САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5

по курсу «Алгоритмы и структуры данных»

Темы:

Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами

Хеширование. Хеш-таблицы

Динамическое программирование №1

Вариант 19

Выполнил:
Шевченко С.О.
К3144

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

5 Лаба

Задача №1. Куча ли?

1 задача. Куча ли?

Структуру данных «куча», или, более конкретно, «неубывающая пирамида», можно реализовать на основе массива.

Для этого должно выполнятся основное свойство неубывающей пирамиды, которое заключается в том, что для каждого $1 \leq i \leq n$ выполняются условия:

1. если $2i \leq n$, то $a_i \leq a_{2i}$,
2. если $2i + 1 \leq n$, то $a_i \leq a_{2i+1}$.

Дан массив целых чисел. Определите, является ли он неубывающей пирамидой.

- **Формат входного файла (input.txt).** Первая строка входного файла содержит целое число n ($1 \leq n \leq 10^6$). Вторая строка содержит n целых чисел, по модулю не превосходящих $2 \cdot 10^9$.
- **Формат выходного файла (output.txt).** Выведите «YES», если массив является неубывающей пирамидой, и «NO» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

№	input.txt	output.txt
1	5 1 0 1 2 0	NO
2	5 1 3 2 5 4	YES

Листинг кода

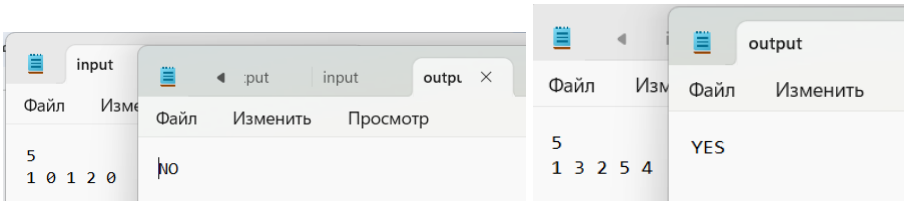
```
import time, tracemalloc

def check(n, a):
    for i in range(1, n + 1):
        if 2 * i <= n:
            print(a[i], a[2 * i])
            if a[i] > a[2 * i]:
                return False
        if 2 * i + 1 <= n:
```

```
            print(a[i], a[2 * i + 1])
            if a[i] > a[2 * i + 1]:
                return False
    return True

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = list(map(int, f1.readline().split()))
a.insert(0, 0)
if not check(n, a):
    f2.write("NO")
else:
    f2.write("YES")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
```

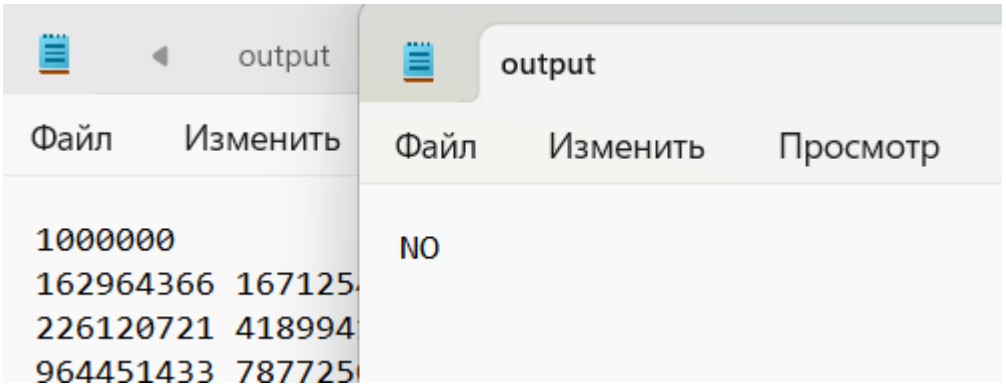
Объяснение

Всё делаем прямо по условию задачи. 0 проблем.

