**ЛАБОРАТОРНАЯ РАБОТА №1**

**«Вероятностные алгоритмы»**

по дисциплине «Программирование на языке *Python*»

Вариант 13

Студента Колодницкого Ильи Михайловича

фамилия, имя, отчество полностью

Курс 2 Группа ФИТ-232

Направление 02.03.02 Фундаментальная информатика и информационные технологии и

код, наименование

Проверил ассистент

должность

Плескунов Д. А.

фамилия, инициалы

Выполнил

дата, подпись студента

дата, подпись преподователя

**Задание №1 «фильтр Блума»**

1. Реализовать фильтр Блума с использованием стандартной библиотеки Python (при этом реализовать собственные хеш-функции).
2. Определить процент ложноположительных срабатываний конкретной реализации.
3. Оценить зависимость ложноположительных срабатываний относительно размерности массива m и числа хеш-функций k (таблица и графики зависимостей).
4. Произвести многофакторный дисперсионный анализ для выявления степени значимости следующих факторов: размерности массива m и числа хеш-функций k. (3 курс)
5. При реализации учесть возможность пересечения и объединения фильтров Блума.

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def hash\_mult\_add(item, m):

hash\_value = 0

for char in str(item):

hash\_value = (hash\_value \* 31 + ord(char))

return hash\_value % m

def hash\_xor(item, m):

hash\_value = 0

for char in str(item):

hash\_value ^= (hash\_value << 5) + ord(char) + (hash\_value >> 2)

return hash\_value % m

class BloomFilter:

def \_\_init\_\_(self, m, k, hash\_fn):

self.m = m

self.k = k

self.hash\_fn = hash\_fn

self.bit\_array = [0] \* m

def add(self, item):

for i in range(self.k):

index = self.hash\_fn(f"{item}\_{i}", self.m)

self.bit\_array[index] = 1

def contains(self, item):

for i in range(self.k):

index = self.hash\_fn(f"{item}\_{i}", self.m)

if self.bit\_array[index] == 0:

return False

return True

def \_\_or\_\_(self, other):

if self.m != other.m or self.k != other.k:

raise ValueError("Фильтры должны иметь одинаковые m и k")

new\_filter = BloomFilter(self.m, self.k, self.hash\_fn)

new\_filter.bit\_array = [a | b for a, b in zip(self.bit\_array, other.bit\_array)]

return new\_filter

def \_\_and\_\_(self, other):

if self.m != other.m or self.k != other.k:

raise ValueError("Фильтры должны иметь одинаковые m и k")

new\_filter = BloomFilter(self.m, self.k, self.hash\_fn)

new\_filter.bit\_array = [a & b for a, b in zip(self.bit\_array, other.bit\_array)]

return new\_filter

def false\_positive\_rate(bloom\_filter, added\_elements, test\_elements):

for element in added\_elements:

bloom\_filter.add(element)

false\_positives = 0

for element in test\_elements:

if bloom\_filter.contains(element):

false\_positives += 1

return false\_positives / len(test\_elements)

def evaluate\_false\_positives(n, m\_values, k\_values, hash\_fn):

results = []

for m in m\_values:

for k in k\_values:

bloom\_filter = BloomFilter(m, k, hash\_fn)

added\_elements = np.random.randint(0, 100000, n)

test\_elements = np.random.randint(100001, 200000, n)

fp\_rate = false\_positive\_rate(bloom\_filter, added\_elements, test\_elements)

results.append((m, k, fp\_rate))

return results

n = 1000

m\_values = [1000, 5000, 10000]

k\_values = [3, 5, 7]

hash\_functions = {

"hash\_mult\_add": hash\_mult\_add,

"hash\_xor": hash\_xor,

}

for hash\_name, hash\_fn in hash\_functions.items():

print(f"Результаты для хеш-функции: {hash\_name}")

results = evaluate\_false\_positives(n, m\_values, k\_values, hash\_fn)

df = pd.DataFrame(results, columns=["m", "k", "Ложноположительные срабатывания"])

print(df)

plt.figure(figsize=(10, 6))

for k in k\_values:

m\_list = [m for (m, k\_val, \_) in results if k\_val == k]

fp\_list = [fp for (\_, k\_val, fp) in results if k\_val == k]

plt.plot(m\_list, fp\_list, marker='o', label=f"k={k}")

plt.xlabel("Размер массива (m)")

plt.ylabel("Ложноположительные срабатывания")

plt.title(f"Зависимость ложноположительных срабатываний от m и k ({hash\_name})")

plt.legend()

plt.grid()

plt.show()

Пример работы программы представлен на рисунке 1 и 2.

