ОГЛАВЛЕНИЕ

Разбор домашнего задания

Level 1

Последовательность `([{}])` является правильной, а последовательности `([)],` `{()]` правильными не являются. Докажите это используя стек!

Level 2

Дан односвязный список. Написать функцию или блок схему, определяющую, образуют ли его элементы симметричную последовательность. Для решения задачи использовать стек и очередь. Функция будет возвращать значение **true**, если список является симметричным, **false** – в противном случае.
По определению пустой список является симметричным. Поэтому, если список пуст, то возвращаем значение **true**. Для проверки симметричности списка нужно проверить на равенство все пары элементов, равноотстоящих от середины списка. Каждая пара содержит один элемент из первой половины списка и один – из второй. Элементы первой половины списка последовательно заносятся в очередь, второй половины в стек. Если количество элементов списка будет нечетным, то серединный элемент будет симметричен сам себе и не будет помещен ни в очередь, ни в стек.

Level 1

JAVA	JAVA SCRIPT
``java public static void main(String[] args) { String str1 = "([{}])"; String str2 = "([)]"; String str3 = "{()]"; System.out.println(str1 + " " + validate(str1)); // [({})] true System.out.println(str2 + " " + validate(str2)); // ([)] false System.out.println(str3 + " " + validate(str3)); // {()] false } public static boolean validate(String str) { Stack<Character> stack = new Stack<>(); for (char c : str.toCharArray()) { if (c == '(' || c == '[' || c == '{') { stack.push(c); } else if (c == ')' || c == ']' || c == '}') { if (stack.isEmpty()) { return false; } char top = stack.pop(); if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) { return false; } } } return stack.isEmpty(); } ```	```javascript function validate(str) { const stack = []; for (let i = 0; i < str.length; i++) { const c = str[i]; if (c === '(' || c === '[' || c === '{') { stack.push(c); } else if (c === ')' || c === ']' || c === '}') { if (stack.length === 0) { return false; } const top = stack.pop(); if ((c === ')' && top !== '(') || (c === ']' && top !== '[') || (c === '}' && top !== '{')) { return false; } } } return stack.length === 0; } const str1 = "([{}])"; const str2 = "([)]"; const str3 = "{()]"; console.log(str1 + " " + validate(str1)); // ([{}]) true console.log(str2 + " " + validate(str2)); // ([)] false console.log(str3 + " " + validate(str3)); // {()] false ```
Как работает этот код:	Как работает этот код:

• Создается стек stack, в который будут добавляться скобки • Входная строка str проходится посимвольно, и если символ - это открывающая скобка ((, [, {), она добавляется в стек • Если символ - это закрывающая скобка (),], }), то мы проверяем, соответствует ли закрывающая скобка последней открывающей скобке в стеке. Если они не соответствуют друг другу, возвращается false, что означает, что строка неправильно сбалансирована. • Если в конце обхода стек оказывается пустым, то это означает, что все скобки были закрыты в правильном порядке, и строка сбалансирована.	• Создается пустой массив stack, в который будут добавляться скобки • Входная строка str проходится посимвольно, и если символ - это открывающая скобка ((, [, {), она добавляется в стек • Если символ - это закрывающая скобка (),], }), то мы проверяем, соответствует ли закрывающая скобка последней открывающей скобке в стеке. Если они не соответствуют друг другу, возвращается false, что означает, что строка неправильно сбалансирована. • Если в конце обхода длина стека равна нулю, то это означает, что все скобки были закрыты в правильном порядке, и строка сбалансирована.