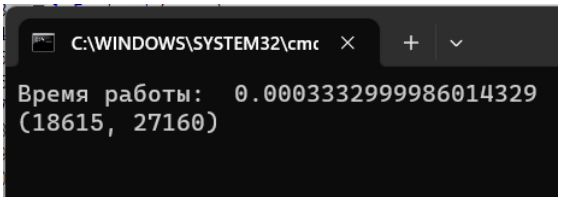
Результат работы кода на примерах из текста задачи

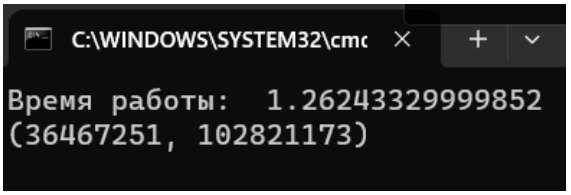


Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).





(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.0003сек	27519 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈1.26сек	102821173 bytes ≈ 100411Kb

Вывод по задаче

Надеюсь, что работает. Очень надеюсь. Но вроде всё сделал по условию, хоть и пришлось в начало вставлять 0, иначе криво с индексами.

Задача №6. Очередь с приоритетами

6 задача. Очередь с приоритетами

Реализуйте очередь с приоритетами. Ваша очередь должна поддерживать следующие операции: добавить элемент, извлечь минимальный элемент, уменьшить элемент, добавленный во время одной из операций.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^6$) - число операций с очередью.

Следующие n строк содержат описание операций с очередью, по одному описанию в строке. Операции могут быть следующими:

- A x – требуется добавить элемент x в очередь.
- X – требуется удалить из очереди минимальный элемент и вывести его в выходной файл. Если очередь пуста, в выходной файл требуется вывести звездочку «*».
- D x y – требуется заменить значение элемента, добавленного в очередь операцией A в строке входного файла номер $x + 1$, на y. Гарантируется, что в строке $x + 1$ действительно находится операция A, что этот элемент не был ранее удален операцией X, и что y меньше, чем предыдущее значение этого элемента.

В очередь помещаются и извлекаются только целые числа, не превышающие по модулю 10^9.

- **Формат выходного файла (output.txt).** Выведите последовательно результат выполнения всех операций X, по одному в каждой строке выходного файла. Если перед очередной операцией X очередь пуста, выведите вместо числа звездочку «*».
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
8 A 3 A 4 A 2 X D 2 1 X X X	2 1 3 *

Листинг кода

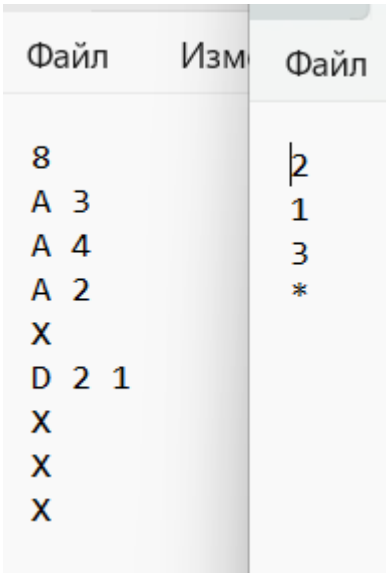
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = []
mi = -1
m = 999999999
k = 0
for i in range(n):
    s = list(f1.readline().split())
    if len(s) == 1:
        a.append(999999999)
```

```
        if k != 0:
            f2.write(str(m) + "\n")
            a[mi] = 999999999
            k -= 1
            if k != 0:
                m = min(a)
                mi = a.index(m)
            else:
                m = 999999999
                mi = -1
        else:
            f2.write("*\n")
    elif len(s) == 2:
        a.append(int(s[-1]))
        if int(s[-1]) < m:
            m = int(s[-1])
            mi = i
        k += 1
    else:
        a.append(999999999)
        a[int(s[1]) - 1] = int(s[2])
        if int(s[2]) < m:
            m = int(s[2])
            mi = int(s[1]) - 1
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

Объяснение

Чуть интересная задачка. Необычно искать именно по строке файла. Я решил, что вместо того, чтобы отдельно сохранять только строки, добавляющие числа, можно взять все, просто в остальных делать огромные значения.

