**Дисциплина «Алгоритмы решения прикладных задач»**

**Рабочая тетрадь 4.**

**Хэш-таблица. Ассоциативный массив**

|  |  |
| --- | --- |
| **Теоретический материал** | |
| **Ассоциативный массив** — абстрактный тип данных, в котором хранятся пары «ключ — значение» с уникальными ключами. **Пара «ключ** — **значение»** со­стоит из двух фрагментов данных, отображаемых вместе: ключа и значения. **Ключ** — это фрагмент данных для извлечения значения. **Значение** — фрагмент данных, для извлечения которого используется ключ. В качестве ключа могут выступать как целые числа, так и строки или какие-либо другие данные. Элемент «**Значение**» - это данные любой природы и сложности (начиная от целых чисел и заканчивая объектами классов).  Существует много различных реализаций ассоциативного массива, но наиболее популярной реализацией являются хеш-таблицы. Хеш-таблица — линейная структура данных, в которой хранятся пары «ключ — значение» с уникальными ключами, а это значит, что вы не можете хранить дубликаты ключей в хеш-таблице.  Языки программирования, конечно, содержат в готовом виде хэш-таблицу. В Python это словари, в C++ это std::unordered\_set.  Для «ручной» реализации хэш-таблицы используется обычный массив. При этом в паре «ключ-значение» ключ преобразуется в индекс элемента массива. Для преобразования используют хэш-функцию, которая преобразует ключ в число из диапазона от 0 до размера массива для хранения хэш-таблицы. Хэш-функции бывают разные (хотя бы потому, что в качестве ключа могут выступать как числовые значения, так и, например, строки).  Рассмотрим пример. Для простоты, в данном примере элемент «значение» пары будет равняться ключу. Пустая хэш-таблица (на основе массива) содержит 7 элементов и выглядит так:    Будем заполнять хэш-таблицу элементами:  86, 90, 27, 29, 38, 39, 40.  Первое число, которое нужно сохранить, — 86. Чтобы сохранить 86 в хеш- таблице, необходима хеш-функция. Одна из простых хеш-функций состоит в том, чтобы взять каждое число и выполнить операцию остатка от деления на количество элементов массива. Например, чтобы получить хеш- значение для числа 86, вы вычисляете 86 % 7. Результат равен 2, а значит, помещаем 86 в ячейку с индексом 2 в массиве, который используете для сохранения данных хеш-таблицы.    Следующее число, которое нужно сохранить в хеш-таблице, — 90, поэтому вы вычисляете 90 % 7, что равно 6. Итак, помещаем 90 в индекс 6 в вашем массиве.    И наконец, нужно добавить 21, 29, 38, 39 и 40 в хеш-таблицу. Вот что про­изойдет, когда найдем остаток от деления на 7 для этих чисел:  27 % 7 = 0  29 % 7 = 1  38 % 7 = 3  39 % 7 = 4  40 % 7 = 5    До сих пор добавление данных в хеш-таблицу происходило по плану. Предположим, мы хотим еще добавить число 30. Поскольку 30 % 7 равно 2, мы должны добавить 30 в слот 2. Здесь, однако, возникает проблема, потому что в этом слоте уже есть число 86. Когда два числа попадают в один и тот же слот, возникает **коллизия**. Чтобы разрешить ее, мы помещаете 30 в следующий пустой слот (допустим, в нашем массиве еще есть пустые слоты). Такое решение работает, однако, если вам потребуется найти чис­ло 30, придется сделать следующее: использовать хеш-функцию для поиска местоположения числа в массиве, проверить ячейку с индексом 2, понять, что в ней записано не число 30, а за­тем просматривать последующие ячейки до тех пор, пока вы не найдете искомое число. Все это добавляет вычислительной сложности.  Существуют и другие способы решения коллизий, такие как хранение в каждой ячейке массива указателя на «голову» связанного списка и помещение каждой конфликтующей пары в список, соответствующий исходной конфликтной ячейке. То есть в любом случае каждое «значение» помещается в связанный список, указатель на голову которого хранится в ячейке массива (хэш-таблицы). Если в связанном списке, предназначенном для добавляемого элемента, уже есть элемент, то новое значение добавляется в хвост хранимого связанного списка. Если элементов в списке нет, то указатель на голову является нулевым (nullptr).  Если есть необходимость сохранить сами ключи (отличающиеся от значений), то организуют еще один массив для хранения ключей. Индекс элемента для хранения ключа также определяется на основе хэш-функции. | |
