



▼ Лабораторная работа #1: хэширование

Выполнили: Смольникова Полина, Макридин Максим

Рассмотренные алгоритмы:

- хэширование цепочками 
- хэширование по методу открытой адресации
 - Линейное перехэширование
 - Двоичное перехэширование
 - Квадратичное перехэширование
- метод кукушки 

Алгоритмы написаны на C++

Юнит-тесты реализованы с помощью Google Tests Тестирование производительности производилось на таблице 4n размера относительно кол-ва используемых элементов размера n.

Данные производительности записывались в .csv-файлы и визуализировались в Python

[Ссылка на репозиторий с кодом, тестами и маленьким CI](#)

▼ int-ы

$$h_{a,b}(x) = ((ax + b) \bmod 2^w) \div 2^{w-M}$$

w - размер машинного слова, $w = 32, 64, 128$

$m = 2^M$ - размер таблицы

$a \in \{0, 1, \dots, 2^w - 1\}$, нечетное

$b \in \{0, \dots, 2^{w-M} - 1\}$

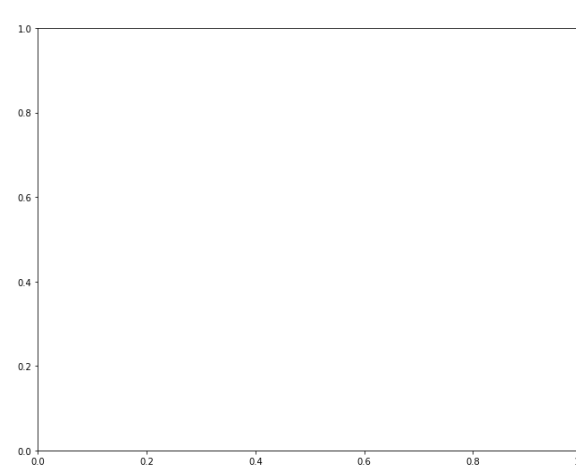
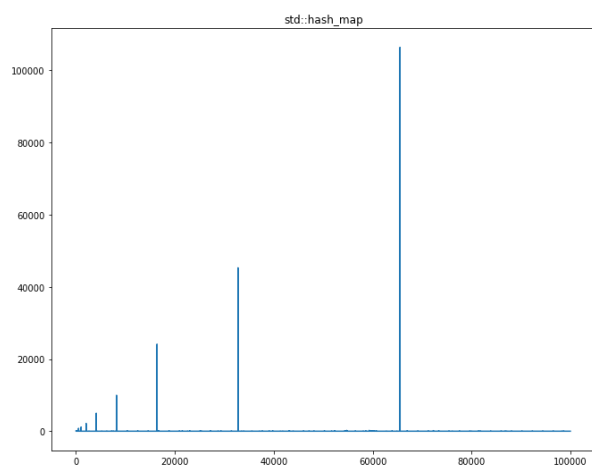
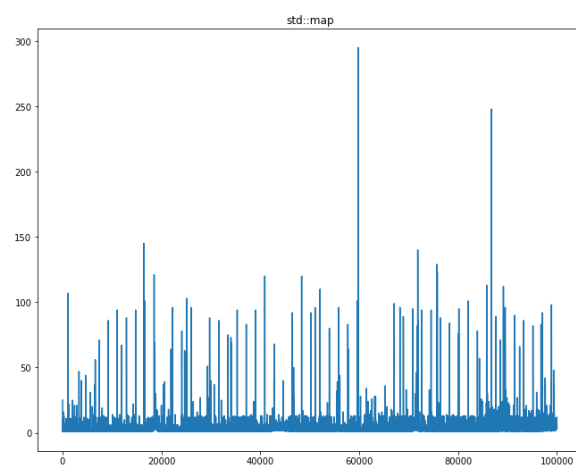
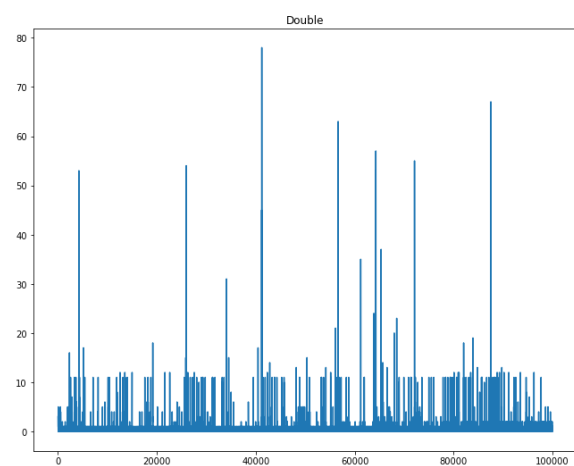
