**Теоретична частина**

**Хеш-функція**, або **геш-функція** — [функція](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), що перетворює вхідні дані будь-якого (як правило великого) розміру в дані фіксованого розміру.

**Хешування** (гешування, [англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *hashing*) — перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються **хеш-функціями**, або **функціями згортання**, а їхні результати називають хешем, **хеш-кодом**, **хеш-сумою**, або **дайджестом повідомлення** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *message digest*).

Хеш-функція використовується зокрема у структурах даних — [хеш-таблицях](https://uk.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8F), широко вживаних у програмному забезпеченні для швидкого пошуку даних. Хеш-функції використовуються для оптимізації таблиць та баз даних за рахунок того, що у однакових записів однакові значення хеш-функції. Такий підхід пошуку дублікатів ефективний у файлах великого розміру. Прикладом цього буде знаходження подібних ділянок у послідовностях [ДНК](https://uk.wikipedia.org/wiki/%D0%94%D0%9D%D0%9A). [Криптографічна геш-функція](https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%87%D0%BD%D0%B0_%D0%B3%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F) дозволяє легко перевірити, що деякі вхідні дані зіставляються із заданим значенням хешу, але, якщо вхідні дані невідомі, то навмисно важко відновити вхідне значення (або еквівалентну альтернативу), знаючи збережене значення хеш-функції. Це використовується для забезпечення [цілісності](https://uk.wikipedia.org/wiki/%D0%A6%D1%96%D0%BB%D1%96%D1%81%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97) переданих даних, і є будівельним блоком для [HMACs](https://uk.wikipedia.org/wiki/HMAC" \o "HMAC), які забезпечують [аутентифікацію повідомлень](https://uk.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F_%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D1%8C).

Хеш-функції пов'язані (і їх часто плутають) з [контрольною сумою](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0_%D1%81%D1%83%D0%BC%D0%B0), контрольними цифрами, [відбитками пальців](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%B1%D0%B8%D1%82%D0%BA%D0%B8_%D0%BF%D0%B0%D0%BB%D1%8C%D1%86%D1%96%D0%B2_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [рандомізацією функцій](https://uk.wikipedia.org/wiki/%D0%A0%D0%B0%D0%BD%D0%B4%D0%BE%D0%BC%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97" \o "Рандомізація функції), кодами, що виправляють помилки, і з шифрами. Хоча ці поняття певною мірою збігаються, кожен з них має свою власну область застосування і вимоги і є розробленим і оптимізованим по-різному.

Вимоги до хеш функції:

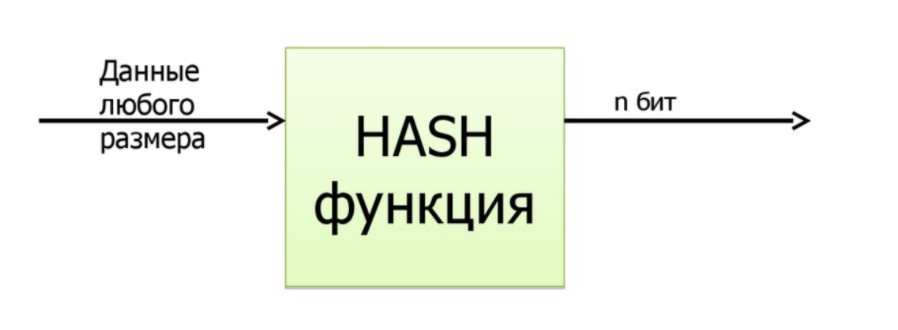
1. хеш-функція Н повинна застосовуватись до блоку даних довільної

довжини.

2. хеш-функція Н створює вихід фіксованої довжини.

3. Н (М) відносно легко (за поліноміальний час) обчислюється для

довільного значення М.

4. Для довільного даного значення хеш-кода h обчислювально неможливо знайти M таке, що Н (M) = h. 

5. Для довільного даного х обчислювально неможливо знайти таке y != x, що H (y) = H (x). Таку властивість називають слабим опором колізіям.