| **Задание 1** | |
| ***Задача:*** | |
|  | |  |  | | --- | --- | | Повторить пример, похожий на разобранный в «Теоретическом материале»: на вход программе подается цепочка целых чисел: 86, 90, 27, 29, 38, 30, 40. Сформировать из этого набора данных хэш-таблицу (ключ и значение совпадают). Размер массива равен 7. В качестве функции хэширования рассмотреть остаток от деления на 7. Выбрать любой способ избежания коллизии. |  | |  |  | |
| ***Решение:*** | |
|  | dict\_s = dict()  lst = [86, 90, 27, 29, 38, 30, 40]  for i in lst:      if i % 7 not in dict\_s:          dict\_s[i % 7] = [i]      else:          dict\_s[i % 7].append(i)  sorted\_dict = {k: v for k, v in sorted(dict\_s.items(), key=lambda x: x[0])}  print(sorted\_dict) |
| ***Ответ:*** | |
|  |  |
| **Задание 2** | |
| ***Задача:*** | |
|  | Написать программу, на вход которой подается строка, а на выходе перечисляются содержащиеся в строке символы с указанием количества вхождений символа. Реализовать алгоритм с использованием хэш-таблицы. Можно использовать самостоятельно написанную хэш-таблицу, но можно использовать и готовые решения (например, словари в Python).  Рекомендации по созданию алгоритма:   1. Реализовать алгоритм в виде функции, на вход которой подается строка) 2. Создаем пустую хэш-таблицу 3. В цикле for перебираем символы строки 4. Если символа еще нет в хэш-таблице, добавляем в первый массив (в первую по счету пустую ячейку) новый ключ (символ), а во второй массив значени­е 1 (в ячейку с тем же индексом), поскольку символ впервые появляется в хэш-таблице. 5. Если символ есть в хэш-таблице, увеличиваем значение на 1 6. После прогона через цикл всей строки выводим на печать пары «ключ-значение» |
| ***Решение:*** | |
|  | n = input()  dict\_of\_n = dict()  for i in n:      if i not in dict\_of\_n:          dict\_of\_n[i] = n.count(i)  print(dict\_of\_n) |
| ***Ответ:*** | |
|  |  |
| **Задание 3\*** | |
| ***Задача:*** | |
|  | Задача «сумма двух». Вернуть индексы двух чисел в неотсортирован­ном массиве, которые в сумме дают заданное значение. Решить задачу на основе хэш-таблицы. Например, для массива [-1, 2, 3, 4, 7] и заданного значения 5 выводятся индексы 1 и 2 (поскольку 2+3=5). |
| ***Решение:*** | |
|  | n = list(map(int, input().split()))  dict\_n = dict()  for i in range(len(n)):      for j in range(i + 1, len(n)):          if n[i] + n[j] not in dict\_n:              dict\_n[n[i] + n[j]] = [[i, j]]          else:              dict\_n[n[i] + n[j]].append([i, j])  dict\_n = {k: v for k, v in sorted(dict\_n.items(), key=lambda x: x[0])}  print(\*dict\_n[int(input())]) |
| ***Ответ:*** | |
|  |  |
| **Задание 4** | |
| ***Задача:*** | |
|  | Напишите с использованием хэш-таблицы функцию, которая возвращает пересечение двух массивов - третий массив со всеми значениями из обоих исходных. Например, пересечение массивов [1, 2, 3, 4, 5] и [0, 2, 4, 6, 8] - это [2, 4]. Сложность вашей функции должна быть O(N). Если ваш язык программирования предусматривает встроенный способ решения этой задачи, не используйте его. Создайте этот алгоритм самостоятельно. |
| ***Решение:*** | |
|  | lst\_1 = list(map(int, input().split()))  lst\_2 = list(map(int, input().split()))  lst\_1\_and\_2 = []  for i in range(len(lst\_1)):      if lst\_1[i] in lst\_2:          lst\_1\_and\_2.append(lst\_1[i])  print(lst\_1\_and\_2) |
| ***Ответ:*** | |
|  |  |
| **Задание 5\*** | |
| ***Задача:*** | |
|  | Напишите с использованием хэш-таблицы функцию, которая принимает массив строк и возвращает первое повторяющееся значение. Например, в случае с массивом ["а", "b", "с", "d", "с", "е", "f"] эта функция должна возвратить "с", так как оно встречается в массиве более одного раза. Убедитесь, что сложность этой функции - O(N). |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |
| **Задание 6\*** | |
| ***Задача:*** | |
|  | Напишите с использованием хэш-таблицы функцию, которая возвращает первый неповторяющийся символ в строке. Например, в строке "minimum" есть два неповторяющихся символа - "n" и "u ", поэтому ваша функция должна возвратить "n", так как он встречается первым. Временная сложность вашей функции должна быть равна O(N). |
| ***Решение:*** | |
|  |  |
| ***Ответ:*** | |
|  |  |