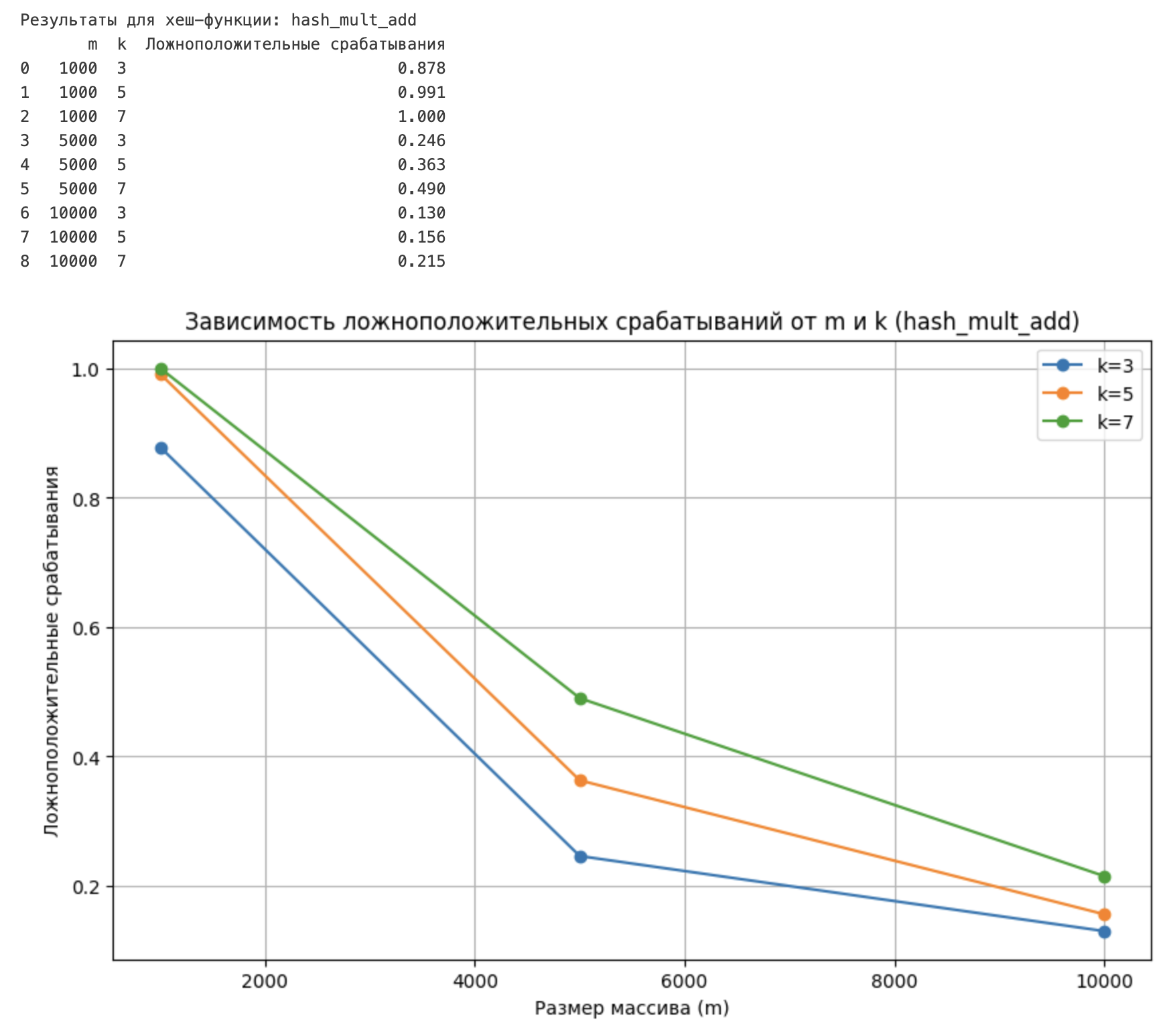


Рисунок 1 – работа программы для задания №1

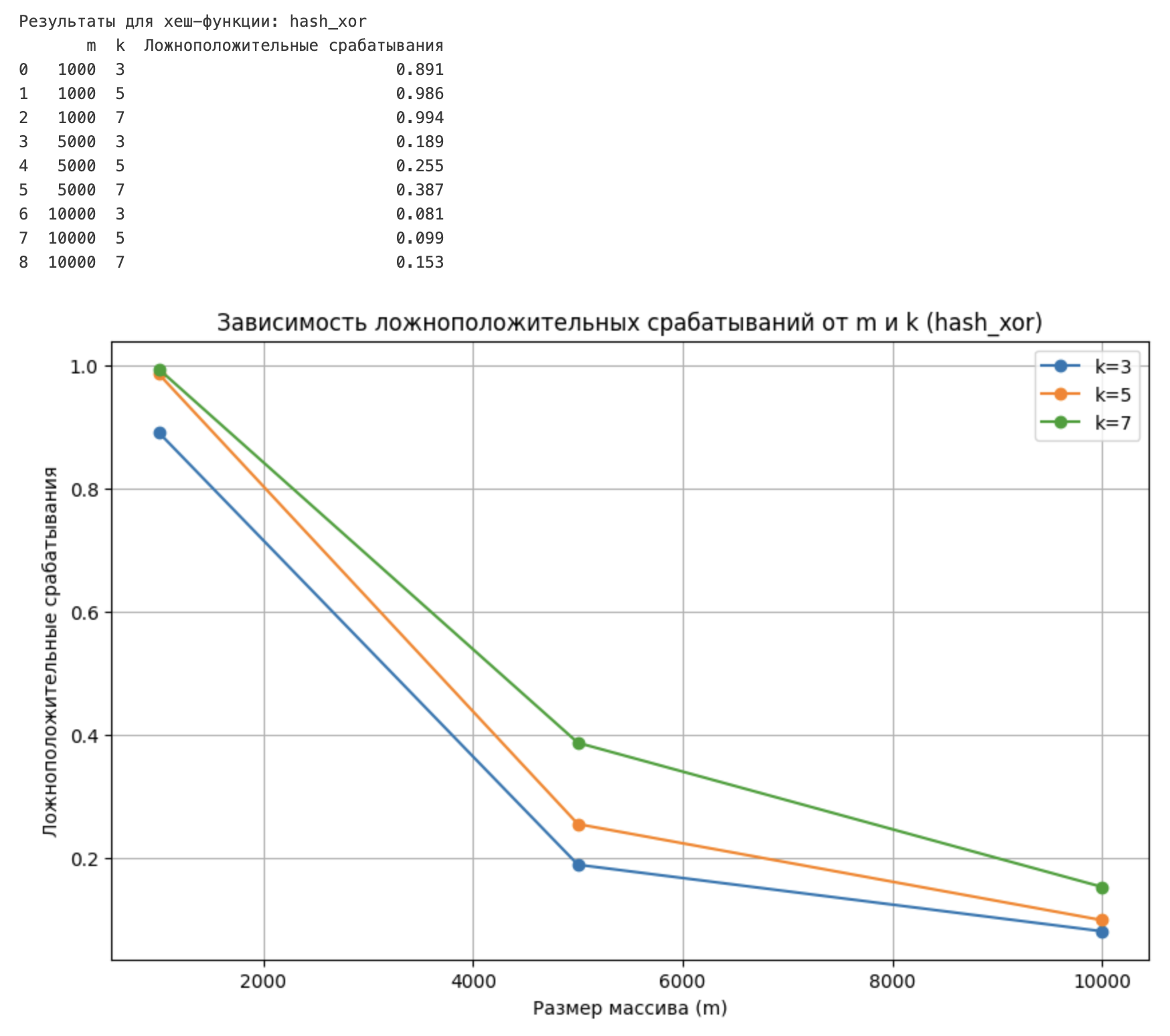


Рисунок 2 – работа программы для задания №1

bf1 = BloomFilter(m=1000, k=3, hash\_fn=hash\_xor)

bf2 = BloomFilter(m=1000, k=3, hash\_fn=hash\_xor)

words1 = ["apple", "banana", "cherry"]

for word in words1:

bf1.add(word)

words2 = ["banana", "cherry", "date"]

for word in words2:

bf2.add(word)

bf\_union = bf1 | bf2

bf\_intersection = bf1 & bf2

print("Объединенный фильтр:")

for word in ["apple", "banana", "cherry", "date", "elderberry"]:

if bf\_union.contains(word):

print(f"Слово '{word}' вероятно принадлежит объединенному множеству.")

print("\nПересечение фильтров:")

for word in ["apple", "banana", "cherry", "date", "elderberry"]:

if bf\_intersection.contains(word):

print(f"Слово '{word}' вероятно принадлежит пересечению множеств.")

Пример работы программы представлен на рисунке 3.

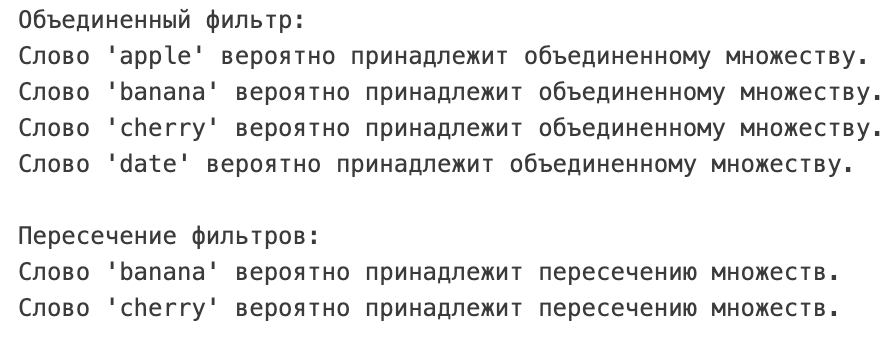


Рисунок 3 – работа программы для задания №1

**Задание №2 «фильтр Блума со счетом»**

1. Реализовать фильтр Блума со счетом
2. Определить процент ложноположительных срабатываний конкретной реализации на добавление и удаление.
3. Оценить зависимость ложноположительных срабатываний от гиперпараметров алгоритма (таблица и графики зависимостей).
4. При реализации учесть возможность пересечения и объединения фильтров Блума.

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def hash\_mult\_add(item, m):

hash\_value = 0

for char in str(item):

hash\_value = (hash\_value \* 31 + ord(char))

return hash\_value % m

def hash\_xor(item, m):

hash\_value = 0

for char in str(item):

hash\_value ^= (hash\_value << 5) + ord(char) + (hash\_value >> 2)

return hash\_value % m

class CountingBloomFilter:

def \_\_init\_\_(self, m, k, hash\_fn):

self.m = m

self.k = k

self.hash\_fn = hash\_fn

self.counter\_array = [0] \* m

def add(self, item):

for i in range(self.k):

index = self.hash\_fn(f"{item}\_{i}", self.m)

self.counter\_array[index] += 1

def contains(self, item):

for i in range(self.k):

index = self.hash\_fn(f"{item}\_{i}", self.m)

if self.counter\_array[index] == 0:

return False

return True

def remove(self, item):

for i in range(self.k):

index = self.hash\_fn(f"{item}\_{i}", self.m)

if self.counter\_array[index] > 0:

self.counter\_array[index] -= 1

def \_\_or\_\_(self, other):

if self.m != other.m or self.k != other.k:

raise ValueError("Фильтры должны иметь одинаковые m и k")

new\_filter = CountingBloomFilter(self.m, self.k, self.hash\_fn)

new\_filter.counter\_array = [a + b for a, b in zip(self.counter\_array, other.counter\_array)]

return new\_filter

def \_\_and\_\_(self, other):

if self.m != other.m or self.k != other.k:

raise ValueError("Фильтры должны иметь одинаковые m и k")

new\_filter = CountingBloomFilter(self.m, self.k, self.hash\_fn)

new\_filter.counter\_array = [min(a, b) for a, b in zip(self.counter\_array, other.counter\_array)]

