Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 7

з дисципліни «Теорія алгоритмів»

на тему:

**"** **Хеш-таблиці"**

Виконав:

студент гр. ІС-02

Плостак Ілля

Викладач:

Cт. Вик. Новікова П. А.

Київ – 2021

**1. Завдання**

Нехай заданий масив чисел A та число S. Потрібно дізнатись, чи присутні в масиві A два числа, сума яких дорівнює S.

В роботі необхідно реалізувати різні типи хеш-таблиць із використанням різних хеш-функцій для розв’язання наведеної вище задачі. При цьому потрібно порівняти ефективність різних підходів шляхом підрахунку кількості колізій для кожного типу хеш-функцій та хеш-таблиць.

Нижче наведений перелік типів хеш-таблиць та хеш-функцій:

1. Хеш-таблиця на основі ланцюгів (chained hash) із використанням хеш-функції за методом ділення.

2. Хеш-таблиця на основі ланцюгів (chained hash) із використанням хеш-функції за методом множення.

**2. Програмний код:**

[main.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab7/main.py):

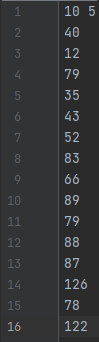
from algorithms import foundNumsForSum, makeHashTable, hashDiv, hashMult  
  
arrLen = 0  
sumsLen = 0  
arr = []  
sums = []  
  
for i, line in enumerate(open('input.txt')):  
 if i == 0:  
 tempArr = line.split(" ")  
 arrLen = int(tempArr[0])  
 sumsLen = int(tempArr[1])  
 elif i <= arrLen:  
 arr.append(int(line))  
 elif i <= arrLen + sumsLen:  
 sums.append(int(line))  
  
fOut = open("output.txt", "w")  
for i in range (len(sums)+1):  
 if i == 0:  
 T1, counter1 = makeHashTable(A=arr, hashFunc=hashDiv)  
 T2, counter2 = makeHashTable(A=arr, hashFunc=hashMult)  
 fOut.write(str(counter1) + " " + str(counter2) + "\n")  
 else:  
 temp = foundNumsForSum(A=arr, S=sums[i-1], hashFunc=hashMult)  
 fOut.write(str(temp[0]) + " " + str(temp[1]) + "\n")  
fOut.close()

[algorithms.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab7/algorithms.py):

import math  
  
def makeHashTable(A, hashFunc): # Создание хеш-таблицы  
 T = [None for i in range(len(A)\*3)] # Создаём пустую таблицу (все элементы = Null)  
 # длиной в три раза большем чем размер входного массива  
 counter = 0 # Создаём счётчик коллизий  
 for i in range(len(A)): # От 0 до длины входных данных  
 temp = hashFunc(A[i], len(T)) # Создаём временную переменную хранящую хеш текущего элемента  
 if T[temp] == None: # Если по хешу текущего элемента пусто -  
 T[temp] = [A[i]] # создаём пустой писок и кладём туда текущий элемент  
 else: # Иначе  
 counter += 1 # Произошла коллизия  
 T[temp].append(A[i]) # Добавляем текущий элемент в конец списка для утранения коллизий  
 return T, counter # Возвращаем созданную таблицу и количество коллизий  
  
def foundNumsForSum(A, S, hashFunc): # Поиск пары чисел по переданной сумме  
 T, counter = makeHashTable(A, hashFunc) # Создание хеш-таблицы на основе переданных данных  
 for i in range(len(A)): # От 0 до длины входных данных  
 temp = hashFunc(S - A[i], len(T)) # Создаём временную переменную хранящую хеш искомого элемента  
 if temp >= 0 and temp < len(T): # Если хеш-ключ не выходит за границы таблицы  
 if T[temp] != None: # Если по хешу искомого элемента не пусто  
 if SearchInList(T[temp], S - A[i]): # Если в списке по хешу искомого элемента есть искомый элемент  
 return [A[i], S - A[i]] # Возвращаем искомый элемент  
 return [0, 0] # Иначе возвращаем два нуля  
  
def hashDiv(k, m): # Хеш-алгоритм деления  
 return k % m  
  
def hashMult(k, m): # Хеш-алгоритм умножения  
 A = (math.sqrt(5)-1)/2  
 return math.ceil(m\*((k\*A) % 1))  
  
def SearchInList(arr, value):  
 for i in arr:  
 if i == value:  
 return True  
 return False

**3. Приклад вхідного файлу та приклад вихідного файлу:**

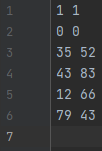
input.txt:



output.txt:

Перший рядок – кількість колізій обох методів

Інші рядки – пари шуканих чисел



**4. Висновок:**

В цій лабораторній роботі ми вивчили нову структуру даних: хеш-таблиці. За допомогою цієї структури даних ми вирішили задачу по пошуку двох чисел у масиві, що дають у сумі задане число. Цей алгоритм виглядає таким чином:

1. Додати до хеш-таблиці T всі елементи масиву A. Час роботи цієї операції — O(n).
2. Для кожного числа x з A визначити, чи присутнє в хеш-таблиці T число S - x. Якщо так, то ми знайшли два числа, які утворюють цю суму та знаходяться у вхідному масиві A. Час роботи цього етапу — O(n).

Таким чином, на відміну від тривіального алгоритму, який попарно перебирає всі можливі пари чисел (складність O(n2)), загальний час розв'язку задачі із використанням хеш-таблиць становить O(n).

Перевіримо правильність на прикладі:

Вхідні дані: 40, 12, 79, 35, 43, 52, 83, 66, 89, 79.

Сума: 126.

Перебравши усі пари чисел ми знайдемо потрібну пару чисел: 43 та 83.

Перевіряємо, та дійсно програма дала нам той самий результат: 

Програма працює коректно.

Також ми порівняли дві хеш-функції на кількість колізій. Перша хеш-функція на основі алгоритму ділення у різних випадках розміру хеш-таблиці дає різні результати:

1. Потрійний розмір вхідних даних – 1 колізія.
2. Подвійний розмір вхідних даних – 3 колізії.
3. Розмір вхідних даних – 4 колізії.

Друга хеш-функція на основі алгоритму множення у різних випадках розміру хеш-таблиці дає кращі результати:

1. Потрійний розмір вхідних даних – 1 колізія.
2. Подвійний розмір вхідних даних – 1 колізія.
3. Розмір вхідних даних – 2 колізії.

З чого робимо висновок – метод множення кращій ніж метод ділення за рахунок меншої кулькості колізій при рівних розмірах хеш-таблиць.

Ввід та вивід у програми здійснюється через файли.