Результат работы кода на примерах из текста задачи



Проверка задачи на (acmp и тд при наличии в задаче).

```
Время работы:  0.0006112999981269 24
(19196, 27802)
```

(пример вывода консоли для примера из задачи)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.0006сек	27802 bytes ≈ 27.5 Kb

Вывод по задаче

Прикольная задача, мне понравилось. Единственное, простите пожалуйста, не успеваю никак написать генератор, там же надо мало того что 3 команды, надо помнить, к каким строкам вообще можно применять D и т.п. В остальном - всё супер

6 Лаба

Задача №1. Множество

1 задача. Множество

Реализуйте множество с операциями «добавление ключа», «удаление ключа», «проверка существования ключа».

- **Формат входного файла (input.txt).** В первой строке входного файла находится строго положительное целое число операций N, не превышающее $5 \cdot 10^5$. В каждой из последующих N строк находится одна из следующих операций:
  - A x – добавить элемент x в множество. Если элемент уже есть в множестве, то ничего делать не надо.
  - D x – удалить элемент x. Если элемента x нет, то ничего делать не надо.
  - ? x – если ключ x есть в множестве, выведите «Y», если нет, то выведите «N».

  Аргументы указанных выше операций – **целые числа**, не превышающие по модулю 10^{18}.
- **Формат выходного файла (output.txt).** Выведите последовательно результат выполнения всех операций «?». Следуйте формату выходного файла из примера.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
8 A 2 A 5 A 3 ? 2 ? 4 A 2 D 2 ? 2	Y N N

Листинг кода

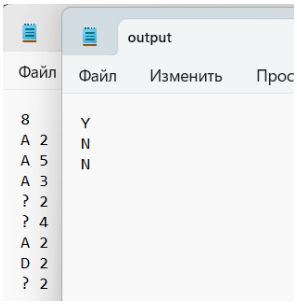
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
```

```
f2 = open("output.txt", "w")
n = int(f1.readline())
d = set()
for i in range(n):
    com, x = f1.readline().split()
    if com == "A":
        d.add(x)
    elif com == "D":
        d.remove(x)
    else:
        if x in d:
            f2.write("Y" + "\n")
        else:
            f2.write("N" + "\n")
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

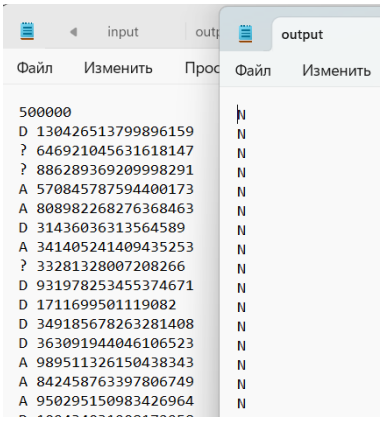
Объяснение

Просто работаем с множествами. Нечего сказать.

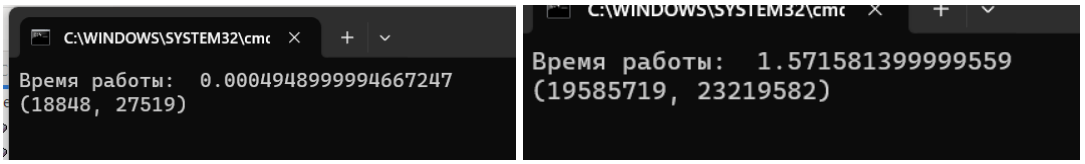
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.0005сек	27519 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈1.57сек	23219582 bytes ≈ 22675Kb

Вывод по задаче

Множества

Нечего сказать)

Задача №8. Почти интерактивная хеш-таблица

8 задача. Почти интерактивная хеш-таблица

В данной задаче у Вас не будет проблем ни с вводом, ни с выводом. Просто реализуйте быструю хеш-таблицу.