return new\_filter

def false\_positive\_rate(bloom\_filter, added\_elements, test\_elements):

for element in added\_elements:

bloom\_filter.add(element)

false\_positives = 0

for element in test\_elements:

if bloom\_filter.contains(element):

false\_positives += 1

return false\_positives / len(test\_elements)

def evaluate\_false\_positives(n, m\_values, k\_values, hash\_fn):

results = []

for m in m\_values:

for k in k\_values:

bloom\_filter = CountingBloomFilter(m, k, hash\_fn)

added\_elements = np.random.randint(0, 100000, n)

test\_elements = np.random.randint(100001, 200000, n)

fp\_rate = false\_positive\_rate(bloom\_filter, added\_elements, test\_elements)

results.append((m, k, fp\_rate))

return results

n = 1000

m\_values = [1000, 5000, 10000]

k\_values = [3, 5, 7]

hash\_functions = {

"hash\_mult\_add": hash\_mult\_add,

"hash\_xor": hash\_xor,

}

for hash\_name, hash\_fn in hash\_functions.items():

print(f"Результаты для хеш-функции: {hash\_name}")

results = evaluate\_false\_positives(n, m\_values, k\_values, hash\_fn)

df = pd.DataFrame(results, columns=["m", "k", "Ложноположительные срабатывания"])

print(df)

plt.figure(figsize=(10, 6))

for k in k\_values:

m\_list = [m for (m, k\_val, \_) in results if k\_val == k]

fp\_list = [fp for (\_, k\_val, fp) in results if k\_val == k]

plt.plot(m\_list, fp\_list, marker='o', label=f"k={k}")

plt.legend()

plt.grid()

plt.show()

Пример работы программы представлен на рисунке 4 и 5.