Колізією називається співпадіння дайджестів для різних даних.

6. Обчислювально неможливо знайти довільну пару (х, y) таку, що

H (y) = H (x). Цю властивість називають сильним опором колізіям.

**Хеш-таблиця** - це структура даних, яка впроваджує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і виконувати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення пари по ключу.

**Постановка задачі**

Запропонувати свій варіант алгоритму побудови хеш-коду. Дотримуватись вимог до хеш-функції. Обгрунтувати, навести приклад. Надати схему алгоритму (словесну, блок-схему або будь-яку іншу)

**Хід роботи**

Для виконання цієї домашньої роботи створив свій власний алгорим шифрування його сутність полягає у ось таких діях:

**Алгоритм хешування:**

1. Першим ділом переводимо повідомлення у юнікод
2. Знаходимо середнє арифметичне(записуємо цілим числом)
3. Якщо елементи не однакової довжини добавляємо свого роду «соль» щоб зрівняти їхню довжину(використовуючи середнє арифметичне)
4. Знаходимо суму результуючих значень із кроку 3 і ділим його на суму кожного четвертого елемента починаючи із першого не враховуючи нулі (записуємо цілим числом)
5. Знаходимо *mod* віл *e*(стала Ейлера «2.71…»)
6. Беремо перші 15 символів після коми і записуємо їх, якщо першою цифрою виступає 0 змінюємо його на першу цифру значення із 4 кроку
7. Отримане значення остаточно шифруємо записуючи його у 36-ній системі

**Приклад:**

1. Візьмем слово Wiki і спробуємо захешувати його. Для кожного символу знаходимо його значення у юнікоді:

W = 87

i = 105

k = 107

i = 105

1. Середнє арифметичне рівняється (87 + 105 +107 +105) // 4 = 404 // 4 = 101
2. Потім бачим що не всі елементи одної довжини 87(2 цифри), 105(3 цифри) , 107(3 цифри). Добавим у кінець 87 перший символ із середнього арифметичного(101 => index[0] = 1 ) і у нас вийде 871
3. Знаходимо суму всіх 4 елементів не враховуючи нулі **8**71, 10**5,** 107, 10**5**

**8 + 5 +5 = 18** після цього знаходимо суму всіх елементів із кроку 3 і розділяємо на отриману раніше суму (871 + 105 +107 +105) // 18 =

1188 // 18 = 66

1. Знаходимо остачу від ділення на число Ейлера: 66 *mod e* = 0.7612361169829178
2. Виписуємо перші 15 символів після коми 7612361169829178 але у нашу випадку на початку не стоїть 0 тому (але якщо б стояв ми його замінюємо на 1 перша цифра значення і кроку 4 яке рівняється **1**8) у нас виходить 7612361169829178
3. І нарешті записуємо обраховане значення у 36-системі (base 36) і воно рівнятиметься 7612361169829178 == 6NNYWDESFN

**Обґрунтування:**

Я створив саме такий алгоритм хешування тому що він є доволі простий і швидко виконується також він відповідає всім вимогам до хеш-функції у тестування і об’ясню чому.

**Тестування**

Хоча перед мною не стояла задача зробити саму хеш функцію а лише її алгоритм но для кращого зрозуміння і перевірки я реалізував її на мові програмування Python сам код думаю не потребує об’яснення по скільки він відтворює вище описаний алгоритм(*примітка:* у архіві із звітом знаходиться файл Hash.exe)