В этой хеш-таблице будут храниться целые числа из диапазона $[0; 10^{15} - 1]$. Требуется поддерживать добавление числа x и проверку того, есть ли в таблице число x. Числа, с которыми будет работать таблица, генерируются следующим образом. Пусть имеется четыре целых числа N, X, A, B такие что:

- $1 \le N \le 10^7$
- $1 \le X \le 10^{15}$
- $1 \le A \le 10^3$
- $1 \le B \le 10^{15}$

Требуется N раз выполнить следующую последовательность операций:

- Если X содержится в таблице, то установить $A \leftarrow (A + A_C) \bmod 10^3$, $B \leftarrow (B + B_C) \bmod 10^{15}$.
- Если X не содержится в таблице, то добавить X в таблицу и установить $A \leftarrow (A + A_D) \bmod 10^3$, $B \leftarrow (B + B_D) \bmod 10^{15}$.
- Установить $X \leftarrow (X \cdot A + B) \bmod 10^{15}$.

Начальные значения X, A и B, а также N, A_C, B_C, A_D и B_D даны во входном файле. Выведите значения X, A и B после окончания работы.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится четыре целых числа N, X, A, B. Во второй строке содержится еще четыре целых числа A_C, B_C, A_D и B_D такие что $0 \le A_C, A_D < 10^3$, $0 \le B_C, B_D < 10^{15}$.
- **Формат выходного файла (output.txt).** Выведите значения X, A и B после окончания работы.

- Ограничение по времени. 5 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt
4 0 0 0 1 1 0 0
output.txt
3 1 1

Листинг кода

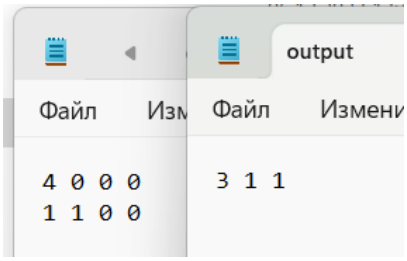
```
import time, tracemalloc

class fastHash:
    def __init__(self):
        self.table = set()

    def addNumber(self, X, A, B, Ac, Bc, Ad, Bd):
        if X in self.table:
            A = (B + Ac) % 1000
            B = (B + Bc) % 10**15
        else:
            self.table.add(X)
            A = (A + Ad) % 1000
            B = (B + Bd) % 10**15

        X = (X * A + B) % 10**15
        return X, A, B

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
N, X, A, B = map(int, f1.readline().split())
Ac, Bc, Ad, Bd = map(int, f1.readline().split())
table = fastHash()
for i in range(N):
    X, A, B = table.addNumber(X, A, B, Ac, Bc, Ad,
Bd)
f2.write(str(X) + " " + str(A) + " " + str(B))
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
```

Объяснение

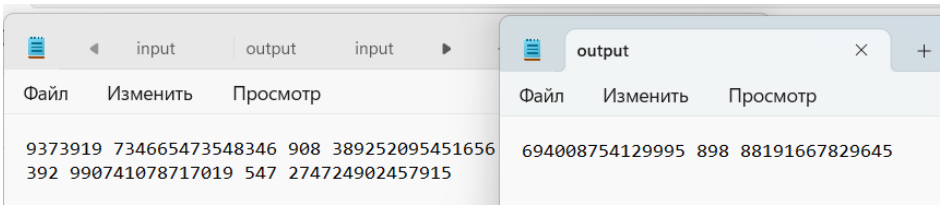
БОль. Я много не спал, поэтому простите, совсем никак не могу оптимизировать. Пытался сделать многое, таблицу просто булевых

значений, работать через класс и т.п., но всё равно, на крайних значениях время очень большое, не проходит по критериям.