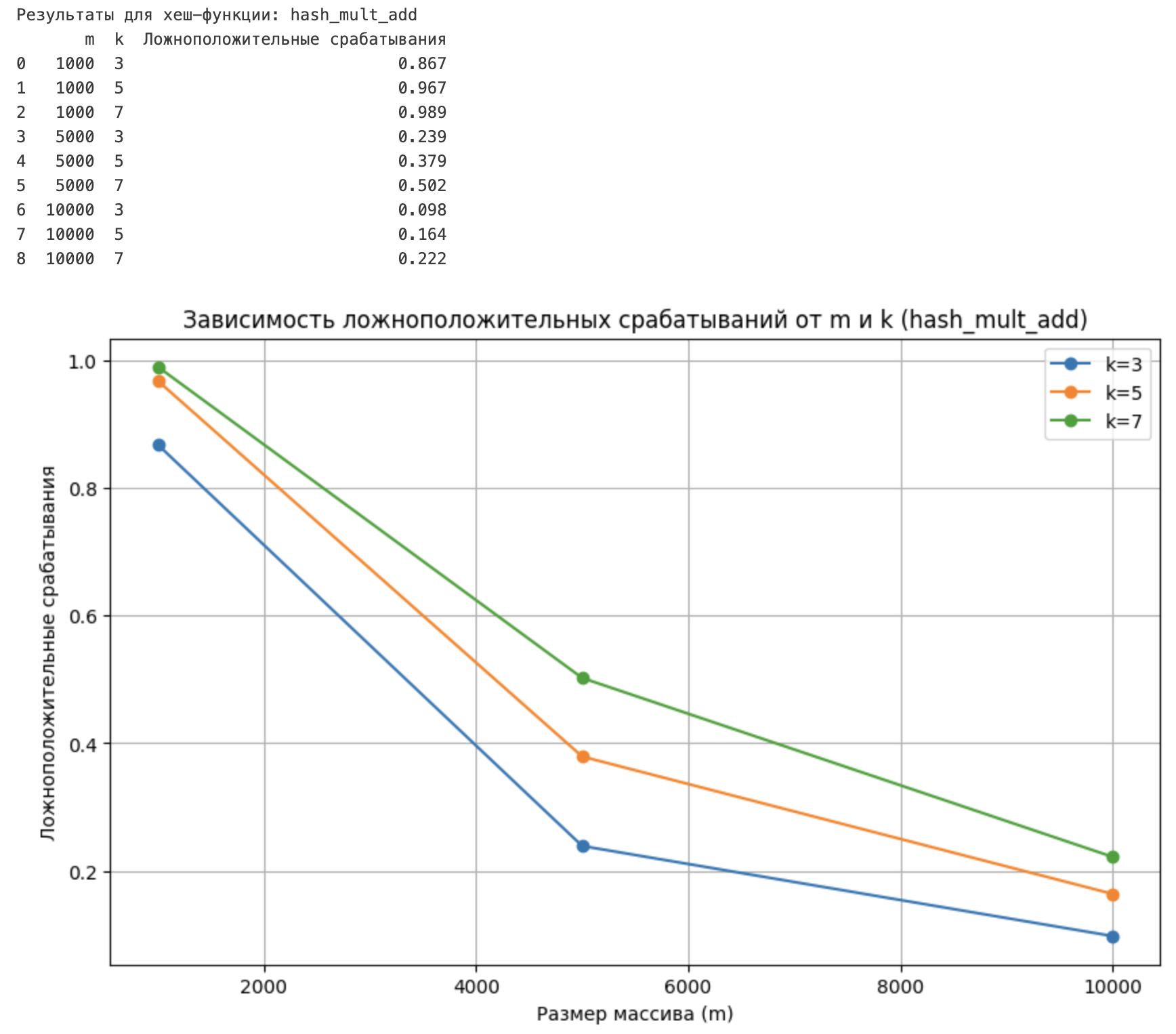


Рисунок 4 – работа программы для задания №2

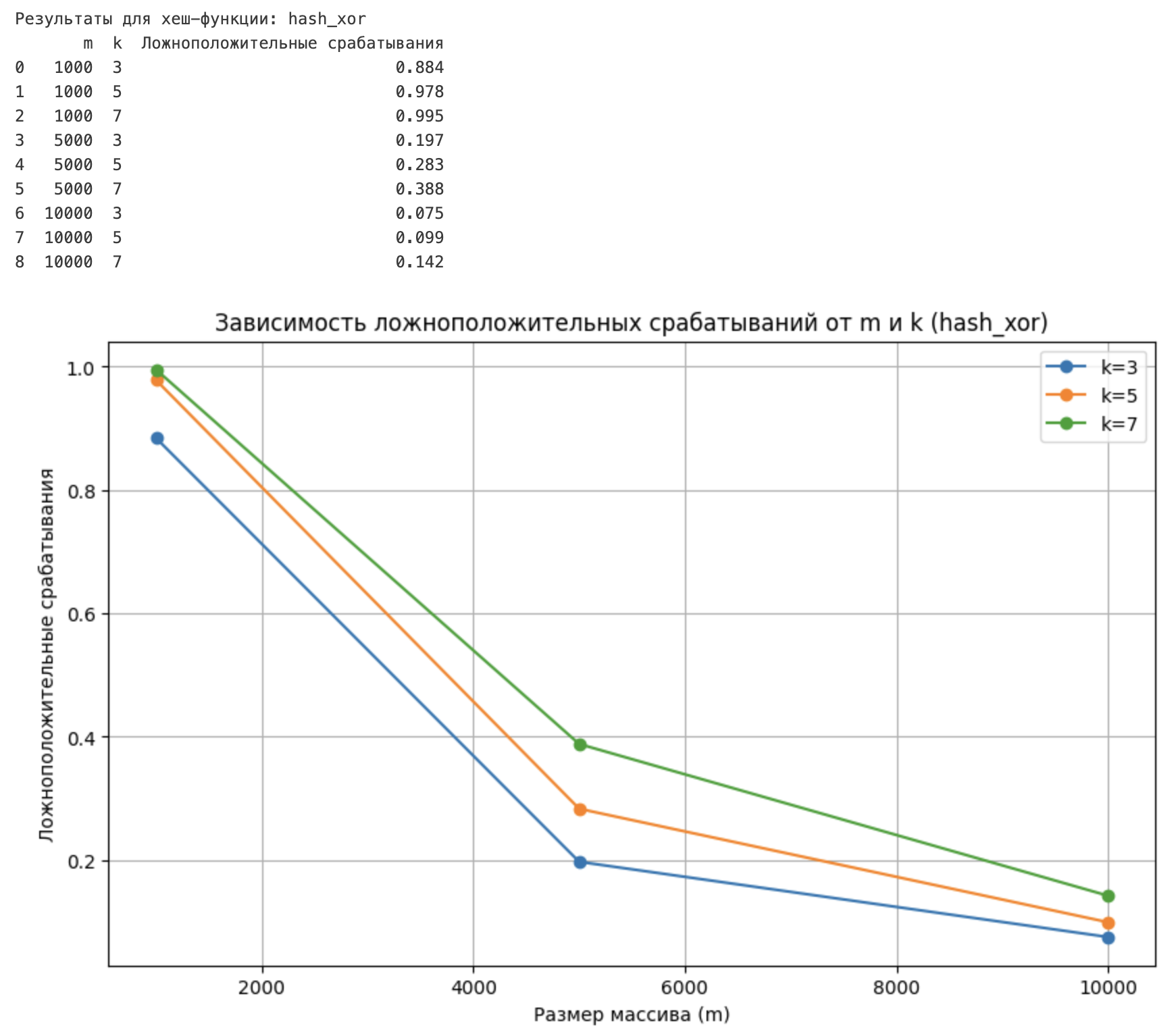


Рисунок 5 – работа программы для задания №2

bf1 = CountingBloomFilter(m=1000, k=3, hash\_fn=hash\_xor)

bf2 = CountingBloomFilter(m=1000, k=3, hash\_fn=hash\_xor)

words1 = ["apple", "banana", "cherry"]

for word in words1:

bf1.add(word)

words2 = ["banana", "cherry", "date"]

for word in words2:

bf2.add(word)

bf\_union = bf1 | bf2

bf\_intersection = bf1 & bf2

print("Объединенный фильтр:")

for word in ["apple", "banana", "cherry", "date", "elderberry"]:

if bf\_union.contains(word):

print(f"Слово '{word}' вероятно принадлежит объединенному множеству.")

print("\nПересечение фильтров:")

for word in ["apple", "banana", "cherry", "date", "elderberry"]:

if bf\_intersection.contains(word):

print(f"Слово '{word}' вероятно принадлежит пересечению множеств.")

Пример работы программы представлен на рисунке 6.

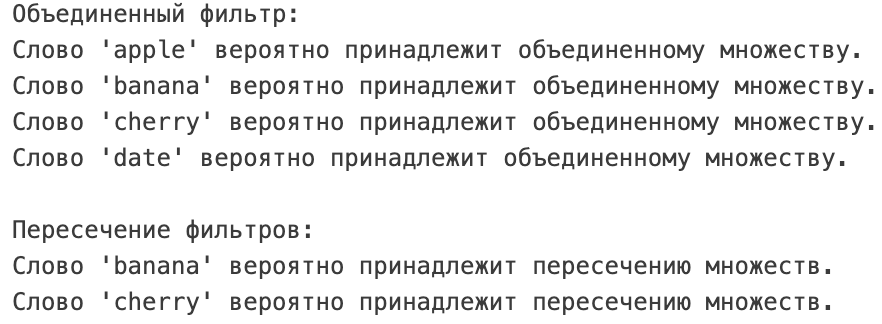


Рисунок 6 – работа программы для задания №2

**Задание №3 «HyperLogLog»**

1. Реализовать HyperLogLog.
2. Определить процент ложноположительных срабатываний конкретной реализации.
3. Оценить зависимость ложноположительных срабатываний от гиперпараметров алгоритма (таблица и графики зависимостей).

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def hash\_mult\_add(item, m):

hash\_value = 0

for char in str(item):

hash\_value = (hash\_value \* 31 + ord(char))

return hash\_value % m

def hash\_xor(item, m):

hash\_value = 0

for char in str(item):

hash\_value ^= (hash\_value << 5) + ord(char) + (hash\_value >> 2)

return hash\_value % m

class HyperLogLog:

def \_\_init\_\_(self, b=4, hash\_fn=hash\_mult\_add):

self.b = b

self.m = 2 \*\* b

self.registers = [0] \* self.m

self.alpha = self.\_get\_alpha()

self.hash\_fn = hash\_fn

def \_get\_alpha(self):

if self.m == 16:

return 0.673

elif self.m == 32:

return 0.697

elif self.m == 64:

return 0.709

else:

return 0.7213 / (1 + 1.079 / self.m)

def \_hash(self, item):

return self.hash\_fn(item, 1 << 64)

def add(self, item):

hash\_value = self.\_hash(item)

index = hash\_value & (self.m - 1)

w = hash\_value >> self.b

self.registers[index] = max(self.registers[index], self.\_count\_leading\_zeros(w) + 1)

def \_count\_leading\_zeros(self, w):

if w == 0:

return 64 - self.b

return bin(w)[2:].zfill(64 - self.b).index('1')

def count(self):

harmonic\_mean = sum(2 \*\* -r for r in self.registers)

estimate = self.alpha \* self.m \*\* 2 / harmonic\_mean

if estimate <= 2.5 \* self.m:

zeros = self.registers.count(0)

if zeros != 0:

return self.m \* np.log(self.m / zeros)

elif estimate > (1 << 64) / 30:

return -(1 << 64) \* np.log(1 - estimate / (1 << 64))

return estimate

def evaluate\_hyperloglog\_accuracy(n, b\_values, hash\_fn):

results = []

for b in b\_values:

hll = HyperLogLog(b=b, hash\_fn=hash\_fn)

elements = np.random.randint(0, 1000000, n)

unique\_elements = set(elements)

for element in elements:

hll.add(element)

estimated\_count = hll.count()

true\_count = len(unique\_elements)

error = abs(estimated\_count - true\_count) / true\_count

results.append((b, estimated\_count, true\_count, error))

return results

n = 100000

b\_values = [4, 6, 8, 10]

hash\_functions = {

"hash\_mult\_add": hash\_mult\_add,

"hash\_xor": hash\_xor,

}

for hash\_name, hash\_fn in hash\_functions.items():

print(f"Результаты для хеш-функции: {hash\_name}")

results = evaluate\_hyperloglog\_accuracy(n, b\_values, hash\_fn)

df = pd.DataFrame(results, columns=["b", "Оценка", "Истинное значение", "Ошибка"])

print(df)

plt.figure(figsize=(10, 6))

m\_list = [2 \*\* b for b in b\_values]

error\_list = [error for (b, est, true, error) in results]

plt.plot(m\_list, error\_list, marker='o', label=f"{hash\_name}")

plt.xlabel("Размер массива (m = 2^b)")

plt.ylabel("Ошибка оценки")

plt.title(f"Зависимость ошибки оценки от размера массива (m) ({hash\_name})")

plt.legend()

plt.grid()

plt.show()

hll = HyperLogLog(b=4, hash\_fn=hash\_mult\_add)

words = ["apple", "banana", "cherry", "date", "elderberry"]

for word in words:

hll.add(word)

estimated\_count = hll.count()

print(f"Оценка количества уникальных элементов: {estimated\_count}")

Пример работы программы представлен на рисунке 7.

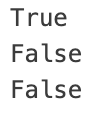


Рисунок 7 – работа программы для задания №3

**Задание №4 «Quotient filter»**

1. Реализовать Фильтр коэффициентов (Quotient filter).
2. Определить процент ложноположительных срабатываний конкретной реализации.
3. Оценить зависимость ложноположительных срабатываний от гиперпараметров алгоритма (таблица и графики зависимостей).