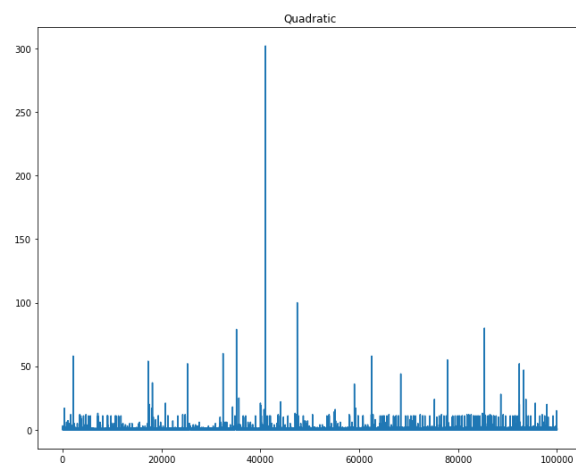
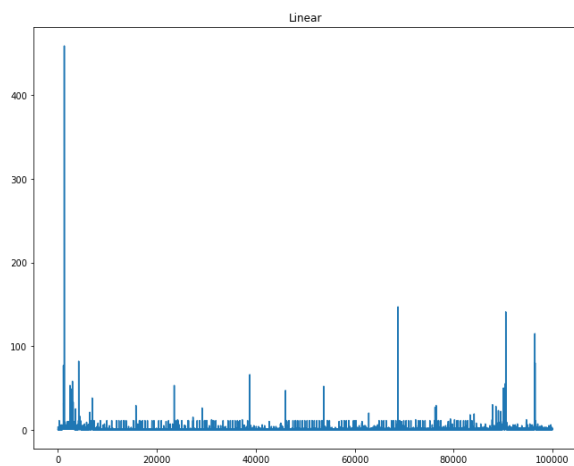
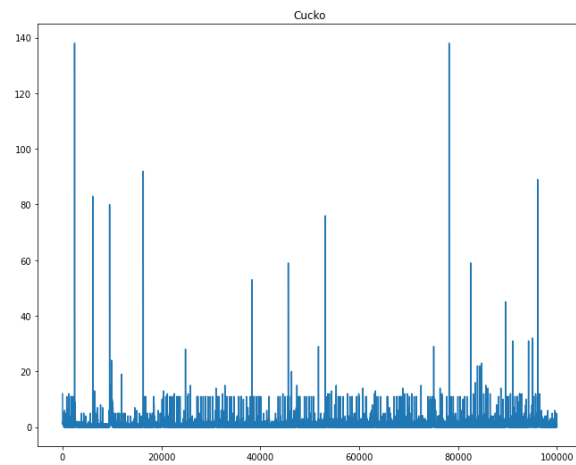
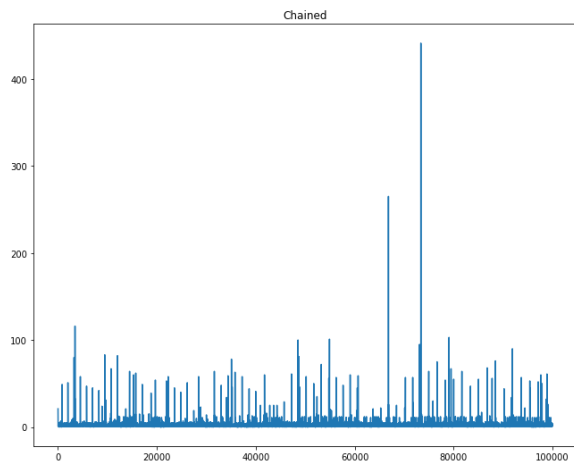
▼ Вставка int-ов

```
1 import csv
2 import pprint
3 pp = pprint.PrettyPrinter(indent=4)
4
5
6 hash_insert_int = {}
7 with open('hash_insert_int.csv') as csvfile:
8     reader = csv.reader(csvfile, delimiter=',')
9     for row in reader:
10         key = row[0]
11         hash_insert_int[key] = list(map(int, row[1:]))
12         print(len(row))
13
```

```
13
14 keys = list(hash_insert_int.keys())
15 indices = [i for i in range(len(hash_insert_int[keys[0]]))]
16 # for key in hash_insert_int:
17     # print(f'{key}: {hash_insert_int[key]}')
18
```

```
100001
100001
100001
100001
100001
100001
100001
100001
```

```
1 import matplotlib.pyplot as plt
2
3 fig, plots = plt.subplots(4, 2, figsize=(24, 40))
4 type(plots[2][1])
5 for i in range(7):
6     plots[i // 2][i % 2].set_title(keys[i])
7     plots[i // 2][i % 2].plot(indices, hash_insert_int[keys[i]])
```