*Лістинг:*

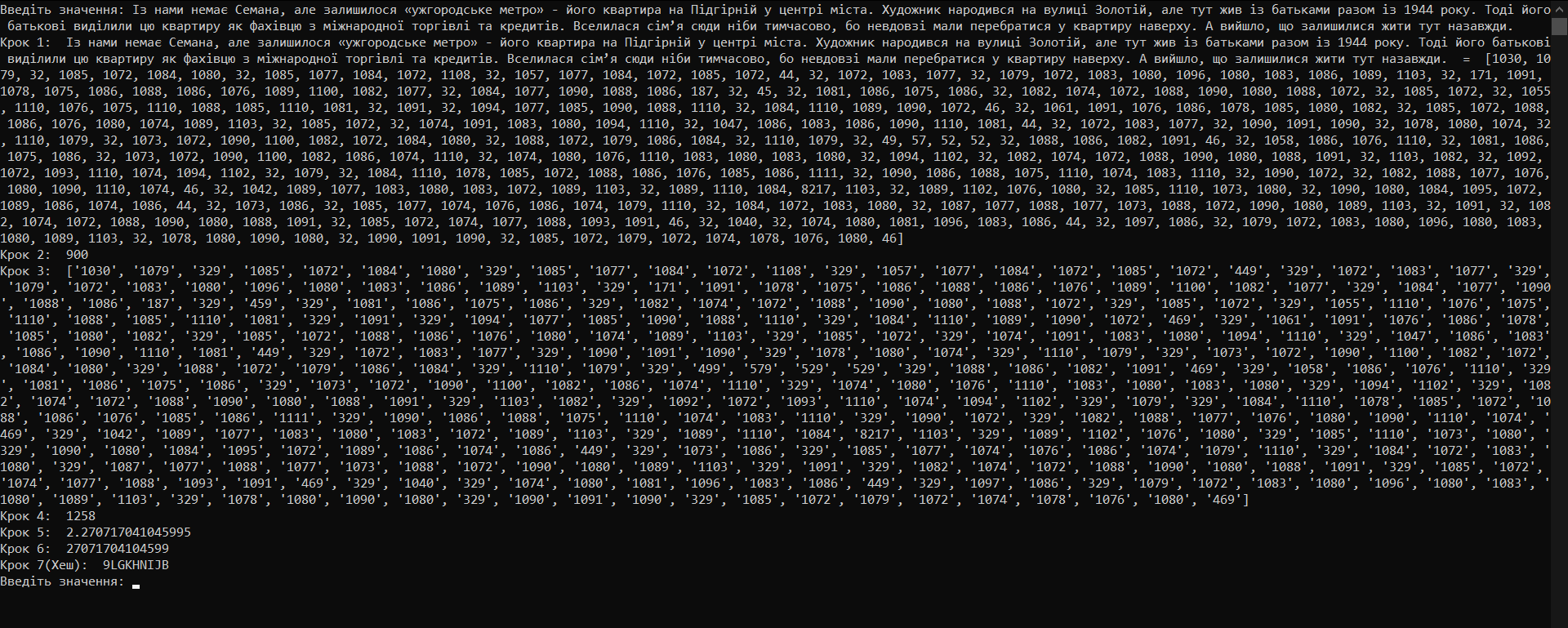
from math import e  
  
  
def dec\_to\_base(N, base):  
 if not hasattr(dec\_to\_base, 'table'):  
 dec\_to\_base.table = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'  
 x, y = divmod(N, base)  
 return dec\_to\_base(x, base) + dec\_to\_base.table[y] if x else dec\_to\_base.table[y]  
  