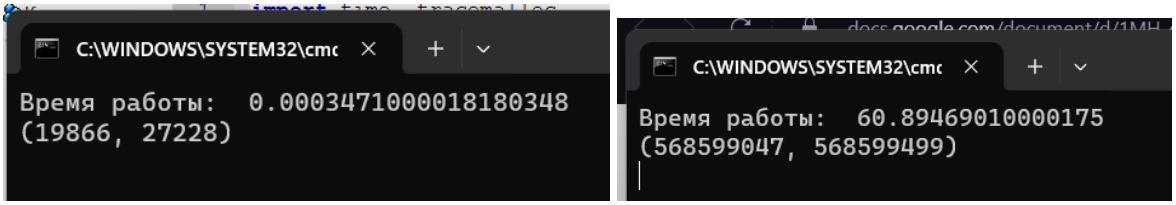
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00034сек	27228 bytes ≈ 27 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈61сек	568599499bytes ≈ 542Mb

Вывод по задаче

Простите, никак быстро не выходит(

7 Лаба

Задача №1. Обмен монет

1 задача. Обмен монет

Как мы уже поняли из лекции, не всегда "жадное"решение задачи на обмен монет работает корректно для разных наборов номиналов монет. Например, если доступны номиналы 1, 3 и 4, жадный алгоритм поменяет 6 центов, используя три монеты (4 + 1 + 1), в то время как его можно изменить, используя всего две монеты (3 + 3). Теперь ваша цель - применить динамическое программирование для решения задачи про обмен монет для разных номиналов.

- **Формат ввода / входного файла (input.txt).** Целое число $money$ ($1 \leq money \leq 10^3$). Набор монет: количество возможных монет k и сам набор $coins = \{coin_1, ..., coin_k\}$. $1 \leq k \leq 100$, $1 \leq coin_i \leq 10^3$. Проверку можно сделать на наборе $\{1, 3, 4\}$. Формат ввода: первая строка содержит через пробел $money$ и k; вторая - $coin_1 coin_2 ... coin_k$.
  - ***Вариация 2:*** Количество монет в кассе ограничено. Для каждой монеты из набора $coins = \{coin_1, ..., coin_k\}$ есть соответствующее целое число - количество монет в кассе данного номинала $c = \{c_1, ..., c_k\}$. Если они закончились, то выдать данную монету невозможно.
- **Формат вывода / выходного файла (output.txt).** Вывести одно число – минимальное количество необходимых монет для размена $money$ доступным набором монет $coins$.
- Ограничение по времени. 1 сек.
- Примеры:

input.txt	output.txt	input.txt	output.txt
2 3 1 3 4	2	34 3 1 3 4	9

Листинг кода

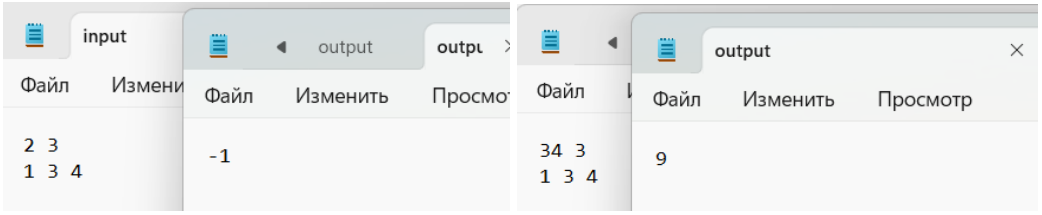
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
money, k = map(int, f1.readline().split())
a = list(map(int, f1.readline().split()))
dp = [100000] * (money + 1)
dp[0] = 0
for x in a:
```