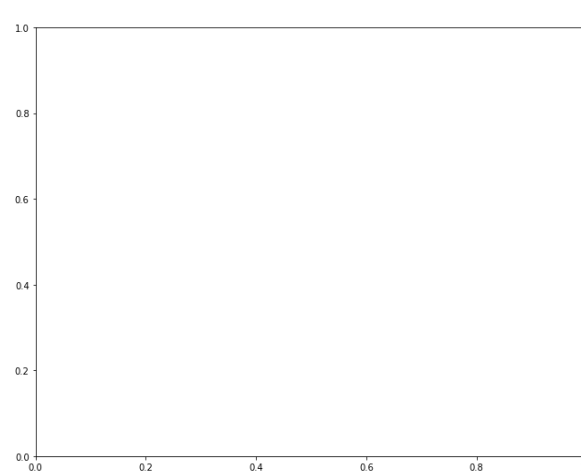
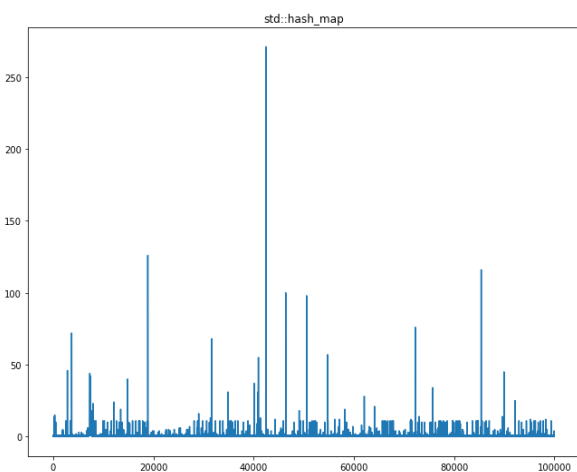
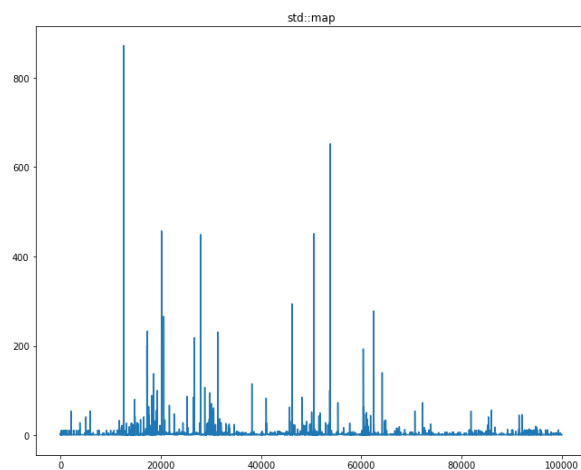
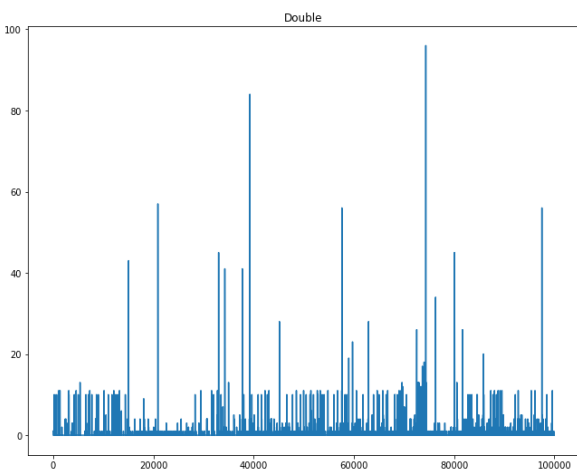
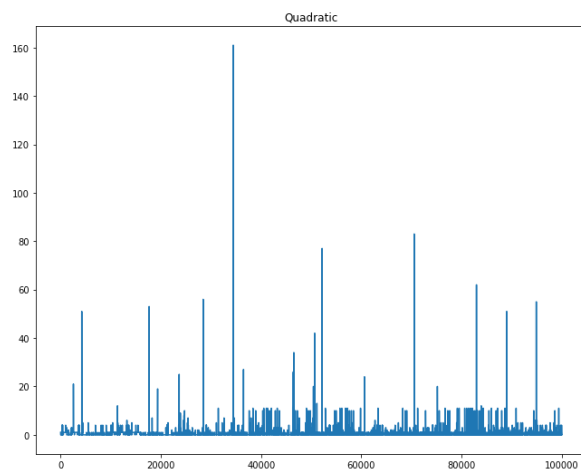
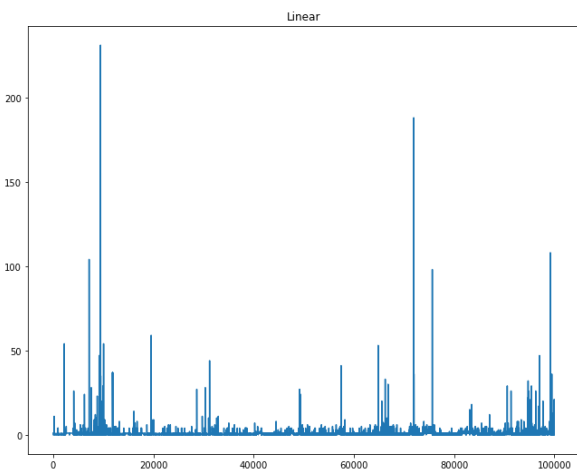
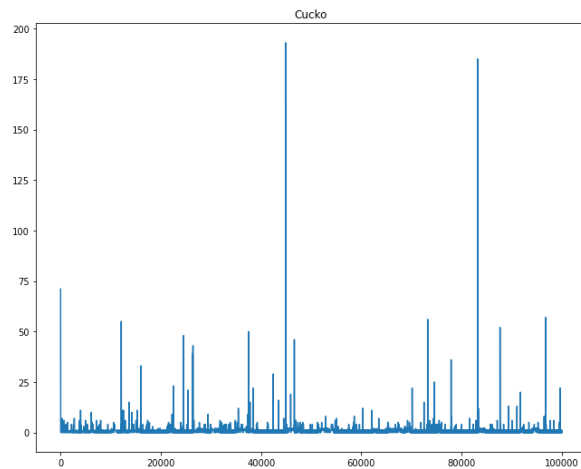
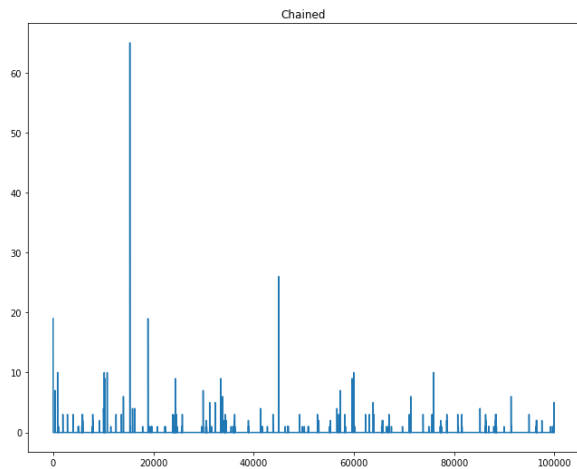