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

class QuotientFilter:

def \_\_init\_\_(self, q, r):

self.q = q

self.r = r

self.size = 1 << q

self.table = [None] \* self.size

self.occupied = [False] \* self.size

self.runends = [False] \* self.size

def \_hash(self, item):

if isinstance(item, str):

item = item.encode()

hash\_val = hash(item) & ((1 << (self.q + self.r)) - 1)

quotient = hash\_val >> self.r

remainder = hash\_val & ((1 << self.r) - 1)

return quotient, remainder

def insert(self, item):

quotient, remainder = self.\_hash(item)

if self.occupied[quotient]:

start = quotient

while start > 0 and self.occupied[start - 1]:

start -= 1

pos = start

while pos < quotient:

if self.table[pos] is None:

break

pos += 1

else:

while pos < self.size and self.occupied[pos] and self.table[pos] is not None:

pos += 1

if pos >= self.size:

pos = 0

while pos < quotient and self.occupied[pos] and self.table[pos] is not None:

pos += 1

if pos < quotient:

for i in range(pos, quotient):

if self.table[i] is None:

self.table[i] = self.table[i-1]

self.runends[i] = self.runends[i-1]

self.runends[i-1] = False

self.table[quotient] = remainder

self.runends[quotient] = True

else:

self.table[pos] = remainder

self.runends[pos] = True

else:

self.table[quotient] = remainder

self.runends[quotient] = True

self.occupied[quotient] = True

def lookup(self, item):

quotient, remainder = self.\_hash(item)

if not self.occupied[quotient]:

return False

start = quotient

while start > 0 and self.occupied[start - 1]:

start -= 1

pos = start

while pos < self.size and self.occupied[pos]:

if self.table[pos] == remainder:

return True

if self.runends[pos]:

break

pos += 1

return False

def delete(self, item):

quotient, remainder = self.\_hash(item)

if not self.occupied[quotient]:

return False

start = quotient

while start > 0 and self.occupied[start - 1]:

start -= 1

pos = start

found\_pos = -1

while pos < self.size and self.occupied[pos]:

if self.table[pos] == remainder:

found\_pos = pos

if self.runends[pos]:

break

pos += 1

if found\_pos == -1:

return False

self.table[found\_pos] = None

if found\_pos == quotient:

has\_other = False

pos = start

while pos < self.size and self.occupied[pos]:

if pos != found\_pos and (self.\_get\_quotient(pos) == quotient):

has\_other = True

break

if self.runends[pos]:

break

pos += 1

if not has\_other:

self.occupied[quotient] = False

self.runends[quotient] = False

return True

def \_get\_quotient(self, pos):

start = pos

while start > 0 and self.occupied[start - 1]:

start -= 1

current = start

quotient = None

while current <= pos:

if self.runends[current]:

quotient = current

current += 1

return quotient

def \_\_contains\_\_(self, item):

return self.lookup(item)

def \_\_repr\_\_(self):

return f"QuotientFilter(q={self.q}, r={self.r}, size={self.size})"

qf = QuotientFilter(10, 4)

qf.insert("712cetbg")

qf.insert("39gyfhau")

qf.insert("ra4j5pgoipongdae")

print("712cetbg" in qf)

print("39gyfhau" in qf)

print("ra4j5pgoipongdae" in qf)

qf.delete("ra4j5pgoipongdae")

print("ra4j5pgoipongdae" in qf)

Пример работы программы представлен на рисунке 8.

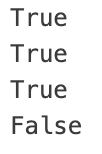


Рисунок 8 – работа программы для задания №4

import random

import string

def generate\_random\_string(length=10):

return ''.join(random.choices(string.ascii\_letters + string.digits, k=length))

def measure\_false\_positive\_rate(q, r, num\_inserted, num\_tested):

qf = QuotientFilter(q, r)

inserted\_elements = set()

while len(inserted\_elements) < num\_inserted:

item = generate\_random\_string()

if item not in inserted\_elements:

qf.insert(item)

inserted\_elements.add(item)

tested\_elements = set()

while len(tested\_elements) < num\_tested:

item = generate\_random\_string()

if item not in inserted\_elements and item not in tested\_elements:

tested\_elements.add(item)

false\_positives = 0

for item in tested\_elements:

if item in qf:

false\_positives += 1

false\_positive\_rate = (false\_positives / num\_tested) \* 100

return false\_positive\_rate

Q = 10

R = 6

NUM\_INSERTED = 500

NUM\_TESTED = 10000

fp\_rate = measure\_false\_positive\_rate(Q, R, NUM\_INSERTED, NUM\_TESTED)

print(f"Процент ложноположительных срабатываний: {fp\_rate:.2f}%")

Процент ложноположительных срабатываний: 0.56%

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import random

import string

def generate\_random\_string(length=10):

return ''.join(random.choices(string.ascii\_letters + string.digits, k=length))

def analyze\_quotient\_filter\_performance():

q\_values = [8, 10, 12]

r\_values = [4, 6, 8]

num\_inserted = 500

num\_tested = 5000

trials = 3

results = []

for q in q\_values:

for r in r\_values:

if q + r > 64:

continue

total\_fp = 0

for \_ in range(trials):

qf = QuotientFilter(q, r)

inserted\_elements = set()

while len(inserted\_elements) < num\_inserted:

item = generate\_random\_string()

if item not in inserted\_elements:

qf.insert(item)

inserted\_elements.add(item)

tested\_elements = set()

while len(tested\_elements) < num\_tested:

item = generate\_random\_string()

if item not in inserted\_elements and item not in tested\_elements:

tested\_elements.add(item)

false\_positives = 0

for item in tested\_elements:

if item in qf:

false\_positives += 1

total\_fp += (false\_positives / num\_tested) \* 100

avg\_fp = total\_fp / trials

results.append({

'q': q,

'r': r,

'table\_size': 2 \*\* q,

'False Positive Rate (%)': avg\_fp

})

df = pd.DataFrame(results)

print("\nРезультаты:")

display(df[['q', 'r', 'table\_size', 'False Positive Rate (%)']]

.sort\_values(['q', 'r']))

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

for q\_val in q\_values:

subset = df[df['q'] == q\_val]

plt.plot(subset['r'], subset['False Positive Rate (%)'],

marker='o', label=f'q={q\_val}')

plt.legend()

plt.grid(True)

plt.subplot(1, 2, 2)

for r\_val in r\_values:

subset = df[df['r'] == r\_val]

plt.plot(subset['q'], subset['False Positive Rate (%)'],

marker='o', label=f'r={r\_val}')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

return df

results\_df = analyze\_quotient\_filter\_performance()

Пример работы программы представлен на рисунке 9.

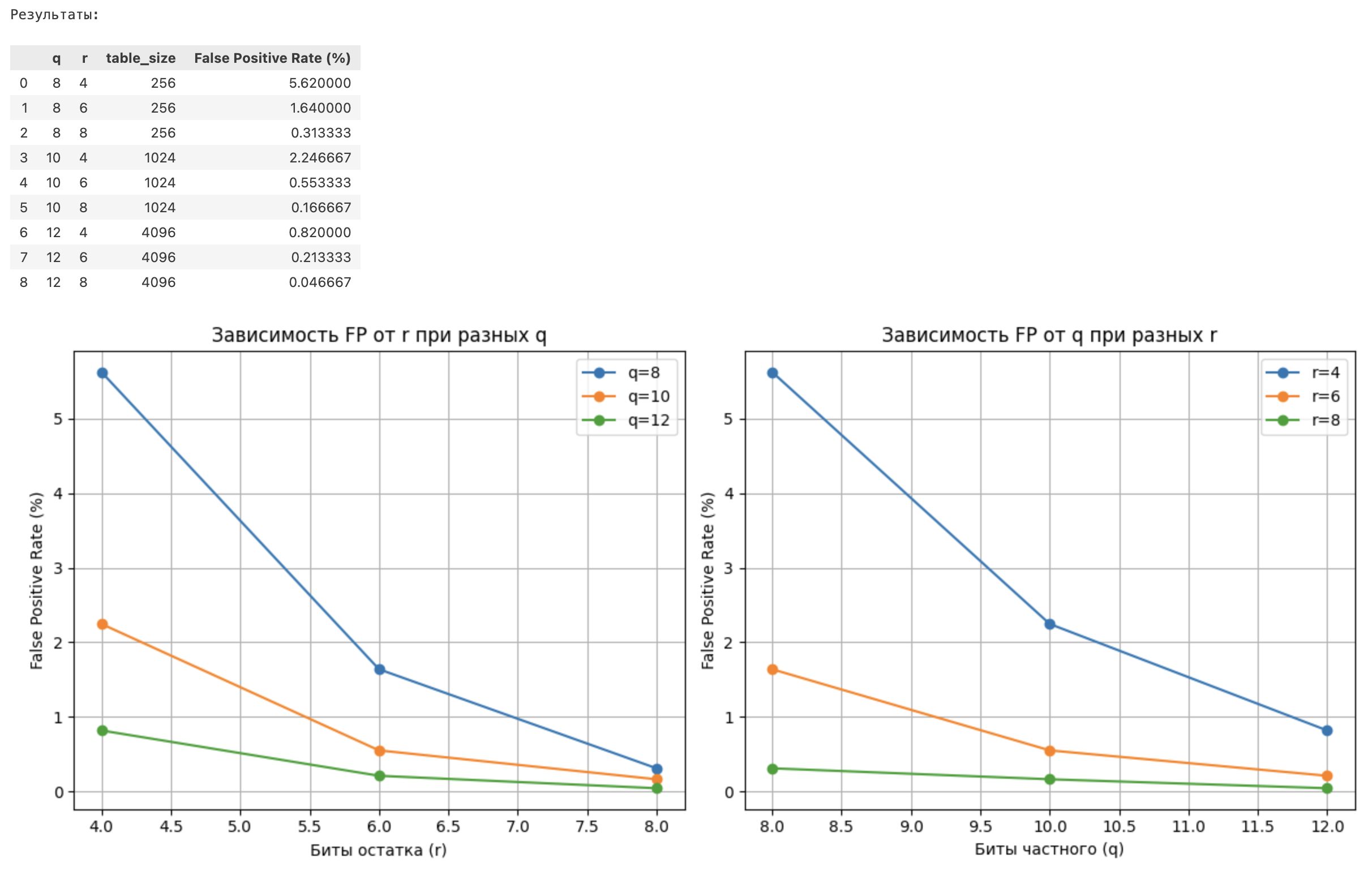


Рисунок 9 – работа программы для задания №4

**Задание №5 «Count-Min Sketch»**

1. Реализовать Count-Min Sketch.
2. Определить процент ложноположительных срабатываний конкретной реализации.
3. Оценить зависимость ложноположительных срабатываний от гиперпараметров алгоритма (таблица и графики зависимостей).

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

import matplotlib.pyplot as plt

import numpy as np

import random

class CountMinSketch:

def \_\_init\_\_(self, width, depth):

self.width = width

self.depth = depth

self.table = np.zeros((depth, width), dtype=np.int32)

self.seeds = [random.randint(0, 1024) for \_ in range(depth)]

def \_hash(self, item, seed):

return (hash(str(item)) ^ seed) % self.width

def update(self, item, count=1):

for i in range(self.depth):

pos = self.\_hash(item, self.seeds[i])

self.table[i][pos] += count

def estimate(self, item):

return min(self.table[i][self.\_hash(item, self.seeds[i])]

for i in range(self.depth))

random.seed(42)

data = [random.randint(1, 100) for \_ in range(10000)]

true\_counts = {}

for num in data:

true\_counts[num] = true\_counts.get(num, 0) + 1

widths = [50, 100, 200, 500, 1000]

depths = [3, 5, 7]

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

for depth in depths:

errors = []

for width in widths:

cms = CountMinSketch(width, depth)

for num in data:

cms.update(num)

sample\_size = min(100, len(true\_counts))

sample = random.sample(list(true\_counts.keys()), sample\_size)

total\_error = 0

for num in sample:

total\_error += cms.estimate(num) - true\_counts[num]

avg\_error = total\_error / sample\_size

errors.append(avg\_error)

plt.plot(widths, errors, marker='o', label=f'Depth={depth}')

plt.xlabel('Width of Sketch')

plt.ylabel('Average Overestimation Error')

plt.title('Count-Min Sketch: Error vs Width')

plt.legend()

plt.grid()

plt.subplot(1, 2, 2)

cms = CountMinSketch(200, 5)

for num in data:

cms.update(num)

sample\_size = min(200, len(true\_counts))

sample = random.sample(list(true\_counts.keys()), sample\_size)

errors = []

for num in sample:

errors.append(cms.estimate(num) - true\_counts[num])

plt.hist(errors, bins=30, alpha=0.7)

plt.grid()

plt.tight\_layout()

plt.show()

Пример работы программы представлен на рисунке 10.

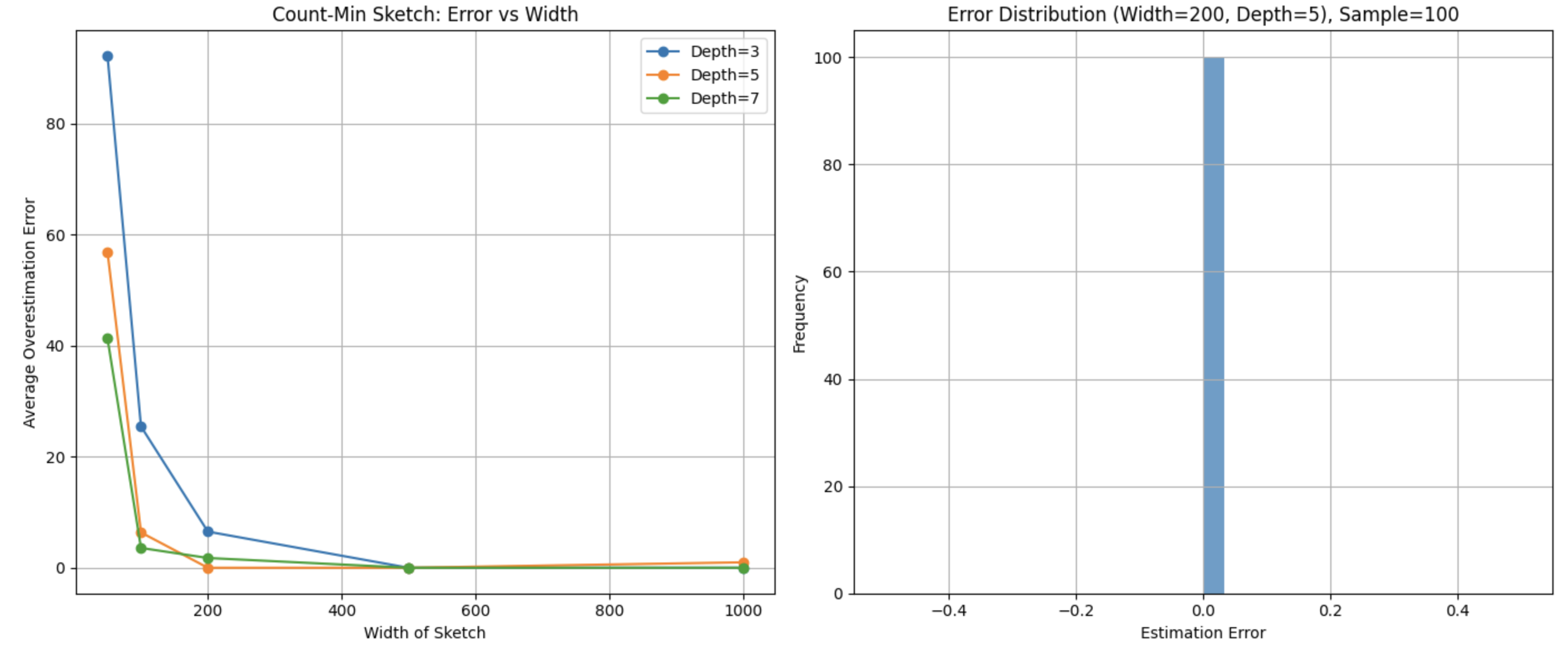


Рисунок 10 – работа программы для задания №5

import random

import matplotlib.pyplot as plt

words = ["apple", "banana", "cherry", "date", "elderberry", "fig", "grape"]

test\_data = [random.choice(words) for \_ in range(1000)]

cms = CountMinSketch(width=50, depth=3)

for word in test\_data:

cms.update(word)

exact\_counts = {}

for word in test\_data:

if word not in exact\_counts:

exact\_counts[word] = 0

exact\_counts[word] += 1

print("{:<10} {:<10} {:<10}".format("Word", "Estimate", "Real"))

print("-" \* 30)

for word in words:

estimated = cms.estimate(word)

real = exact\_counts.get(word, 0)

print("{:<10} {:<10} {:<10}".format(word, estimated, real))

errors = [cms.estimate(word) - exact\_counts.get(word, 0) for word in words]

plt.figure(figsize=(10, 4))

plt.bar(words, errors, color='skyblue')

plt.axhline(0, color='red', linestyle='--', linewidth=1)

plt.grid(axis='y', alpha=0.3)

Пример работы программы представлен на рисунке 11.