```
    for i in range(x, money + 1):
        if dp[i] != 100000:
            dp[i] = min(dp[i], dp[i - x] + 1)
        else:
            dp[i] = dp[i - x] + 1
if dp[money] >= 100000:
    f2.write(str(-1))
else:
    f2.write(str(dp[money]))
print("Время работы: ", (time.perf_counter() - t_start))
print(tracemalloc.get_traced_memory())
```

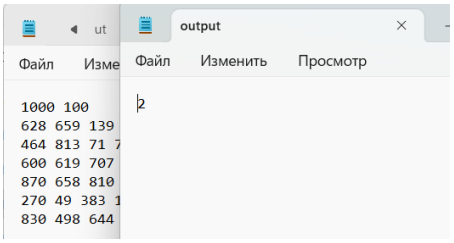
Объяснение

Урааа, дп. Просто заполняем список, в нём элемент с индексом i - минимальное количество монет для получения такой суммы денег. Если не получается никак набрать такое количество - вывожу -1.

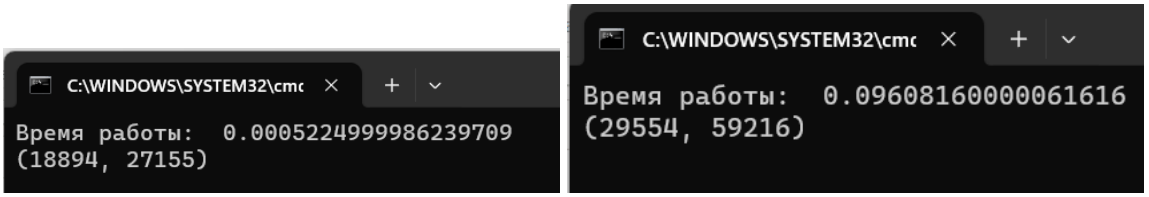
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00052 сек	27155 bytes ≈ 26.5 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈0.096сек	59216 bytes ≈ 57.8Kb

Вывод по задаче

Урааа, дп. В целом всё понятно, хоть вспомнил как это делается(впервые года за 2).

Задача №5. Наибольшая общая подпоследовательность трёх последовательностей

Вычислить длину самой длинной общей подпоследовательности из *трех* последовательностей.

Даны три последовательности $A = (a_1, a_2, ..., a_n)$, $B = (b_1, b_2, ..., b_m)$ и $C = (c_1, c_2, ..., c_l)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \leq i_1 < i_2 < ... < i_p \leq n$, $1 \leq j_1 < j_2 < ... < j_p \leq m$ и $1 \leq k_1 < k_2 < ... < k_p \leq l$ такие, что $a_{i_1} = b_{j_1} = c_{k_1}, ..., a_{i_p} = b_{j_p} = c_{k_p}$.

- **Формат ввода / входного файла (input.txt).**
  - Первая строка: n - длина первой последовательности.
  - Вторая строка: $a_1, a_2, ..., a_n$ через пробел.
  - Третья строка: m - длина второй последовательности.
  - Четвертая строка: $b_1, b_2, ..., b_m$ через пробел.
  - Пятая строка: l - длина второй последовательности.
  - Шестая строка: $c_1, c_2, ..., c_l$ через пробел.
- Ограничения: $1 \leq n, m, l \leq 100$; $-10^9 < a_i, b_i, c_i < 10^9$.
- **Формат вывода / выходного файла (output.txt).** Выведите число p.
- Ограничение по времени. 1 сек.

- Примеры:

input.txt	output.txt	input.txt	output.txt
3 1 2 3 3 2 1 3 3 1 3 5	2	5 8 3 2 1 7 7 8 2 1 3 8 10 7 6 6 8 3 1 4 7	3

- В первом примере одна общая подпоследовательность – $(1, 3)$ длиной 2. Во втором примере есть две общие последовательности длиной 3 элемента – $(8, 3, 7)$ и $(8, 1, 7)$.

Листинг кода

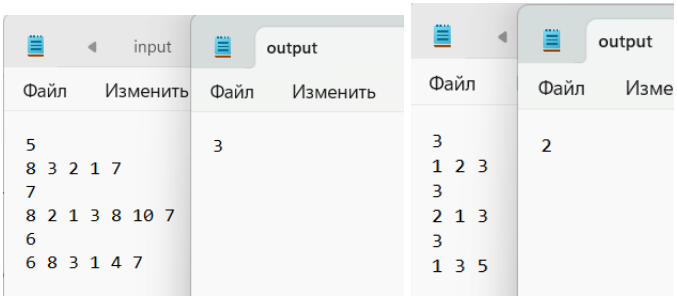
```
import time, tracemalloc

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
A = list(map(int, f1.readline().split()))
```

```
m = int(f1.readline())
B = list(map(int, f1.readline().split()))
l = int(f1.readline())
C = list(map(int, f1.readline().split()))
dp = [[[0 for i in range(l+1)] for j in range(m+1)]
for k in range(n+1)]
for i in range(1, n+1):
    for j in range(1, m+1):
        for k in range(1, l+1):
            if A[i-1] == B[j-1] == C[k-1]:
                dp[i][j][k] = dp[i-1][j-1][k-1] + 1
            else:
                dp[i][j][k]   =   max(dp[i-1][j][k],
dp[i][j-1][k], dp[i][j][k-1])
f2.write(str(dp[n][m][l]))
print("Время   работы:   ",   (time.perf_counter()   -
t_start))
print(tracemalloc.get_traced_memory())
```

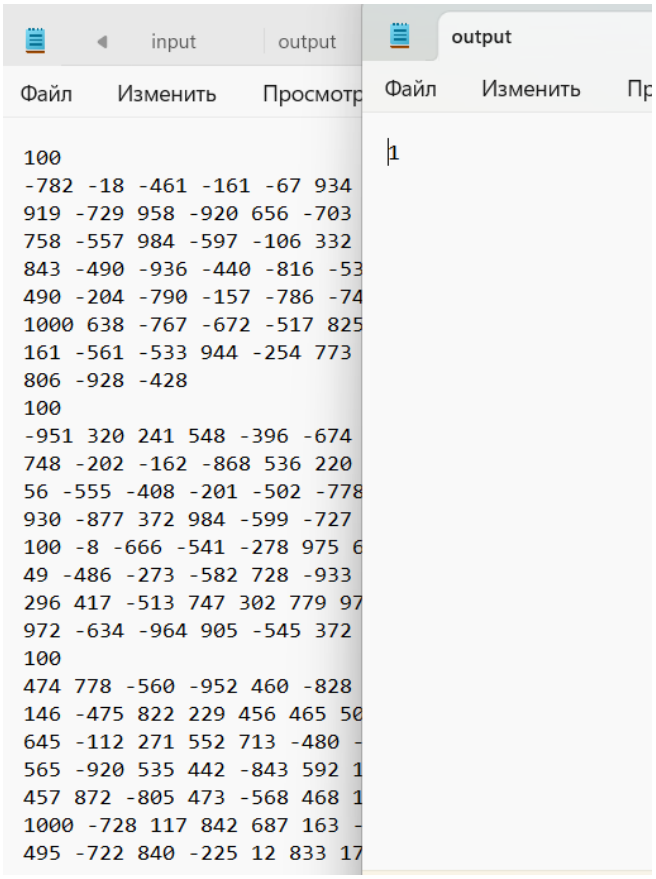
Объяснение

Тут пришлось делать трёхмерный массив. В целом суть заключается в том, что если мы находим какой-то один повторяющийся элемент, то этот элемент массива увеличиваем. Если же не равны - то просто берём наибольший из трёх вариантов длин. Вроде работает. Надеюсь)

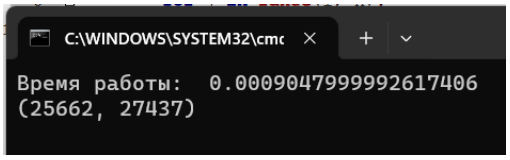
Результат работы кода на примерах из текста задачи

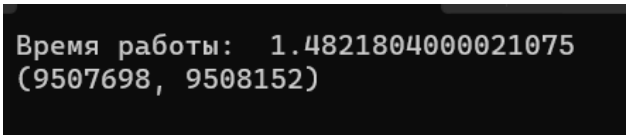


Результат работы кода на максимальных значениях



Проверка задачи на (acmp и тд при наличии в задаче).





(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.0009 сек	27437 bytes ≈ 26.5 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈1.48сек	9508152 bytes ≈ 9285Kb

Вывод по задаче

Вроде работает.. Мерзко работать с трехмерным массивом, но вроде как по аналогии с другими должно работать…

Вывод

Простите, не успел завершить лабораторную работу, объясню более подробно собственно на защите. Но из того что успел, узнал кучи, вспомнил множества и попытался придумать хэш-таблицы, также вспомнил дп, это было самой приятной частью)