Level 2

JAVA

```java
public static boolean isSymmetric(LinkedList<Integer>
list) {
    if (list.isEmpty()) { // проверка на пустой список
        return true;
    }

    int size = list.size(); // размер списка
    int mid = size / 2; // середина списка

    Queue<Integer> queue = new LinkedList<Integer>(); //
очередь для первой половины списка
    Stack<Integer> stack = new Stack<Integer>(); // стек
для второй половины списка
```

JAVA SCRIPT

```javascript
class ListNode {
    constructor(val, next = null) {
        this.val = val;
        this.next = next;
    }
}

function isSymmetric(head) {
    if (!head) {
        return true;
    }

    let slow = head;
```

```
    // Заносим первую половину списка в очередь
    for (int i = 0; i < mid; i++) {
        queue.add(list.get(i));
    }

    // Заносим вторую половину списка в стек
    for (int i = mid; i < size; i++) {
        stack.push(list.get(i));
    }

    // Сравниваем элементы из очереди и стека
    while (!queue.isEmpty() && !stack.isEmpty()) {
        if (!queue.remove().equals(stack.pop())) { // Если
элементы не равны, то список не симметричен
            return false;
        }
    }

    return true; // Если все элементы совпали, то список
симметричен
}

public static void main(String[] args) {
    LinkedList<Integer> list = new LinkedList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(2);
    list.add(1);

    boolean isSymmetric = isSymmetric(list);

    if (isSymmetric) {
        System.out.println("Список симметричен");
    } else {
        System.out.println("Список не симметричен");
    }
}
```

```
    let fast = head;

    while (fast && fast.next) {
        slow = slow.next;
        fast = fast.next.next;
    }

    const stack = [];
    const queue = [];

    let current = head;

    while (current !== slow) {
        queue.push(current.val);
        current = current.next;
    }

    if (fast) {
        // если список имеет нечетное количество
элементов,
        // то серединный элемент не является симметричным
        current = current.next;
    }

    while (current) {
        stack.push(current.val);
        current = current.next;
    }

    while (queue.length && stack.length) {
        if (queue.shift() !== stack.pop()) {
            return false;
        }
    }

    return true;
}

const list1 = new ListNode(1, new ListNode(2, new
ListNode(3, new ListNode(2, new ListNode(1)))));
const list2 = new ListNode(1, new ListNode(2, new
ListNode(3, new ListNode(3, new ListNode(2, new
```

```
ListNode(1))))));
const list3 = new ListNode(1, new ListNode(2, new
ListNode(3, new ListNode(4, new ListNode(5)))));


console.log(isSymmetric(list1)); // ожидаем true
console.log(isSymmetric(list2)); // ожидаем true
console.log(isSymmetric(list3)); // ожидаем false
```

Что такое Хэш (Hash)?

Хеш-ом называется результат математического преобразования любого объема информации в короткий и уникальный набор символов, который относится только к этим входным данным и служит как идентификатор этих данных.

Хэш или **хэш-функция** — это алгоритм, преобразовывающий произвольный массив данных в состоящую из букв и цифр строку фиксированной длины.

Пример:

Result for sha1: 0bf9a98c82a1fcb21de95bea4213fc1a2173081d

Одни из самых распространенных hash-functions можно попробовать самим на ресурсе:
SHA1 online

Попробуйте и вы свое имя 😇

Hashing

Обязательные свойства **Hash:**

- Хэш всегда **уникален** для каждого массива информации, иногда случаются коллизии, когда для разных входных блоков информации вычисляются одинаковые хеш-коды.

- При **малейшем** изменении входной информации - **хэш полностью меняется.**

- Хэш-функция **не обратима** и не позволяет восстанавливать исходный массив информации из символьной строки.

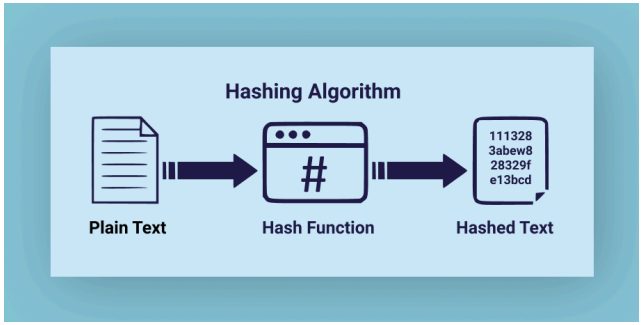
- Хэширование позволяет достаточно быстро вычислить нужный хэш для достаточно большого объема информации.

- Алгоритм работы хэш-функции, как правило, делается открытым, чтобы при необходимости можно было оценить ее стойкость к восстановлению начальных данных по выдаваемому хэшу.

- Хэш-функция должна уметь приводить любой объем данных к числу заданной длины.

Использование Hash

- ☐ Работа с большими объемами информации
- ☐ Проверка целостности данных при передаче
- ☐ Шифрование
- ☐ Электронные цифровые подписи
- ☐ Хранение паролей



Hash или **Hash-function** – одна из основных составляющих современной криптографии и алгоритмов **блокчейна.**

Хеш таблицы

Хеш-таблица в Java представляет собой структуру данных, которая использует хеш-функцию для хранения и поиска элементов. Ключи элементов преобразуются в хеши с помощью хеш-функции, которая возвращает индекс в массиве. Элементы с одинаковыми хешами хранятся в одной ячейке массива.

Сложность поиска элемента в хеш-таблице равна **O(1)** в среднем случае и **O(n)** в худшем случае.

```
Встроенная поддержка хеш-таблиц есть во многих языках программирования, включая
                        Python, Java, Ruby, PHP, C#
```

В Java реализация хеш-таблицы осуществляется классом HashMap. Он предоставляет методы для добавления, удаления и поиска элементов в хеш-таблице. Ключи и значения могут быть любых типов объектов.

P.S.: Кроме того, в Java есть также другие классы, реализующие хеш-таблицы, такие как LinkedHashMap, TreeMap и Hashtable.

О коллизиях (конфликтах)

SHA-256

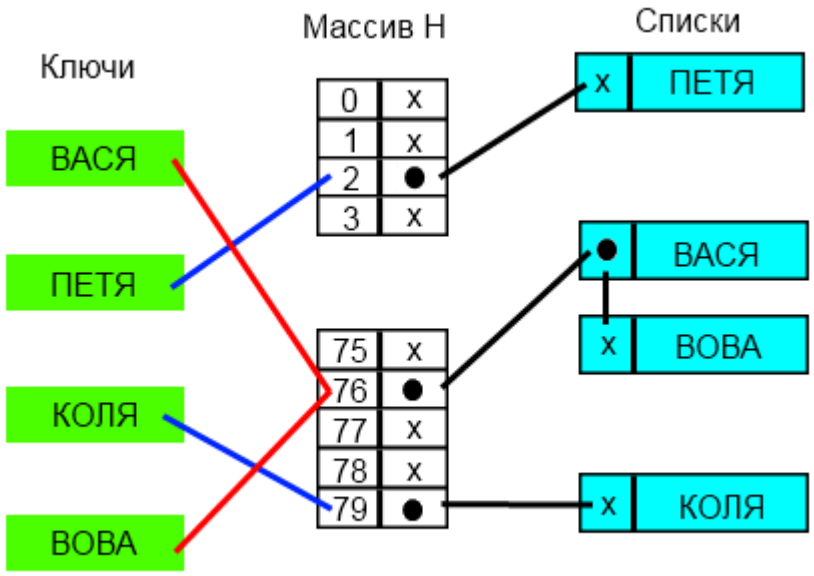
256 бит - это 2^256 соответствий, то есть 2^256 различных входов имеют свой уникальный хеш.

Коллизия происходит, когда разные входные данные производят одинаковый хеш.

Хеш-функция считается **устойчивой** к коллизиям до того момента, пока не будет обнаружена пара сообщений, дающая одинаковый выход.

Хеш-функция считается **устойчивой** к коллизиям, когда вероятность обнаружения коллизии настолько мала, что для этого потребуются миллионы лет вычислений.

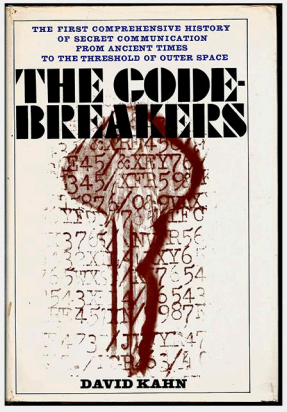
> Хеш-функций без коллизий не существует 😜



Классная работа

Пишем **ПСЕВДОКОД *или* БЛОК СХЕМУ** по применению хеш функции для сохранения пароля и последующей проверки.