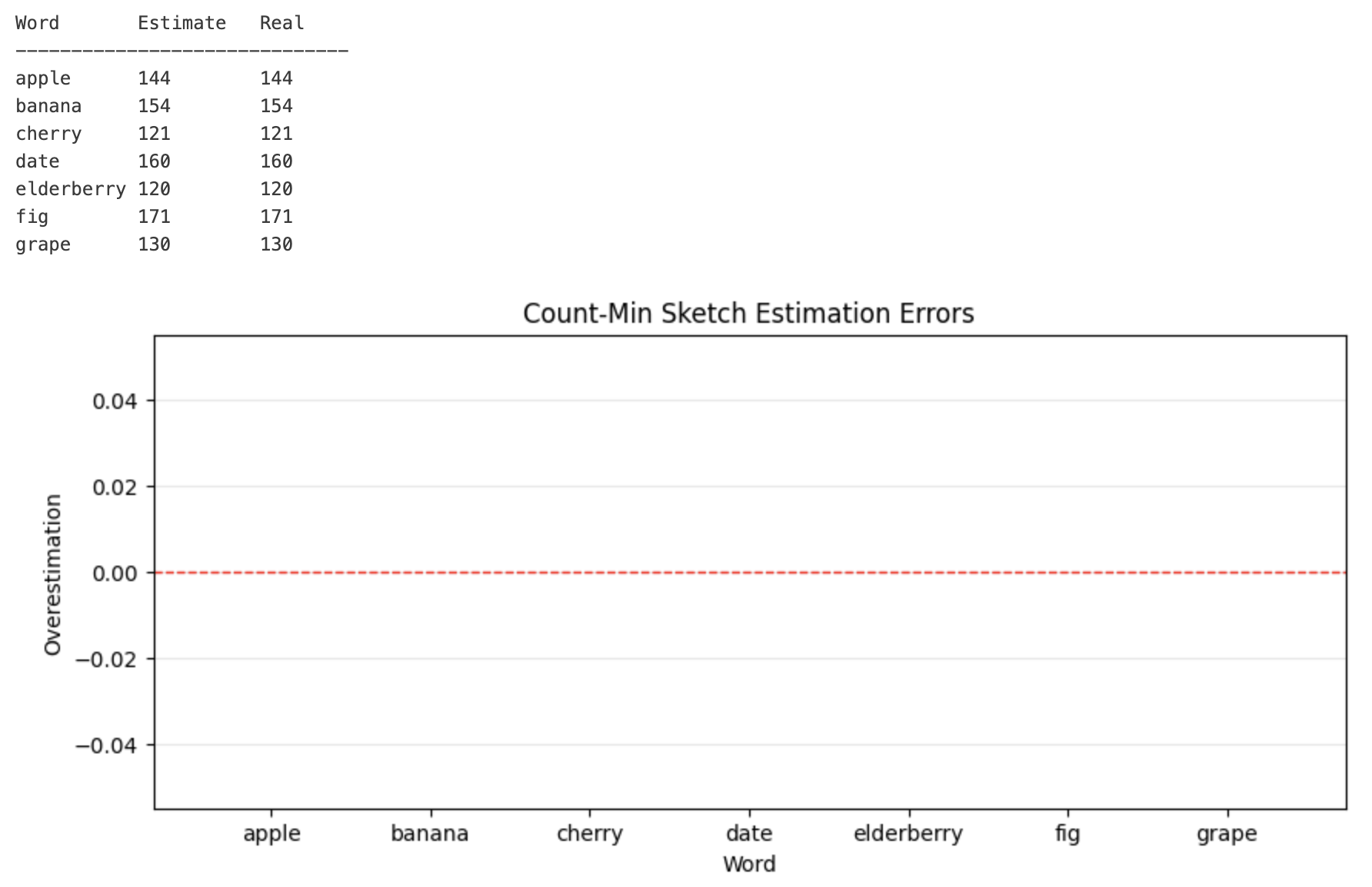


Рисунок 11 – работа программы для задания №5

**Задание №6 «MinHash»**

1. Реализовать MinHash.
2. Определить процент ложноположительных срабатываний конкретной реализации.
3. Оценить зависимость ложноположительных срабатываний от гиперпараметров алгоритма (таблица и графики зависимостей).

**Решение**

Реализация кода для выполнения данной задачи на языке программирования Python представлен ниже:

import matplotlib.pyplot as plt

import numpy as np

import random

class MinHash:

def \_\_init\_\_(self, num\_hashes):

self.num\_hashes = num\_hashes

self.a\_params = [random.randint(1, 2\*\*32) for \_ in range(num\_hashes)]

self.b\_params = [random.randint(1, 2\*\*32) for \_ in range(num\_hashes)]

self.min\_values = np.full(num\_hashes, np.inf)

def \_hash(self, item, a, b):

item\_hash = hash(str(item)) & 0xFFFFFFFF

return (a \* item\_hash + b) & 0xFFFFFFFF

def update(self, item):

for i in range(self.num\_hashes):

h = self.\_hash(item, self.a\_params[i], self.b\_params[i])

if h < self.min\_values[i]:

self.min\_values[i] = h

def jaccard(self, other):

return np.mean(self.min\_values == other.min\_values)

random.seed(42)

data1 = set(random.sample(range(1, 10001), 1000))

data2 = set(random.sample(range(5000, 15001), 1000))

intersection = len(data1 & data2)

union = len(data1 | data2)

true\_jaccard = intersection / union

num\_hashes\_list = [10, 50, 100, 200, 500]

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

errors = []

for num\_hashes in num\_hashes\_list:

mh1 = MinHash(num\_hashes)

mh2 = MinHash(num\_hashes)

for item in data1:

mh1.update(item)

for item in data2:

mh2.update(item)

estimated\_jaccard = mh1.jaccard(mh2)

error = abs(estimated\_jaccard - true\_jaccard)

errors.append(error)

print(f"Хеш-функций: {num\_hashes}, Оценка: {estimated\_jaccard:.4f}, Истинное: {true\_jaccard:.4f}, Ошибка: {error:.4f}")

plt.plot(num\_hashes\_list, errors, marker='o', color='blue')

plt.grid(True)

plt.subplot(1, 2, 2)

num\_trials = 100

num\_hashes = 100

estimates = []

for \_ in range(num\_trials):

mh1 = MinHash(num\_hashes)

mh2 = MinHash(num\_hashes)

for item in data1:

mh1.update(item)

for item in data2:

mh2.update(item)

estimates.append(mh1.jaccard(mh2))

Пример работы программы представлен на рисунке 12.

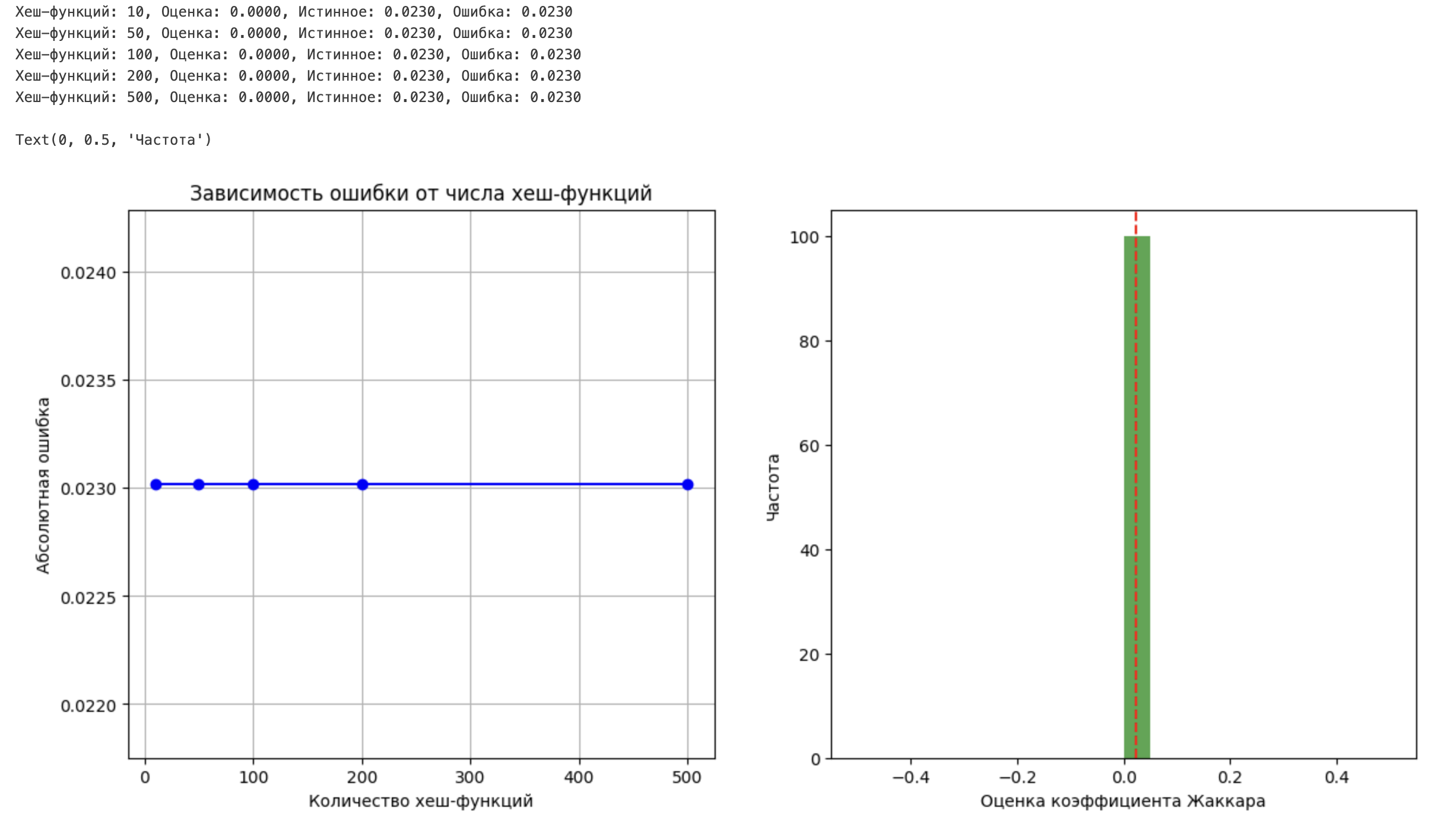


Рисунок 12 – работа программы для задания №6

import matplotlib.pyplot as plt

import numpy as np

import random

set\_A = {'яблоко', 'банан', 'апельсин', 'груша', 'киви', 'манго', 'ананас'}

set\_B = {'банан', 'апельсин', 'виноград', 'арбуз', 'киви', 'персик', 'манго'}

intersection = set\_A & set\_B

only\_A = set\_A - set\_B

only\_B = set\_B - set\_A

true\_jaccard = len(intersection) / len(set\_A | set\_B)

mh\_A = MinHash(100)

mh\_B = MinHash(100)

for item in set\_A:

mh\_A.update(item)

for item in set\_B:

mh\_B.update(item)

estimated\_jaccard = mh\_A.jaccard(mh\_B)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

plt.barh(['Только в A', 'Пересечение', 'Только в B'],

[len(only\_A), len(intersection), len(only\_B)],

color=['blue', 'purple', 'green'])

plt.title(f"Сравнение множеств\nИстинный коэффициент Жаккара = {true\_jaccard:.2f}")

plt.xlabel("Количество элементов")

plt.grid(axis='x')

plt.subplot(1, 2, 2)

num\_hashes\_range = range(10, 501, 50)

errors = []

for num\_hashes in num\_hashes\_range:

mh1 = MinHash(num\_hashes)

mh2 = MinHash(num\_hashes)

for item in set\_A:

mh1.update(item)

for item in set\_B:

mh2.update(item)

errors.append(abs(mh1.jaccard(mh2) - true\_jaccard))

plt.plot(num\_hashes\_range, errors, marker='o', color='red')

plt.xlabel('Количество хеш-функций')

plt.ylabel('Абсолютная ошибка')

plt.title(f'Точность MinHash\nОценка: {estimated\_jaccard:.2f} vs Истинное: {true\_jaccard:.2f}')

plt.grid(True)

plt.tight\_layout()

plt.show()

print("Элементы только в A:", only\_A)

print("Элементы только в B:", only\_B)

print("Общие элементы:", intersection)

print(f"Истинный коэффициент Жаккара: {true\_jaccard:.4f}")

print(f"Оценка MinHash: {estimated\_jaccard:.4f}")

print(f"Абсолютная ошибка: {abs(estimated\_jaccard - true\_jaccard):.4f}")

Пример работы программы представлен на рисунке 13.

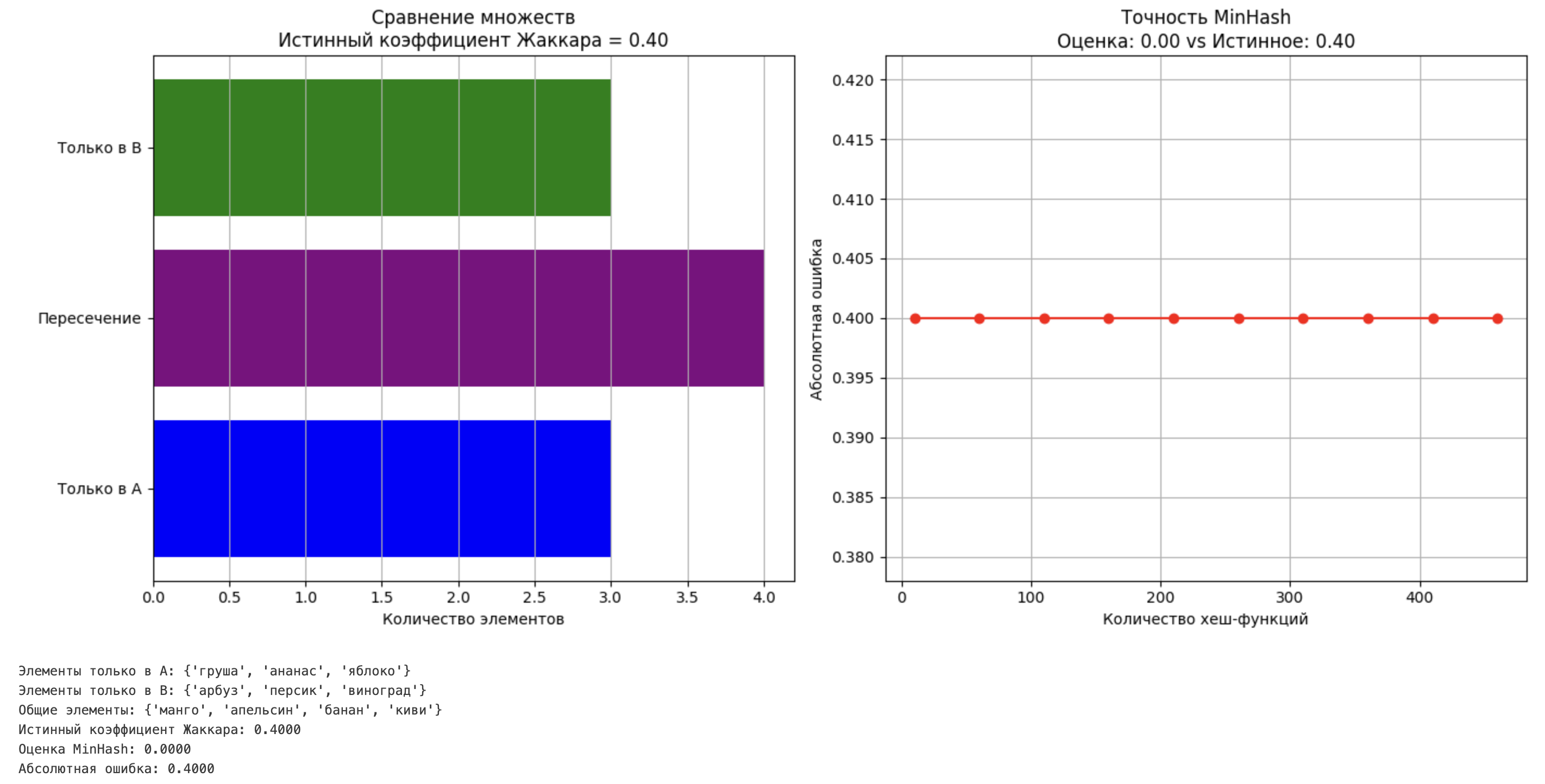


Рисунок 13 – работа программы для задания №6