**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра АПУ**

алгоритмы и Структуры данных

Методические указания

к выполнению курсового проекта

Санкт-Петербург

2022

**ВВЕДЕНИЕ**

Цель курсового проектирования – изучение структур данных и алгоритмов их обработки, а также получение практических навыков их использования при разработке программ.

**1 ОСНОВНЫЕ ТРЕБОВАНИЯ К РАБОТЕ**

Подготовленный курсовой проект должен содержать:

- корректно работающее приложение, написанное на языке C++, и удовлетворяющее требованиям варианта задания;

-  пояснительную записку в печатном и электронном виде.

Пояснительная записка к курсовому проекту должна содержать:

1) титульный лист;

2) содержание;

3) задание на курсовой проект (с указанием выбранного варианта);

4) введение (краткая характеристика решаемой задачи, обоснование необходимости решения данной и подобных задач);

5) описание программы (краткое описание структуры программы, руководство по использованию программы, листинг программы с комментариями);

6) тестирование программы (исходные данные для тестовых прогонов программы, результаты тестирования);

7) заключение;

8) список использованной литературы.

Все приведенные разделы являются обязательными. Листинг программы допускается помещать в приложении к пояснительной записке.

**2 ТЕОРЕТИЧЕСКАЯ СПРАВКА**

В рамках курсового проектирования студент научится организовывать данные в виде хеш-таблиц, производить преобразование этих данных и осуществлять поиск различного рода информации. Прежде чем приступить к описанию задания на курсовой проект, рассмотрим небольшую теоретическую справку.

*Таблица* – это список, состоящий из конечного множества элементов, причем каждый элемент характеризуется рядом признаков (свойств). Один из признаков, называемый *ключом*, позволяет отличить один элемент от другого (идентифицировать элемент). Ключ может однозначно определять элемент таблицы (ключи всех элементов различны) или неоднозначно (в таблице есть элементы с равными ключами).

Все действия над элементами выполняются в соответствии с их ключами: по ключу элементы выбираются из таблицы и добавляются в нее.

Рассмотрим пример (таблица 1): пусть имеется товарная накладная с информацией о датах и городах (когда и куда отправляются товары). Определенному товару соответствует его числовой код в диапазоне от 000000000 до 999999999.

**Таблица 1. Пример товарной накладной**

|  |  |  |
| --- | --- | --- |
| **Код** | **Дата** | **Город доставки** |
| 00 | 2016-03-10 | Тула |
| …… | …… | …… |
| 000142426 |  |  |
| 000142427 | 2018-08-25 | Орел |
| …… | …… | …… |
| 603603003 | 2017-05-14 | Казань |
| 603603004 |  |  |
| …… | …… | …… |
| 999999999 | 2026-09-01 | Хабаровск |

Здесь ключами являются коды товаров. Несмотря на отсутствие некоторых единиц продукции, заполняется весь диапазон. Такая организация данных не оптимальна, поскольку памяти для хранения списка выделяется больше чем нужно. Решением проблемы будет хеширование ключей списка.

Хеш-таблицей называется структура данных, предназначенная для реализации ассоциативного массива, такого в котором адресация реализуется посредством хеш-функции.

Хеш-таблица применяется в случаях, когда мощность универсума U очень велика по отношению к мощности множеств, с которыми нужно работать.

Хеш-функция, или функция отображения, вычисляет для каждого элемента таблицы по его ключу номер (позицию) элемента в массиве, таким образом, преобразует значения ключей к интервалу [0, m – 1], где m — размер хеш-таблицы, m ≪ |U|.

Очевидно, что при этом каждому индексу хеш-таблицы будет соответствовать много различных значений ключей. Поэтому, во-первых, в хеш-таблице приходится хранить не биты, а сами значения ключей, а во-вторых, имеется возможность размещать в ней более одного ключа для каждого значения функции отображения (разрешать коллизии).

Количество возможных коллизий можно уменьшить, если выполнить два условия:

1) выбрать размер хеш-таблицы с запасом. Если размер таблицы превышает мощность хранимого множества более чем вдвое, вероятность коллизии становится меньше 0,5. Если мощность множества заранее неизвестна, то выбирают некоторый начальный размер, а когда его оказывается недостаточно, таблицу перестраивают с увеличением размера (обычно вдвое);

2) подобрать функцию отображения (хеш-функцию) такую, чтобы все ячейки таблицы были востребованы по возможности с равной вероятностью, независимо от того какое распределение имеют хранящиеся в таблице ключи.