  
while True:  
 try:  
 while True:  
 name = input("Введіть значення: ")  
 general = []  
 general\_line = []  
 char\_list = []  
 tenderloin = ""  
 total = ""  
 total\_line = []  
 for i in name:  
 line = str(ord(i))  
 general\_line.append(line)  
 n = int(line)  
 general.append(n)  
 print("Крок 1: ", name, " = ", general)  
 arithmetic\_mean = str(sum(general) // len(name))  
 print("Крок 2: ", arithmetic\_mean)  
 for c in arithmetic\_mean:  
 char\_list.append(c)  
 num = -1  
 while num < len(arithmetic\_mean):  
 num += 1  
 for b in general\_line:  
 if len(b) < len(arithmetic\_mean):  
 m\_index = general\_line.index(b)  
 general\_line.remove(b)  
 b = b + char\_list[num]  
 general\_line.insert(m\_index, b)  
 print("Крок 3: ", general\_line)  
 general.clear()  
 char\_list.clear()  
 for ini in general\_line:  
 total += ini  
 for u in total.replace("0", ""):  
 total\_line.append(u)  
 gim = 0  
 for h in total\_line:  
 h\_index = total\_line.index(h)  
 d = int(h)  
 total\_line.remove(h)  
 total\_line.insert(h\_index, d)  
 sum\_all = sum(total\_line[:len(total\_line):4])  
 print("Крок 4: ", sum\_all)  
 for element in general\_line:  
 element\_int = int(element)  
 general.append(element\_int)  
 value = sum(general) // sum\_all  
 result = value % e  
 print("Крок 5: ", result)  
 result\_str = str(result)  
 removed = result\_str.replace(".", "")  
 lin\_num = []  
 for ele in removed[1:15]:  
 lin\_num.append(ele)  
 if lin\_num[0] == "0":  
 lin\_num.remove("0")  
 lin\_num.insert(0, str(value)[0])  
 for var in lin\_num:  
 tenderloin = tenderloin + var  
 print("Крок 6: ", tenderloin)  
 print("Крок 7(Хеш): ", dec\_to\_base(int(tenderloin), 36))  
 except KeyError:  
 print("Неправильний ввід")  
 except ValueError:  
 print("Неправилльний ввід")  
 except ZeroDivisionError:  
 print("Ти нічого не ввів")  
 except OverflowError:  
 print("Дуже ввелике значення")

По скільки першою вимогою для хеш-функції це введення даних будь-якого розміру то переведем тестування:

Введені дані:

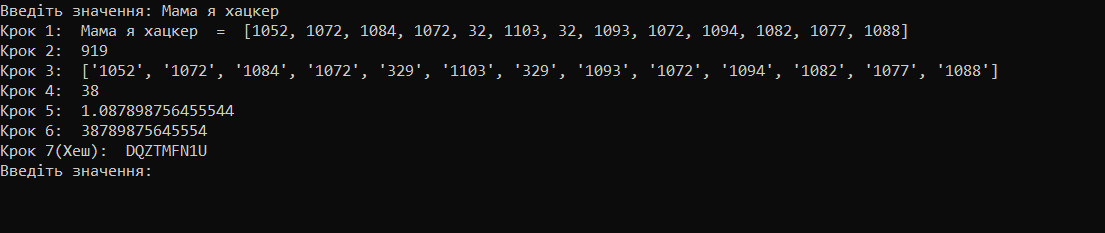
«Із нами немає Семана, але залишилося «ужгородське метро» – його квартира на Підгірній у центрі міста. Художник народився на вулиці Золотій, але тут жив із батьками разом із 1944 року. Тоді його батькові виділили цю квартиру як фахівцю з міжнародної торгівлі та кредитів. Вселилася сім’я сюди ніби тимчасово, бо невдовзі мали перебратися у квартиру наверху. А вийшло, що залишилися жити тут назавжди.»

Результат:



Введені дані: Мама я хацкер

Результат:

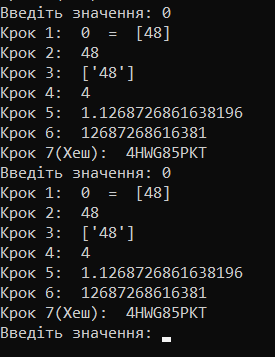


Введені дані:

0

0

Результат:

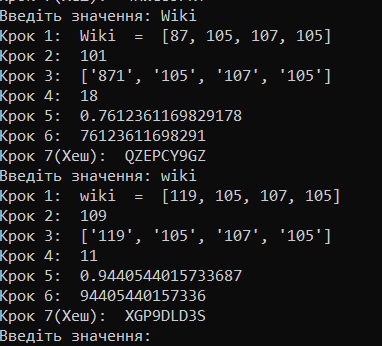


Введені дані:

Wiki

wiki

Результат:



Введені дані:

Nanan

anaNn

Результат:

