**Aufgabenblatt 8 vom 24. Januar 2021, Abgabe am 07. Februar 2021, 22:00 Uhr**

**Задание 8.3:** База контактов27 Баллов

Хеш-функции, хеш-таблицы

**Важно: При обработке этой задачи необходимо соблюдать модификаторы видимости, указанные в диаграмме UML, иначе автоматическое исправление не удастся!**

В этом задании мы займемся хешированием. Хеш-функции сопоставляют большие входные количества с небольшими целевыми количествами. Это позволяет избежать долгих итераций по входному набору, а поиск искомого объекта занимает меньше времени. Мы предоставляем вам класс Contact в материале для этой задачи. Этот класс представляет собой контейнер для хранения связанных контактных данных. Для выполнения этой задачи мы предположим, что адрес электронной почты контакта уникален (т.е. нет двух контактов с одним и тем же адресом электронной почты). По этой причине мы можем использовать адрес электронной почты в качестве ключа для поиска контакта. В ходе выполнения задачи мы разрабатываем хеш-таблицы, по которым затем можем классифицировать(упорядочить) такие контакты. Мы разрабатываем как открытое, так и закрытое хеширование.

1. Создайте новый проект 08-ContactDatabase и добавьте классы Contact и AuDHashTable. Мы используем абстрактный класс AuDHashTable в качестве отправной точки для наших двух реализаций.
2. Сначала вы должны реализовать метод hash (String s) в классе AuDHashTable, который должен вычислять значение int для каждой переданной строки. Действуйте следующим образом:

Здесь s [i] - значение символа (char), которое находится в позиции i символьной строки, k - емкость хеш-таблицы, а n - длина символьной строки.

|  |
| --- |
| ***AuDHashTable*** |
| # capacity: int |
| + AuDHashTable(capacity: int)  *+ insert(c: Contact): void*  *+ remove(c: Contact): void*  *+ getContact(email: String): Contact*  # hash(s: String): int |

|  |  |
| --- | --- |
|  | **AuDOpenHashTable** |
| – table: LinkedList<Contact>[ ] |
|  |
| + AuDOpenHashTable(capacity: int) |

|  |
| --- |
| **AuDClosedHashTable** |
| * table: Contact[ ] * deleted: boolean[ ] * counter: int |
| + AuDClosedHashTable(capacity: int)  + isFull(): boolean  # hash(s: String, i: int): int  – getIndexOf(email: String): int |

Рисунок 1: UML - Диаграмма с реализуемыми классами.

1.**Открытое хеширование:**

Теперь создайте класс AuDOpenHashTable. Этот класс наследуется от AuDHashTable и реализует открытое хеширование. Здесь нет конфликтов индексов(индексных столкновений), но несколько контактов могут оказаться в одной записи в таблице. Таким образом, вам нужно выполнить поиск по записи, прежде чем вы сможете найти нужный контакт, но в то же время вы теоретически неограниченны с точки зрения количества сохраненных контактов.

1. Добавьте все методы и атрибуты из диаграммы UML на рисунке 1 в свой класс AuDOpenHashTable. Также обратите внимание на абстрактные методы надкласса.
2. Напишите конструктор. Здесь тоже следует установить мощность в соответствии со своим параметром. Кроме того, он должен инициализировать таблицу массивов в соответствии с этой емкостью. Кроме того, каждая запись в массиве должна быть инициализирована подходящим списком.
3. Метод insert(Contact c) должен добавлять(вставлять) контакт в вашу таблицу. Для этого рассчитайте позицию вставки с помощью метода int hash(String). Запомните, какой атрибут контакта подходит в качестве параметра поиска. Затем добавьте контакт в конец списка в рассчитанную позицию вставки в таблицe.

**Подсказки:**

* LinkedList уже доступен в Java, возможно, вам потребуется использовать(импортировать) java.util.LinkedList.
* Вы можете создавать массивы из LinkedLists следующим образом:

LinkedList <Contact> [] lists = new LinkedList [size];

При этом игнорируйте предупреждение в IntelliJ.

d)Метод remove должен удалить перенесенный контакт из таблицы. Итак, вычислите позицию контакта и удалите его из найденного там списка. Если контакт даже не существует в таблице, вызовите исключение NoSuchElementException.

e)Метод getContact (String email) должен вычислять позицию контакта с соответствующим адресом электронной почты. Затем необходимо найти контакт в списке и вернуть. Если контакт не существует в таблице, также вызовите исключение NoSuchElementException.

**Подсказка:** Вероятно, Вам нужно будет использовать(импортировать**)** java.util.NoSuchElementException.

**1. Закрытое хеширование:**

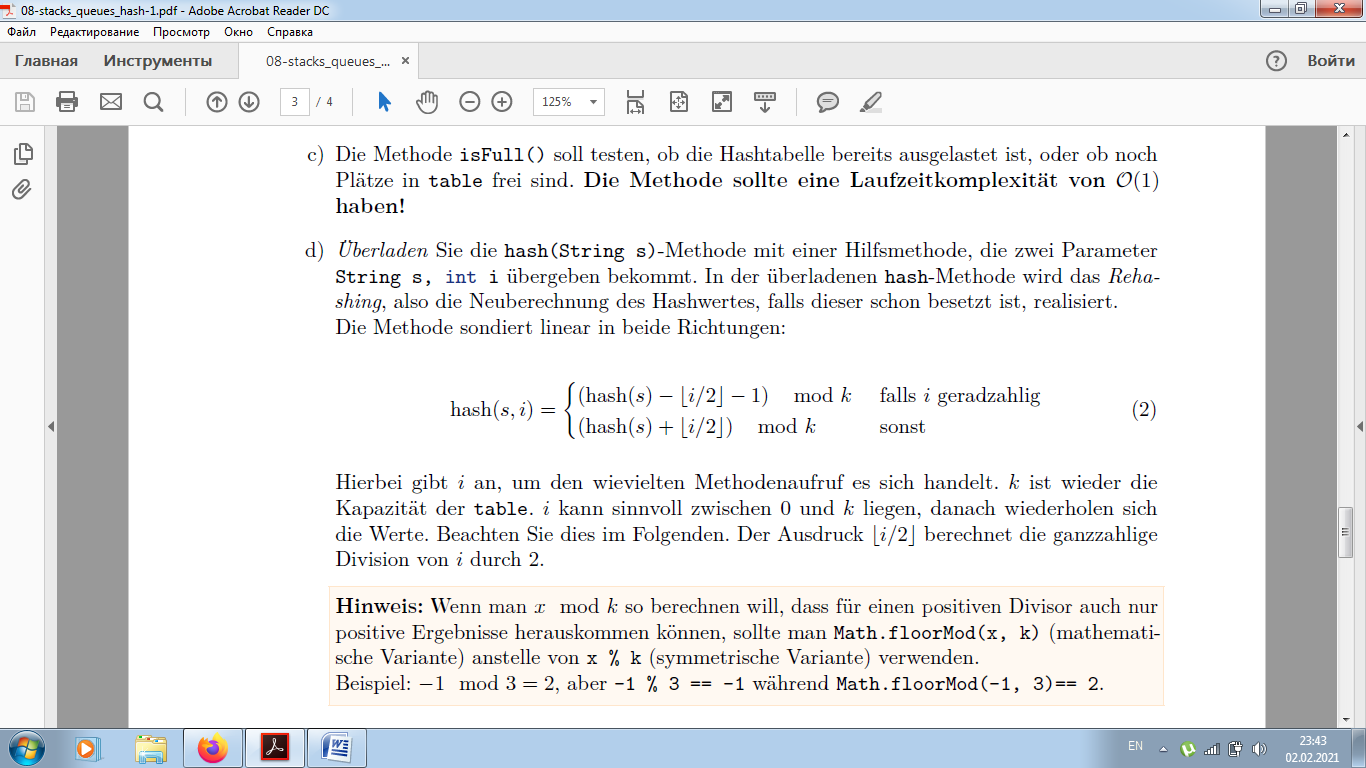
Второй вариант хеш-таблицы, который мы будем выполнять, - это закрытое хеширование. Для этого мы пишем класс AuDClosedHashTable. При закрытом хешировании в каждой записи таблицы может быть только один контакт. Это может привести к конфликтам, если хеш-функция вычисляет один и тот же индекс записи для двух контактов. В этом случае вы должны вызывать (проверять) дополнительные (разные) хеш-функции, пока не найдете пустую запись.

Это также влияет на удаление: теперь вы должны отметить, была ли установлена ​​запись, поскольку в этом случае вам нужно попробовать другие хеш-функции, даже если позиция пуста. Возможно, запись все еще существовала, когда Вы ее вставили, поэтому была использована другая хеш-функция. Кроме того, при закрытом хешировании вы можете сохранить столько значений, сколько памяти зарезервировано в начале.