Известно несколько методов получения хеш-функции, при этом обычно предполагают, что область определения хеш-функции – целые неотрицательные числа, если ключи не являются такими. их всегда можно преобразовать к целому типу:

#### таблица прямого доступа,

#### метод остатков от деления,

#### метод функции середины квадрата,

#### метод свертки и др.

По способу разрешения коллизий различают хеш-таблицы двух типов:

1) Открытое хеширование (метод цепочек).

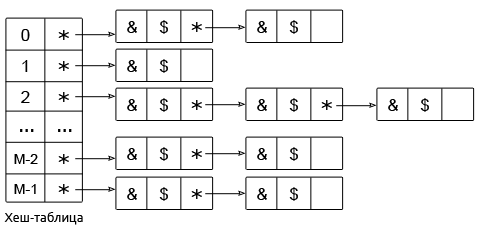
2) Закрытое хеширование (с открытой адресацией).

Подробнее о хеш-таблицах см. [1, с. 529–556], [2, с. 567–601], [3, с. 115–128], [4, с. 316–338], **5, 6**.

**Открытое хеширование**

Принцип организации хеш-таблицы методом открытого хеширования заключается в реализации логически связанных цепочек, начинающихся в ячейках хеш-таблицы. Под цепочками подразумеваются связанные списки, указатели на которые хранятся в ячейках хеш-таблицы. Каждый элемент связанного списка – блок данных, и если с некоторым указателем, хранящимся по адресу i, связаны n таких блоков (n>1), то это свидетельствует о том, что n ключей получили один и тот же хеш-код i, т. е. имеет место коллизия. Но метод открытого хеширования устраняет конфликт, поскольку данные хранятся не в самой таблице, а в связных списках, которые увеличиваются при возникновении конфликта.

На рисунке 1 изображены связанные списки со ссылающейся на них хеш-таблицей (ее размер = M). Первый столбец таблицы содержит хешированные значения ключей, второй – ссылки на списки. Количество последних ограничено лишь числом элементов исходного массива (он не показан, но предполагается). Состоят списки из трех (последний элемент подсписка – из двух) полей: & — адрес элемента списка, $ — данные, \* — указатель (ссылка).



**Рисунок 1 – Пример открытого хеширования**

Если в исходном массиве было всего N элементов (столько же будут содержать в совокупности и все списки), то средняя длина списков будет равна α=N/M, где M – число элементов хеш-таблицы, α – коэффициент заполнения хеш-таблицы. Предположив, например, что в списке на рисунке выше M=5, получим среднее число списков α=2.

Чтобы увеличить скорость работы операций поиска, вставки и удаления нужно, зная N, подобрать M примерно равное ему, т. к. тогда α будет равняться 1-ому или ≈1-ому, следовательно, можно рассчитывать на оптимальное время, в лучшем случае равное *O*(1). В худшем случае все N элементов окажутся в одном списке, и тогда, например, операция нахождения элемента (в худшем случае) потребует *O*(N) времени.

**Закрытое хеширование**

Первый метод назывался открытым, потому что он позволял хранить сколь угодно много элементов, а при закрытом хешировании их количество ограниченно размером хеш-таблицы. В отличие от открытого хеширования закрытое не требует каких-либо дополнительных структур данных. В ячейках таблицы хранятся не указатели, а элементы исходного массива, доступ к каждому из которых осуществляется по хеш-коду ключа, при этом одна ячейка может содержать только один элемент.

Сам процесс заполнения хеш-таблицы с использованием алгоритма закрытого хеширования осуществляется следующим образом:

* имеется изначально пустая хеш-таблица T размера M, массив A размера N (M≥N) и хеш-функция h(), пригодная для обработки ключей массива A;
* элемент xi, ключ которого keyi, помещается в одну из ячеек хеш-таблицы, руководствуясь следующим правилом:

a) если h(keyi) – номер свободной ячейки таблицы T, то в последнюю записывается xi;

b) если h(keyi) – номер уже занятой ячейки таблицы T, то на занятость проверяется другая ячейка, если она свободна то xi заносится в нее, иначе вновь проверяется другая ячейка, и так до тех пор, пока не найдется свободная или окажется, что все M ячеек таблицы заполнены.

Последовательность, в которой просматриваются ячейки хеш-таблицы, называется последовательностью проб.

Последовательность проб задается специальной функцией. В зависимости от типа этой функции выделяют:

* линейное опробование;
* квадратичное опробование;
* двойное хеширование.

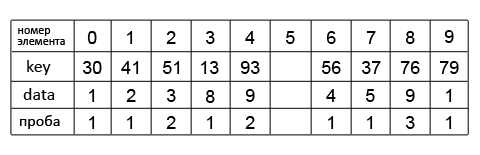
Рассмотрим метод закрытого хеширования на примере построения хеш-таблицы. Положим, имеется целочисленный массив A, состоящий из 9 элементов:

{A[key]=data, здесь key – ключ, data – некоторые данные}

A[13]=8, A[56]=4, A[79]=1, A[37]=5, A[41]=2, A[76]=9, A[51]=3, A[93]=9, A[30]=1

Также есть хеш-таблица размера M=10.

Хеш-функция h(key) основана на методе деления и вычисляется как остаток от деления key на размер хеш-таблицы: key % M (% – операция «остаток от деления»). Заполним хеш-таблицу элементами массив A:



**Рисунок 2 – Хеш-таблица с элементами массива А**

Для расстановки элементов использовалась выбранная формула. Подставив ключ, например первого элемента в нее получим: h(13) = 13 % 10 = 3, поэтому его номер в хеш-таблице 3. Последовательное добавление элементов приведет к возникновению коллизии при обработке элемента A[76]. Хеш-код его ключа 6, но в хеш-таблице ячейка с таким номером уже занята.

Используя формулу линейного пробирования (тип последовательности проб) hi(key)=(h(key) + i) % M (i – число проверок, после первой проверки i=0), продолжим поиск свободной ячейки. Применим функцию при i=1: h1(76)=7; убедившись, что ячейка 7 занята, продолжаем поиск, увеличив i на 1: h2(76)=8. Ячейка 8 свободна, помещаем в нее элемент. Этот же метод используем и для всех остальных элементов.

При любом методе разрешения коллизий необходимо ограничить длину поиска элемента. Если для поиска элемента необходимо более 3 – 4 сравнений, то эффективность пользования такой хеш-таблицы пропадает и ее следует реструктуризировать (т.е. найти другую хеш-функцию), чтобы минимизировать количество сравнений для поиска элемента.

Для успешной работы алгоритмов поиска, последовательность проб должна быть такой, чтобы все ячейки хеш-таблицы оказались просмотренными ровно по одному разу.

Удаление элементов в такой схеме несколько затруднено. Обычно поступают так: заводят логический флаг для каждой ячейки, помечающий, удален ли элемент в ней или нет. Тогда удаление элемента состоит в установке этого флага для соответствующей ячейки хеш-таблицы, но при этом необходимо модифицировать процедуру поиска существующего элемента так, чтобы она считала удаленные ячейки занятыми, а процедуру добавления – чтобы она их считала свободными и сбрасывала значение флага при добавлении.

**3. ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

В соответствии с выбранным вариантом (таблица 2):

1. Организуйте данные из предметной области в виде хеш-таблицы (*первичный ключ хеш-таблицы выделен курсивом*). Метод хеширования также указан в таблице с вариантами заданий. В качестве алгоритма хеширования использовать *метод деления*. Обратите внимание, что для вычисления хеш-функции, значение ключевого поля должно быть целым (для перевода строковых данных в целые можно воспользоваться функцией atoi). Примеры реализации хеш-таблиц с открытым и закрытым хешированием см. [5].

2. Реализуйте следующие функции:

* Меню для выбора функции (используя конструкцию switch — case).
* ~~Добавление записи (в случае закрытого хеширования необходимо отслеживать заполненность хеш-таблицы) (см. [5], [7]).~~
* Проверку корректности вводимой информации. +-
* ~~Удаление записи (см. [5], [7]).~~
* ~~Просмотр всех имеющихся записей.~~
* ~~Удаление всех записей.~~
* ~~Поиск записи по ключевому полю.~~ Пример реализации см. [5], [7].
* ~~Поиск данных по фрагменту одного из не ключевых полей (см. вариант задания). Для поиска подстроки в строке необходимо использовать~~ *~~прямой поиск~~*~~. Подробнее о прямом поиске см. [8].~~

**Таблица 2. Варианты заданий**

| **№** | **Предметная область, характеристики** | **Метод хеширования** | **Поиск по фрагменту строки** |
| --- | --- | --- | --- |
| 1 | **Данные о покупателе:**  *Id – строка формата «NNNNNN», где N – цифры.*  ФИО - строка  Адрес - строка | Закрытое хеширование с линейным опробованием | Поиск покупателей с заданной фамилией |
| 2 | **Данные об обучающемся:**  *№ договора – строка формата «NNNNN», где N – цифры.*  ФИО – строка  Дата рождения - строка | Открытое хеширование | Поиск обучающихся с заданной фамилией |
| 3 | **Данные об авто:**  *Гос. регистрационный № автомобиля – строка формата «ANNNAANNN», где N-цифра, A – буква из множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х.*  Марка - строка  ФИО владельца – строка | Закрытое хеширование с линейным опробованием. В качестве ключевого значения для расчета функции хеширования взять число, состоящее из цифр номера авто. | Поиск авто заданной марки |
| 4 | **Данные о посетителе мероприятия:**  *Id - строка формата «NNNNNNNN», где N – цифры.*  ФИО - строка  Телефон – строка формата «NNNNNNNNNNN», где N – цифра. | Открытое хеширование | Поиск посетителей с заданной фамилией |
| 5 | **Данные о пациенте:**  *№ паспорта -* *строка формата «NN-NNNNNN», где N-цифра*  ФИО – строка  Филиал – строка | Закрытое хеширование с линейным опробованием.  В качестве ключевого значения для расчета функции хеширования взять число, состоящее из цифр серии и номера паспорта. | Поиск пациентов с заданной фамилией |
| 6 | **Данные о покупателе:**  *Id – строка формата «NNNNNN», где N – цифры.*  ФИО - строка  Адрес - строка | Открытое хеширование | Поиск покупателей, живущих на заданной улице |
| 7 | **Данные об обучающемся:**  *№ договора – строка формата «NNNNN», где N – цифры.*  ФИО – строка  Дата рождения - строка | Закрытое хеширование с линейным опробованием | Поиск обучающихся с заданным годом рождения |
| 8 | **Данные об авто:**  *Гос. регистрационный № автомобиля – строка формата «ANNNAANNN», где N-цифра, A – буква из множества: А, В, Е, К, М, Н, О, Р, С, Т, У, Х.*  Марка - строка  ФИО владельца – строка | Открытое хеширование. В качестве ключевого значения для расчета функции хеширования взять число, состоящее из цифр номера авто. | Поиск авто, принадлежащих владельцам с заданной фамилией |
| 9 | **Данные о посетителе мероприятия:**  *Id - строка формата «NNNNNNNN», где N – цифры.*  ФИО - строка  Телефон – строка формата «NNNNNNNNNNN», где N – цифра. | Закрытое хеширование с линейным опробованием | Поиск посетителей с номерами телефона, содержащими определенную комбинацию цифр |
| 10 | **Данные о пациенте:**  *№ паспорта -* *строка формата «NN-NNNNNN», где N-цифра*  ФИО – строка  Филиал – строка | Открытое хеширование.  В качестве ключевого значения для расчета функции хеширования взять число, состоящее из цифр серии и номера паспорта. | Поиск пациентов заданного филиала |

***\*Примечание*** *– длина строк (кроме строк с указанным форматом) определяется студентом самостоятельно.*

**4. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА**

Курсовой проект разрешается выполнять в командах по 2 человека. Вариант задания курсового проекта назначается преподавателем.

При выполнении курсового проекта рекомендуется пользоваться материалами лекций, источниками информации из списков, приведенных в методических указаниях, а также другими источниками информации.

Особые требования к интерфейсу программы не предъявляются. Состав и форма отображаемой информации, а также способы управления программой и ввода данных, должны быть достаточными для демонстрации всех функций и структур данных, которые определены в задании.

При разработке структуры данных студент должен самостоятельно определять размерность некоторых элементов данных (например, длину строк), способ реализации этой структуры данных в памяти вычислительной машины, а также выбирать алгоритмы выполнения некоторых операций. Принятые решения должны быть отражены в пояснительной записке.

Для демонстрации программы необходимо подготовить тестовый набор данных. Объем этих данных должен быть достаточен для демонстрации основных свойств разработанных структур данных и выполнении всех заданных операций. В частности, в хеш-таблицу должны быть внесены несколько элементов, образующих коллизию.

**5. СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ**

1. Седжвик Р. Алгоритмы на С++ / пер. с англ. — М.: И. Д. Вильямс, 2011. — 1156 с.: ил.
2. Седжвик Р. Фундаментальные алгоритмы С++. — К.: Диасофт, 2001. — 484 с.
3. Ахо Дж., Хопкрофт А., Ульман Дж. Построение и анализ вычисли-тельных алгоритмов. — М.: Мир, 1979.
4. Кнут Д. Искусство программирования для ЭВМ. Т. 3: Сортировка и поиск. — М.: Мир, 2013.
5. <https://intuit.ru/studies/courses/648/504/lecture/11467?page=1>
6. <https://kvodo.ru/hash-table.html>
7. <https://github.com/ilyamoskalev/HashTable/blob/master/README.md>
8. <https://intuit.ru/studies/courses/648/504/lecture/11468>