▼ Поиск int-ов

```
1  import csv
2  import pprint
3  pp = pprint.PrettyPrinter(indent=4)
4
5
6  hash_find_int = {}
7  with open('hash_find_int.csv') as csvfile:
8      reader = csv.reader(csvfile, delimiter=',')
9      for row in reader:
10         key = row[0]
11         hash_find_int[key] = list(map(int, row[1:]))
12         print(len(row))
13  keys = list(hash_find_int.keys())
14  indices = [i for i in range(len(hash_find_int[keys[0]]))]
15
16  # for key in hash_insert_int:
17      # print(f'{key}: {hash_insert_int[key]}')
18
100001
100001
100001
```

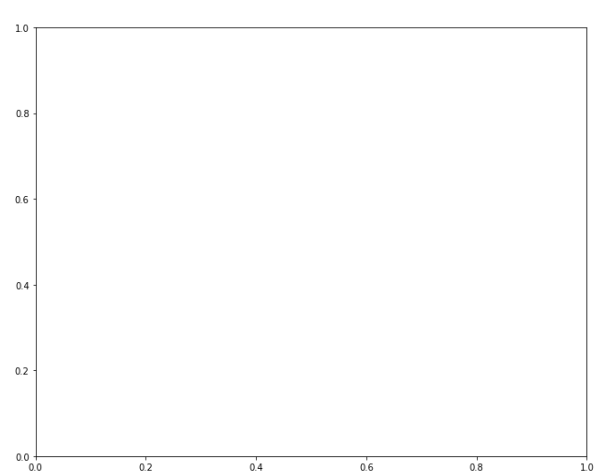
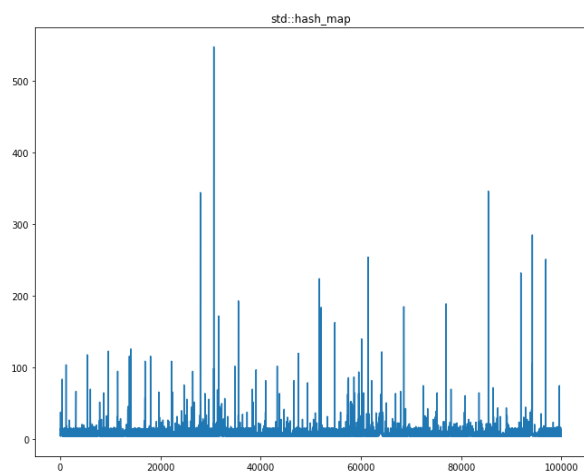
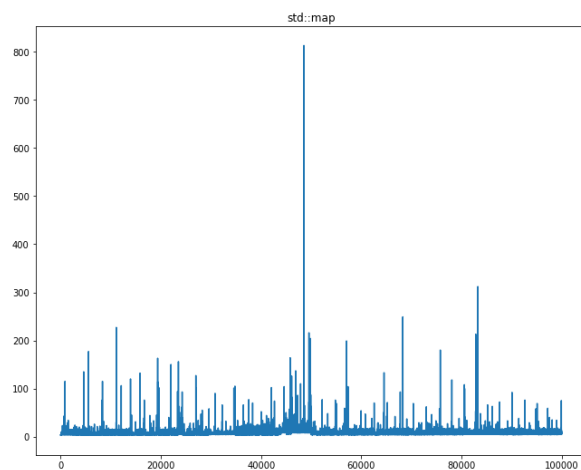
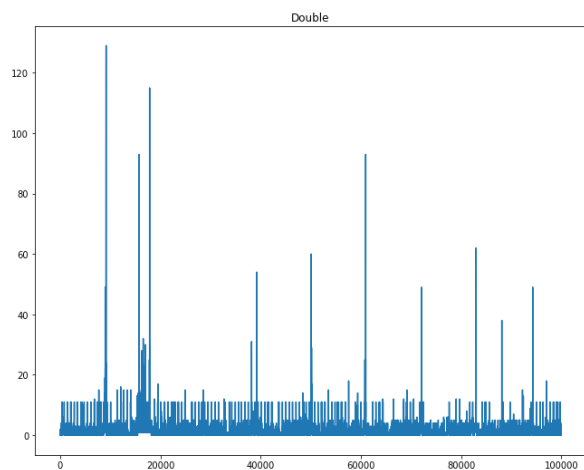
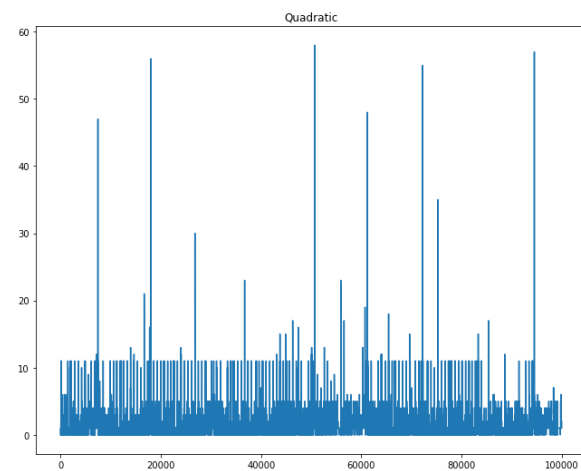
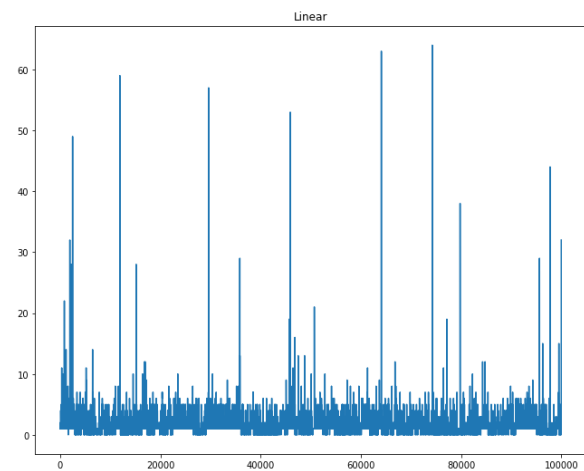
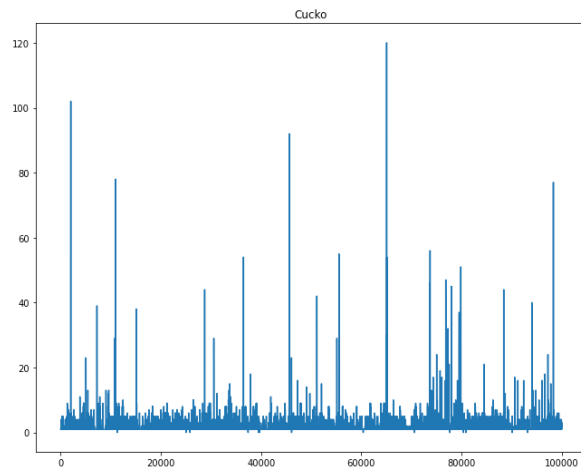
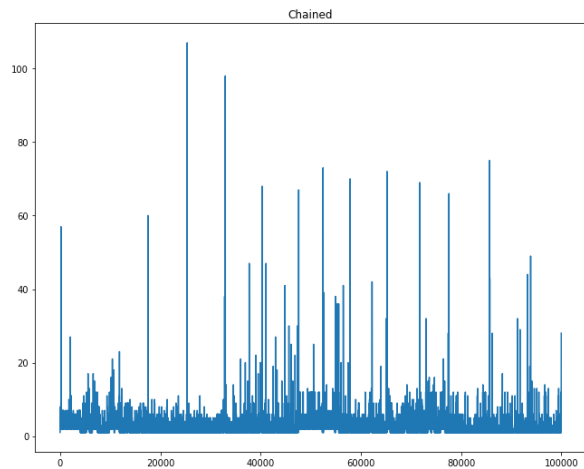
100001
100001
100001
100001

```
1  fig, plots = plt.subplots(4, 2, figsize=(24, 40))
2  type(plots[2][1])
3  for i in range(7):
4      plots[i // 2][i % 2].set_title(keys[i])
5      plots[i // 2][i % 2].plot(indices, hash_find_int[keys[i]])
```



▼ Удаление int-ов

```
1  import csv
2  import pprint
3  pp = pprint.PrettyPrinter(indent=4)
4
5
6  hash_erase_int = {}
7  with open('hash_erase_int.csv') as csvfile:
8      reader = csv.reader(csvfile, delimiter=',')
9      for row in reader:
10         key = row[0]
11         hash_erase_int[key] = list(map(int, row[1:]))
12         # print(len(row))
13  keys = list(hash_erase_int.keys())
14  indices = [i for i in range(len(hash_erase_int[keys[0]]))]
15
16  fig, plots = plt.subplots(4, 2, figsize=(24, 40))
17  type(plots[2][1])
18  for i in range(7):
19      plots[i // 2][i % 2].set_title(keys[i])
20      plots[i // 2][i % 2].plot(indices, hash_erase_int[keys[i]])
```



▼ `std::string-n`

$$x = (x_0, x_1, \dots, x_{s-1})$$

$$h_a(x) = (\sum(a^i x_i) \bmod p) \bmod m$$

$$a \in \{0, 1, \dots, p-1\}$$

p - большое простое число, m - размер таблицы

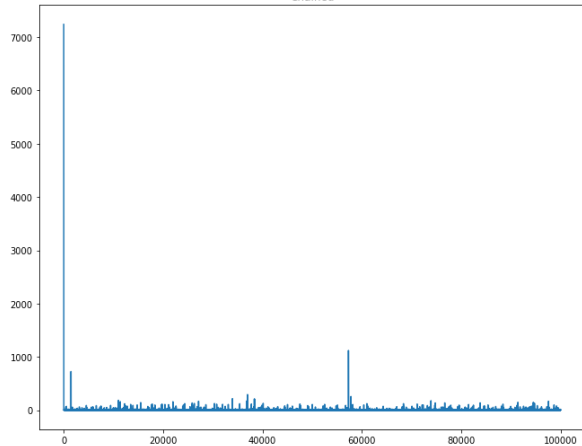
▼ Вставка std::string-ов

```

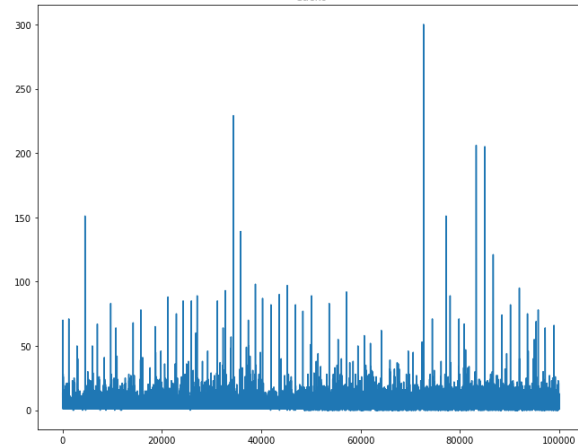
1  import csv
2  import pprint
3  pp = pprint.PrettyPrinter(indent=4)
4
5
6  hash_insert_string = {}
7  with open('hash_insert_string.csv') as csvfile:
8      reader = csv.reader(csvfile, delimiter=',')
9      for row in reader:
10         key = row[0]
11         hash_insert_string[key] = list(map(int, row[1:]))
12         # print(len(row))
13  keys = list(hash_insert_string.keys())
14  indices = [i for i in range(len(hash_insert_string[keys[0]]))]
15
16  fig, plots = plt.subplots(4, 2, figsize=(24, 40))
17  type(plots[2][1])
18  for i in range(7):
19      plots[i // 2][i % 2].set_title(keys[i])
20      plots[i // 2][i % 2].plot(indices, hash_insert_string[keys[i]])

```

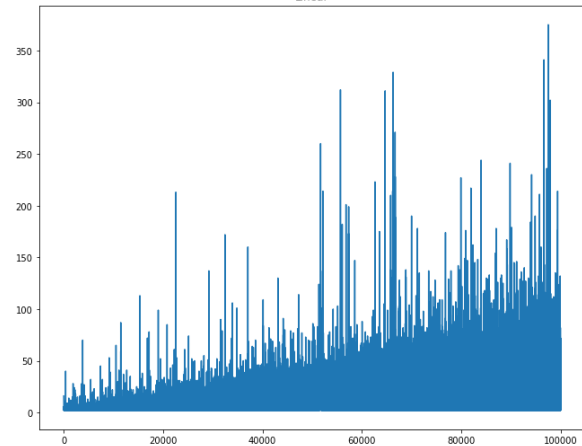
Chained



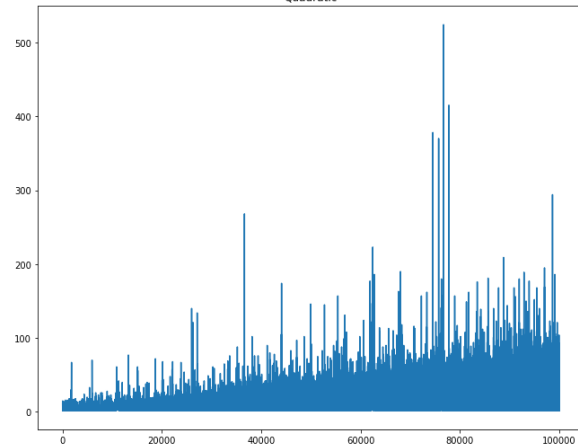
Cucko



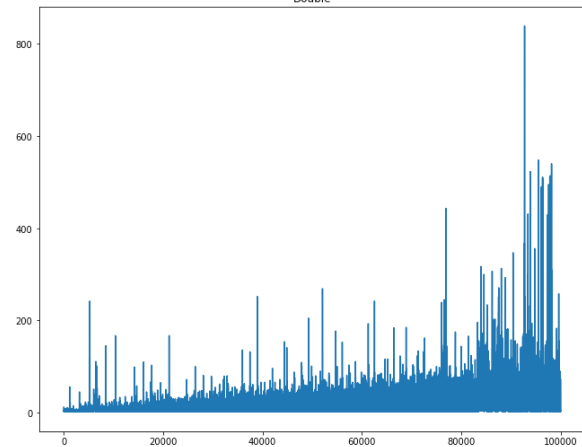
Linear



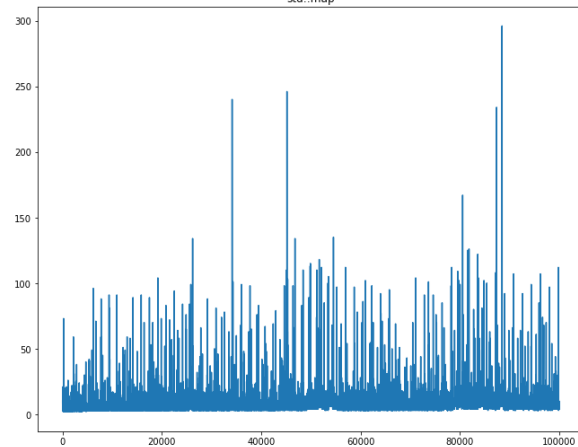
Quadratic



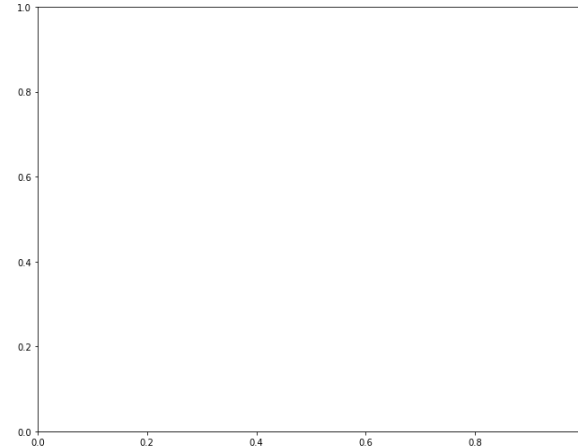
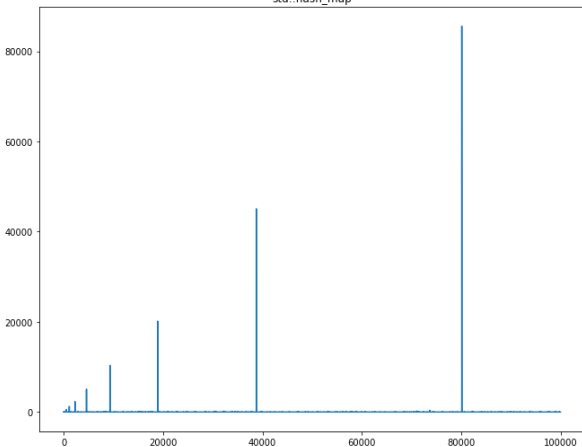
Double



std::map



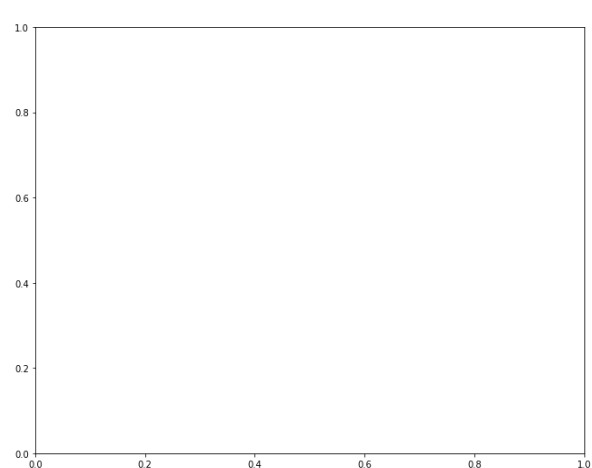
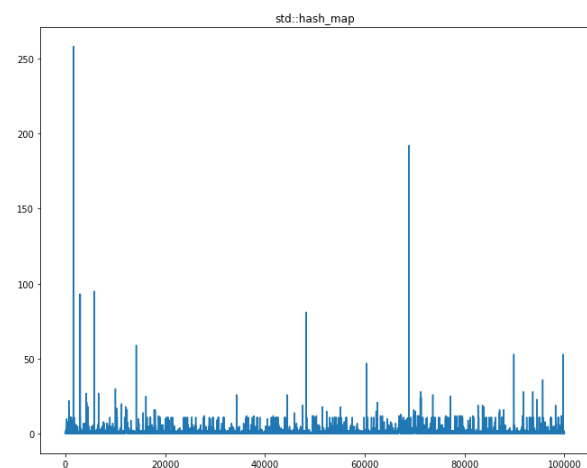
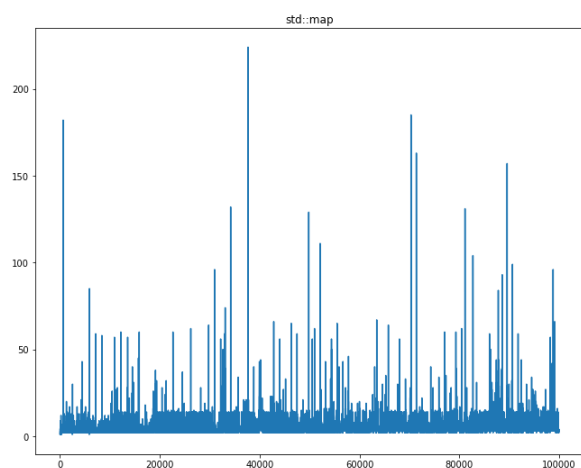
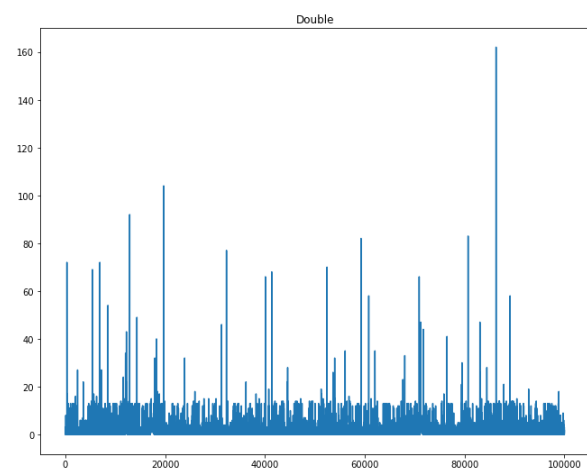
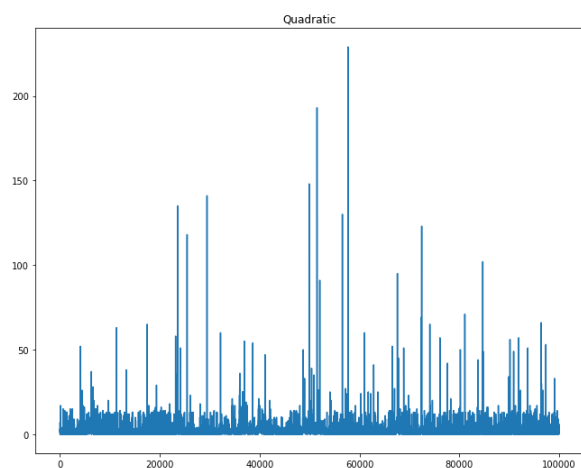
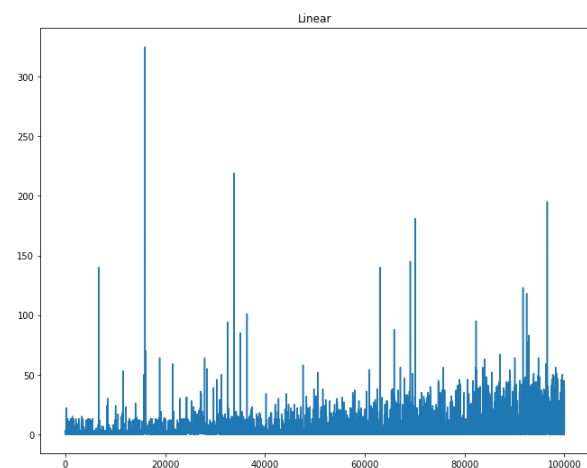
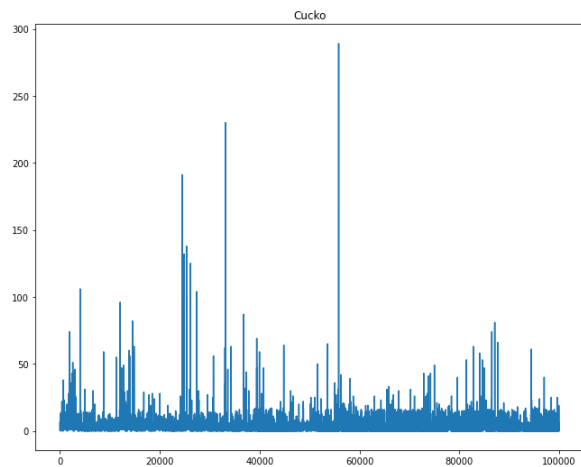
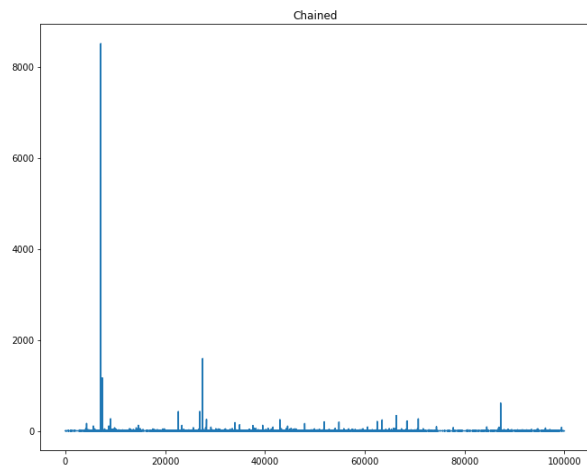
std::hash_map



▼ поиск std::string-ов

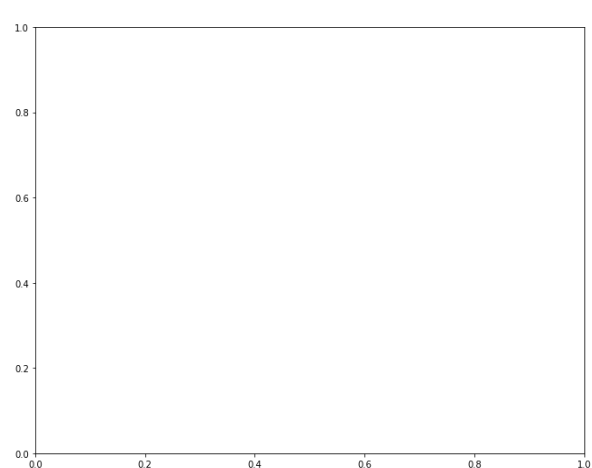
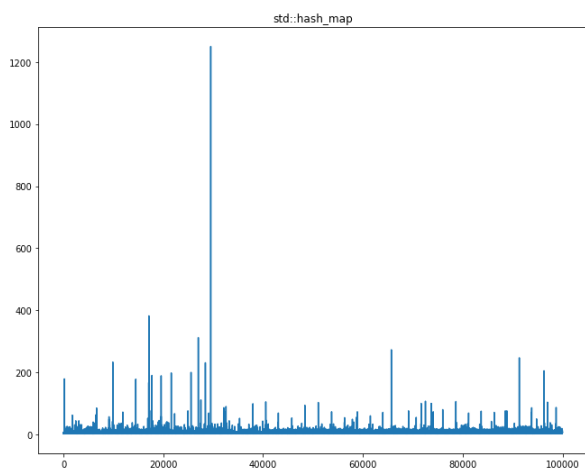
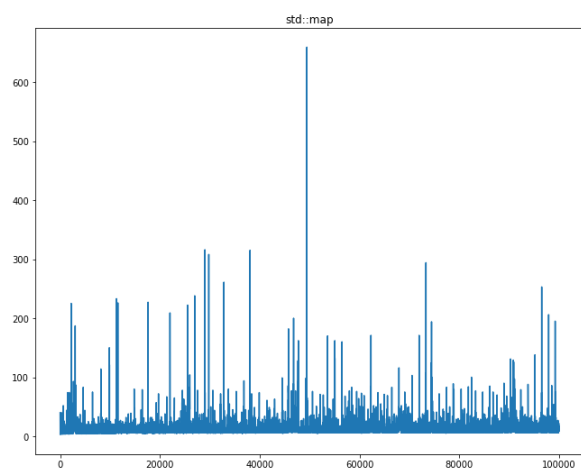
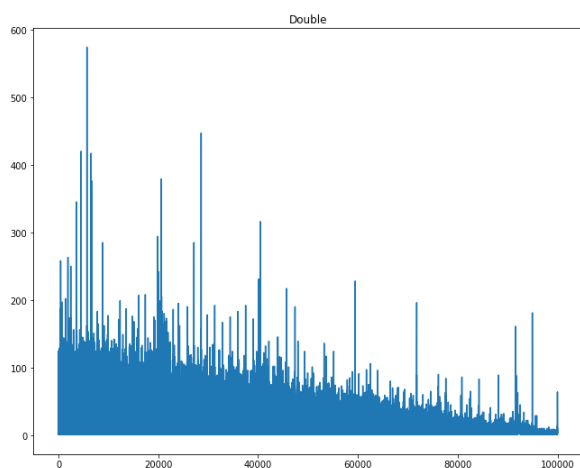
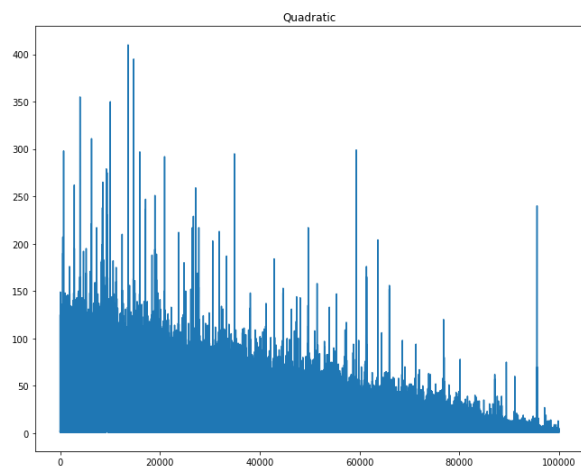
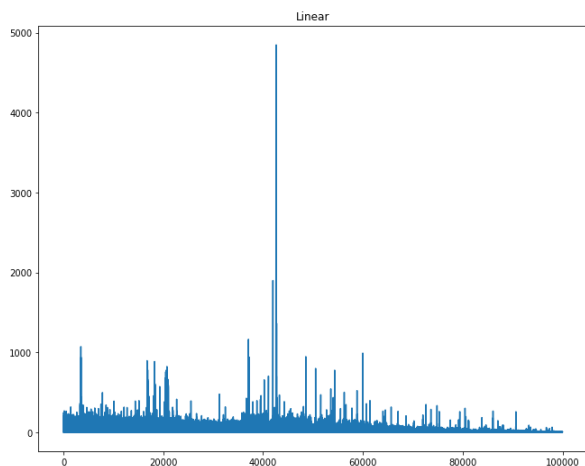
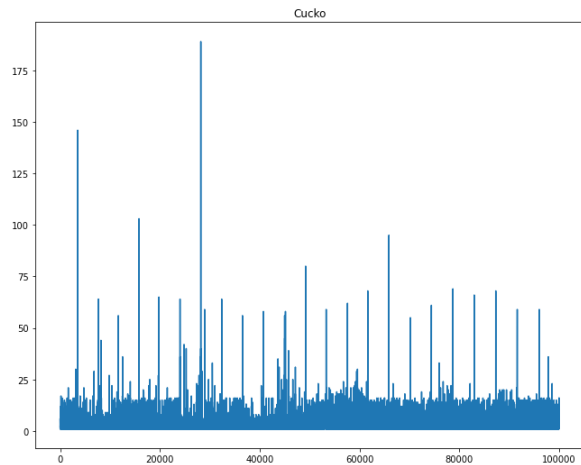
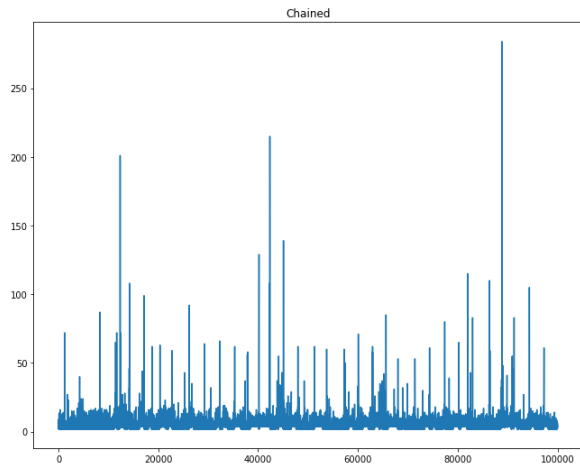
```
1  import csv
2  import pprint
3  pp = pprint.PrettyPrinter(indent=4)
4
5
6  hash_find_string = {}
7  with open('hash_find_string.csv') as csvfile:
8      reader = csv.reader(csvfile, delimiter=',')
9      for row in reader:
10         key = row[0]
11         hash_find_string[key] = list(map(int, row[1:]))
12         # print(len(row))
13  keys = list(hash_find_string.keys())
14  indices = [i for i in range(len(hash_find_string[keys[0]]))]
15
```

```
15
16 fig, plots = plt.subplots(4, 2, figsize=(24, 40))
17 type(plots[2][1])
18 for i in range(7):
19     plots[i // 2][i % 2].set_title(keys[i])
20     plots[i // 2][i % 2].plot(indices, hash_find_string[keys[i]])
```



▼ Удаление std::string-ов

```
1  import csv
2  import pprint
3  pp = pprint.PrettyPrinter(indent=4)
4
5
6  hash_erase_string = {}
7  with open('hash_erase_string.csv') as csvfile:
8      reader = csv.reader(csvfile, delimiter=',')
9      for row in reader:
10         key = row[0]
11         hash_erase_string[key] = list(map(int, row[1:]))
12         # print(len(row))
13  keys = list(hash_erase_string.keys())
14  indices = [i for i in range(len(hash_erase_string[keys[0]]))]
15
16  fig, plots = plt.subplots(4, 2, figsize=(24, 40))
17  type(plots[2][1])
18  for i in range(7):
19      plots[i // 2][i % 2].set_title(keys[i])
20      plots[i // 2][i % 2].plot(indices, hash_erase_string[keys[i]])
```



▼ Выводы

- `std::hash_map` не зря в `std`, он крайне хорош при вставке и поиске
- Кукушка удаляет лучше `std::hash_map`

- Хэшировать строки - ужасно медленно, если есть способ этого избежать в пракических задачах - этого нужно избегать
- Писать на C++ очень больно, важно иметь Rust или/и обратно в Питон :/