* 1. Сначала перенесите атрибуты и методы, которые Вы видите на UML-диаграмме на рисунке 1, в свой класс AuDClosedHashTable.
  2. Реализуйте конструктор. Он устанавливает емкость и инициализирует таблицу(table) и соответственно удаление(deleted). Счетчик(counter) позже подсчитает, сколько элементов в настоящее время хранится в хэш-таблице, и изначально должен быть равен 0. Удаление (deleted[i]) должно быть установлено значение true в следующих случаях, если таблица записей (table [i]) удаляется.
  3. Метод isFull () предназначен для проверки того, заполнена ли хеш-таблица или есть ли еще свободные места в таблице.
  4. Перегрузите метод hash (String s) вспомогательным методом, который принимает два параметра: String s, int i. В перегруженном методе хеширования реализовано перехеширование, т.е. пересчет значения хеш-функции, если оно уже занято.

Метод должен иметь сложность выполнения *O*(1) !

Метод исследует линейно в обоих направлениях:



Здесь i указывает, сколько раз вызывается метод. k - снова емкость таблицы. i может значимо лежать между 0 и k, после чего значения повторяются. Обратите внимание на это ниже. Выражение [*i/*2] вычисляет целочисленное деление i на 2.

**Примечание:** если вы хотите вычислить x mod k таким образом, чтобы для положительного делителя могли быть получены только положительные результаты, вы должны использовать Math.floorMod (x, k) (математический вариант) вместо x % k (симметричный вариант).

Пример: −1 mod 3 = 2, но -1 % 3 = = -1, а Math.floorMod (-1, 3) = = 2

* 1. Метод insert(Contact c) заключается в вставке контакта в таблицу. Сначала проверьте, есть ли место для другой записи. Если нет, должно быть выброшено исключение UnsupportedOperationException. Если это так, позиция вставки должна быть ( ~› hash(s, 0)). Если она уже занята, рассчитайте новую позицию, увеличив i. Повторяйте это, пока не найдете пустую запись. Затем напишите контакт в этом месте.
  2. Создайте вспомогательный метод getIndexOf(String email), который упрощает поиск значений. Для данного адреса электронной почты положение контакта следует определять с помощью метода hash(String s). Теперь вам нужно проверить, действительно ли контакт, который вы ищете, находится в этой точке. Если там есть кто-то еще или контакт уже был удален из этой позиции, необходимо выполнить другой hash()-вызов с увеличенным i и проверить новую позицию для контакта. Продолжайте делать это, пока не найдете нужный контакт. Затем верните его позицию в массиве таблицы.

Как часто вам нужно выполнять дальнейшие hash()-вызовы, пока вы не убедитесь, что нет контакта с соответствующим адресом электронной почты в таблице? Включите в свой код соответствующий критерий завершения.

Если на самом деле ни один контакт не имеет адреса электронной почты, который вы ищете (подумайте, где в коде вы можете это узнать), должно быть сгенерировано исключение NoSuchElementException.

* 1. Используйте только что написанный вспомогательный метод, чтобы в remove (Contact c) найти переданный контакт в массиве и удалить его (установить в ноль). Какие еще атрибуты AuDClosedHashTable Вам нужно изменить?
  2. Вы также можете использовать вспомогательный метод, чтобы в методе getContact (String email) найти и вернуть контакт с соответствующим адресом электронной почты.

Мы закончили с нашей реализацией. Вы видели, как мы можем упростить и ускорить поиск отдельных контактов с помощью умного хранилища. Наконец, протестируйте ваши написанные классы и методы для обеих хэш-таблиц **в подробностях** в основном методе (main) во вновь созданном классе ContactDatabase.

Оставьте и эти тесты, мы их тоже оценим!

1. Сдайте AuDHashTable.java, AuDOpenHashTable.java, AuDClosedHashTable.java и

ContactDatabase.java на EST.

Sollte Ihr Programm nicht übersetz- bzw. ausführbar sein, wird die Lösung mit 0 Punkten bewertet. Stellen Sie also sicher, dass IntelliJ IDEA keine Fehler in Ihrem Programm anzeigt, Ihr Programm übersetz- und ausführbar ist sowie die in der Aufgabenstellung vorgegebenen Namen und Schnittstellen *exakt* eingehalten werden. Geben Sie am Schluss die Dateien AuDHashTable.java, AuDOpenHashTable.java, AuDClosedHashTable.java und ContactDatabase.java über die EST-Webseite ab. Wenn Sie die Aufgabe zusammen mit einem Übungspartner bearbeitet haben, geben Sie im EST unbedingt dessen Gruppenabgabe-Code an! Kontrollieren Sie, ob Ihre Namen am Anfang aller Dateien angegeben sind – schreiben Sie im Quellcode Ihre Angaben in einen Kommentar. Im EST-Abgabesystem können Sie modifizierte Dateien mehrfach abgeben. Nur die zuletzt hochgeladene Version wird bewertet.