Для чтения и для просмотра

📚 Для чтения	👀 Для просмотра
Дэвид Кан **"Взломщики кодов"** - это книга о развитии компьютерной безопасности и об истории группы хакеров, которая называется Легион Хакеров. Книга рассказывает о том, как эта группа разработала новые технологии взлома компьютеров и сетей, а также о том, как они боролись с правоохранительными органами, чтобы сохранить свою свободу. Книга также обсуждает широкий спектр тем, связанных с компьютерной безопасностью, включая криптографию, теорию информации и принципы защиты от взлома. 	**Игра в имитацию,** фильм об Алане Тьюринге и его самом большом криптографическом взломе. 

Примеры реализации

Вот пример кода на Java, который демонстрирует шифрование строки с использованием алгоритма MD5:

```java
import java.security.MessageDigest;

public class MD5Example {
    public static void main(String[] args) throws Exception {
        // Шифруем строку
        String originalString = "MySuperPassword";
        String encryptedString = encrypt(originalString);
        System.out.println("Encrypted string: " + encryptedString);
    }

    private static String encrypt(String originalString) throws
Exception {
        MessageDigest md = MessageDigest.getInstance("MD5");
        md.update(originalString.getBytes());
        byte[] digest = md.digest();
        StringBuilder sb = new StringBuilder();
        for (byte b : digest) {
            sb.append(String.format("%02x", b & 0xff));
        }
        return sb.toString();
    }
}
```

Вот пример кода на JavaScript, который демонстрирует работу хеш-функции MD5:

```javascript
function md5(string, key, raw) {
  if (!key) {
    if (!raw) {
      return hexMD5(string)
    }
    return rawMD5(string)
  }
  if (!raw) {
    return hexHMACMD5(key, string)
  }
  return rawHMACMD5(key, string)
}
```

Официальный репозиторий:

https://github.com/blueimp/JavaScript-MD5

Вот пример реализации хеш-функции на Python: (https://replit.com/languages/python3)

```python
def hash_function(data):
  # Инициализируем хеш нулевым значением
  hash = 0
  # Проходим по каждому байту в данных
  for byte in data:
    # Добавляем значение байта к хешу
    hash += byte
  # Возвращаем хеш
  return hash

#Run
hash = hash_function(b"Gegham")
print("Hash is: ", hash)
```

Вопросы и ответы

Что такое хеш-функция?
Каковы основные свойства хеш-функций?
Какие алгоритмы хеширования вы знаете?
Что такое коллизия хеш-функции?
Для чего используются хеш-таблицы?
Приведите пример использования хеша.

Вопрос на логику, поиск алгоритма решения

Задача

Вы в комнате!

Всего таких комнат N штук. Каждая комната соседствует с двумя другими. Во всех комнатах **рандомно** горит свет.

Двери автоматически раскрываются.

Любая комната имеет:

1) Лампочка света (лампочка не нагревается)
2) Выключатель света (относится к лампочке в комнате)
3) Дверь <L> (обозначение двери)
4) Дверь <R> (обозначение двери)

Вы можете:

1) включать или выключать свет в комнате
2) заходить в дверь <L> и попадать в комнату за этой дверью
3) заходить в дверь <R> и попадать в комнату за этой дверью

Вопрос:

Как посчитать количество комнат? (самый оптимальный алгоритм)

Напишите еще одну строку

```
1
11
21
1211
111221
```

Домашнее задание

Level 1

Приведите несколько примеров применения хеш-функций в реальной жизни.
Создайте ключ **RSA** и скиньте его в открытом виде.

Level 2

Изучить несколько различных алгоритмов хеширования, таких как MD5, SHA-1, SHA-256 и SHA-512. Исследовать примеры применения хешей в реальном мире, такие как Bitcoin, блокчейн, пароли и другие.