых символов с *true* на *false* в булевом массиве) вправо до тех пор, пока значение символа, на который указывает *right,* не станет *false* в булевом массиве.

Ширина окна у нас уменьшилась и оно опять содержит уникальные символы. Далее опять расширяем окно вправо.

Рассмотрим движение окна на примере. Пусть нам дана строка *Abdcdefm.*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

*Abdcdefm*

Опишем теперь алгоритм решения задачи и приведем реализацию на *java:*

*1. проверяем вначале, чтобы строка не была пустой, ecли строка пустая, то возвращаем 0.*

*2. создаем булев массив “set” размером 128*

*3. устанавливаем левый указатель на окно left в 0.*

*4. устанавливаем правый указатель на окно right в 1.*

*5. заносим в set true на позицую первого символа строки*

*6. устанавливаем max (максимальная ширина окна, удовлетворяющая условию задачи) в 1.*

*И далее в цикле, пока правый указатель на окно не достигнет конца строки:*

*7. проверяем находиться ли символ, на который указывает right в set, и если да, то в set устанавливаем значение в false по индексу символа на который указывает левый указатель на окно, инкременируем left и повторям пункт 7, если же такого символа в set нет, то переходим к следующему пункту 8.*

*8. устанавливаем в set значение в true по индексу символа, на который указывает right.*

*9. вычисляем размер окна и сравниваем результат с max, и если результат вычисления больше то меняем max на этот результат.*

*10. инкрементируем right и начинаем цикл снова в пункте 7.  
По окончанию цикла max будет содержать максимальный ответ.*

class Solution {

public int lengthOfLongestSubstring(String s) {

if(s.length() == 0) {

return 0;

}

boolean[] set = new boolean[128];

int left = 0;

int right = 1;

set[s.charAt(left)] = true;

int max = 1;

while(right < s.length()) {

if(set[s.charAt(right)]) {

set[s.charAt(left)] = false;

left++;

continue;

}

set[s.charAt(right)] = true;

int temp = right - left + 1;

if(max < temp) {

max = temp;

}

right++;

}

return max;

}

}

Runtime

Details

**2**ms

Beats 99.56%of users with Java

Memory

Details

**42.19**MB

Beats 93.00%of users with Java

***Вопросы по обработке алгоритма***

1. В каких еще задачах применяется этот алгоритм?

2. Чему равна временная сложность алгоритма? Чему равна временная сложность при решении задачи *«в лоб»?*

3. Чему равна емкостная сложность алгоритма?