**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Отчет**

По индивидуальным задачам по теме «Рекуррентные соотношения»

Студентки 2 курса 9 группы

Шелег Владиславы Михайловны

**Преподаватель:**

Пилипчук Людмила Андреевна

Минск

2017

**Задача 54. Строка**

**1.Условие задачи**

Дана строка *S*, состоящая из *n* маленьких латинских букв (1 ≤ *n* ≤ 300). За один ход Вам разрешается удалить один или несколько подряд идущих одинаковых символов. Необходимо удалить все символы из строки *S* за минимальное число ходов.

**Формат входного файла**

Первая и единственная строка файла содержит исходную строку *S*.

**Формат выходного файла**

Выведите минимальное число ходов.

**2.Описание алгоритма**

Так как нам надо минимизировать количество удалений, то если в строке в подряд идут несколько одинаковых символов то если их и удалять, то только все вместе, исходя из этого, если в исходной строке идут в подряд несколько одинаковых символов, то можно их считать за один символ.

Удаление символа S[i] из строки назовем «присоединенным» к символу S[j], ели есть такой символ S[j] равный S[i](i≠j), что каким-то образом удалив все символы между позициями i и j(не включая сами i и j) мы «присоединяем» символ S[i] к символу S[j], то есть будем считать за один символ, ведь они равны.

Удаление символа S[i] из строки назовем «обособленным», если мы просто удаляем символ, вместе с «присоединенными» символами.

Получим, наша задачи свелась к минимизации «обособленных» удалений.

Рассмотрим произвольную строку P длины L. Легко заметить, что самый первый символ в строке P всегда можно удалить в последнюю очередь, то есть это не отразится на общем количестве удалений. Это следует из того, что при удалении какого-то не крайнего(то есть не первого и не последнего) символа наш строка как бы «сдвигается» и это может приводить к присоединенным удалениям символов, а при удалении первого или последнего символа ничего не «сдвигается», поэтому их всегда можно удалить в самую последнюю очередь, естественно вместе с «присоединенными» символами.

Пусть F[i,j] – минимальное количество «обособленных» удалений, для того чтобы удалить все символы из подстроки S[i,j]. Предположим, что функция F уже высчитана для всех подстрок длины менее j-i.Как мы показали выше первый символ будем удалять последним. Если мы хотим «обособленно» удалить первый символ, то для начала удалим все символы подстроки S[i+1,j] а потом и символ S[i], то есть F[i,j] = F[i+1,j]+1. Теперь рассмотрим «присоединенное» удаление: Возьмем такое t, что i<t≤j, S[i] = S[t]. Тогда для того чтобы S[t] присоединить к S[i] необходимо первоначально удалить все символы подстроки S[i+1,t-1], после это считаем S[i] и S[t] за один символ и удаляем все символы из подстроки S[t,j], то есть F[i,j] = F[i+1,t-1] + F[t,j]. Исходя из всего вышеизложенного верна рекуррентная формула



**3.Примеры**

Строка:acdcbbc

Заполнение матрицы для данного примера происходит в следующем порядке:

1 шаг

1 2 0 0 0 0

0 1 2 0 0 0

0 0 1 2 0 0

0 0 0 1 2 0

0 0 0 0 1 2

0 0 0 0 0 1

2 шаг

1 2 3 0 0 0

0 1 2 2 0 0

0 0 1 2 3 0

0 0 0 1 2 2

0 0 0 0 1 2

0 0 0 0 0 1

3 шаг

1 2 3 3 0 0

0 1 2 2 3 0

0 0 1 2 3 3

0 0 0 1 2 2

0 0 0 0 1 2

0 0 0 0 0 1

4 шаг

1 2 3 3 4 0

0 1 2 2 3 3

0 0 1 2 3 3

0 0 0 1 2 2

0 0 0 0 1 2

0 0 0 0 0 1

5 шаг

1 2 3 3 4 4

0 1 2 2 3 3

0 0 1 2 3 3

0 0 0 1 2 2

0 0 0 0 1 2

0 0 0 0 0 1

Ответ: 4.

**Задача 7. Строго возрастающая почти всюду подпоследовательность**

**1.Условие задачи**

Необходимо из заданной числовой последовательности *A* длины *n*, вычеркнуть минимальное число элементов, чтобы оставшиеся элементы образовали *строго возрастающую почти всюду* подпоследовательность, т. е.  содержащую не более одного разрыва – пары ( подряд идущих элементов, второй из которых не больше первого. Построенный алгоритм должен иметь трудоемкость .

**Формат входного файла**

Первая строка содержит число *n* (1 ≤ *n* ≤ 100 000). Следующая строка содержит *n* элементов последовательности *A*, которые разделены пробелом (элементы последовательности — целые положительные числа, не превосходящие 1 000 000 000).

**Формат выходного файла**

Выведите длину строго возрастающей почти всюду подпоследовательности элементов.

**2. Описание алгоритма**

Будем хранить два массива: массив *B*[*n*] , где *i* -тым элементом является длина максимальной подпоследовательности, где последним элементом является *A*[*i*] ; и массив *C*[*n*] , где *i* -тым элементом является индекс последнего элемента в максимальной подпоследовательности длины (*i*+1) . Начальные значения:

*B*[0]=1 , *C*[0]=0 .

Будем заполнять массив *C* . Для этого для каждого элемента *A*[*i*] будем

выполнять следующее:

* Проверяем, есть ли элемент *C*[*j*] такой, что *A*[*i*] может продолжить подпоследовательность, которую заканчивает *A*[*C*[*j*]] (то есть больше него). Проверяем массив *C* с конца, чтобы алгоритм находил именно максимальную подпоследовательность, которую можно продолжить.
* Если такой элемент *C*[*j*] существует, выполним следующее:
* Обозначим *index*=*C*[*j*] *B*[*i*]=*B*[*index*]+1 — длина подпоследовательности, заканчивающейся *A*[*i*] , теперь есть длина подпоследовательности, которая
* заканчивалась *А*[*index*] , увеличенная на 1. *C*[*B*[*index*]]=*i* — максимальная подпоследовательность длины *B*[*index*]+1 (т.е. увеличенной на 1 длины подпоследовательности,
* которую заканчивает *C*[*j*] ) теперь заканчивается индексом *i* .

Обозначим *index*=*C*[*j*] *B*[*i*]=*B*[*index*]+1 — длина подпоследовательности, заканчивающейся *A*[*i*] , теперь есть длина подпоследовательности, которая

заканчивалась *А*[*index*] , увеличенная на 1. *C*[*B*[*index*]]=*i* — максимальная подпоследовательность длины *B*[*index*]+1 (т.е. увеличенной на 1 длины подпоследовательности,

которую заканчивает *C*[*j*] ) теперь заканчивается индексом *i* . • Если элемент не существует (т.е. он минимальный из рассмотренных):

◦ *B*[*i*]=1 — длина подпоследовательности, заканчивающейся *A*[*i*] , равна 1.

◦ *С*[0]=*i* — надо изменить, т. к. подпоследовательность, заканчивающуюся минимальным элементом, может продолжить больше элементов.

По окончании цикла смотрим размер массива *C* (по последней заполненной ячейке массива *C* индексом *k* ). Это и есть длина максимальной подпоследовательности, т.к. если она заполнена, значит, существует подпоследовательность длины *k*+1 .

Таким же образом переворачиваем(проходим в обратном порядке, иначе трудоемкость задачи выйдет за nlogn) последовательность и рассматриваем подпоследовательности, заканчивающиеся в i-ом элементе. Храним их абсолютно точно также, как и в обычном случае.

Складываем элементы массивов, идя с двух концов, и выбираем максимальный из них, что и является длиной максимальной подпоследовательности с 1им разрывом.

**3.Пример.**

Входные данные:

9

1 2 12 7 3 8 14 13 9

Рассматриваемые массивы:

1 2 3 3 3 4 5 5 5 0 - для обычной последовательности

0 1 1 1 2 3 3 3 4 0 – для просмотренной в обратном порядке

Ответ: 6.