Частное учреждение профессионального образования
«Высшая школа предпринимательства»
(ЧУПО «ВШП»)

ДНЕВНИК ПРАКТИКИ

Вид практики (учебная, производственная, преддипломная): _____

Установленный по КУГ срок прохождения практики: с ___.___.20__г. по ___.___.20__г.

Место прохождения практики (наименование организации):

_____

_____

_____

Выполнил студент __-го курса	_____ *(подпись)*	_____ *(фамилия, имя, отчество)*
Руководитель от образовательной организации	_____ *(подпись)*	_____ *(ученая степень, фамилия, имя, отчество)* _____ _____ *(должность)*
Руководитель от предприятия	_____ *(подпись)*	_____ *(ученая степень, фамилия, имя, отчество)* _____ _____ *(должность)*

Тверь, 2024 г.

Содержание

Введение

Алгоритм – это последовательность действий для решения определенной задачи или выполнения конкретной операции. [1]

Основные Свойства Алгоритмов

Дискретность:

Алгоритм состоит из отдельных, дискретных шагов, каждый из которых выполняется за конечный отрезок времени

Результативность:

Выполнение алгоритма должно привести к конкретному результату, без неопределенностей. Результат может быть успешным или указывать на отсутствие решения

Детерминированность:

На каждом шаге алгоритма следующее действие должно быть однозначно определено. Для одних и тех же входных данных алгоритм должен выдавать один и тот же результат

Виды и Структуры Алгоритмов

Алгоритмы можно классифицировать по различным критериям, но основными структурами являются:

Линейная последовательность:

Набор действий, выполняемых последовательно друг за другом

Выбор(Разветвление):

Набор действий, начинающийся с проверки некоторого условия, с возможными различными действиями в зависимости от результата проверки

Повторение(Цикл):

Набор действий, которые повторяются до тех пор, пока не будет достигнуто определенное условие

Применение Алгоритмов

Алгоритмы используются во всех областях, где требуется решение определенных проблем или выполнение операций:

Информатика и Программирование:

Алгоритмы являются основой для разработки программного обеспечения, обеспечивая эффективное и корректное выполнение задач

Математика:

Алгоритмы используются для решения математических задач, таких как нахождение наибольшего общего делителя или решение линейных и квадратных уравнений. [1]

Цель:

Целью данной работы является изучение алгоритмов Хаффмана и Чисел Фиббоначи, а также решение заданий по этим алгоритмам.

Задачи:

1. Изучить теоретические основы алгоритма Хаффмана и чисел Фиббоначи.
2. Реализовать алгоритм Хаффмана на выбранном программном языке.
3. Вычислить числа Фиббоначи различными методами и сравнить их эффективность.
4. Анализировать результаты и оценивать правильность и эффективность решений.
5. Написать отчет, отражающий весь процесс и полученные знания.

1. Алгоритмы для вычисления чисел Фибоначчи

1.1.Числа Фиббоначи

Числа Фиббоначи являются одной из самых интересных и широко применяемых математических последовательностей. Числа Фиббоначи образуют числовую последовательность, где первые два числа равны 0 и 1, а каждое последующее число является суммой двух предыдущих.

Эти числа были введены итальянским математиком Леонардо Пизанским, известным как Фиббоначчи, в его работе "Liber Abaci" в 1202 году. Однако, они были известны индийским математикам,such as Гопала и Хемачандра, ещё раньше. [3]

Разборы и объяснения решения заданий

1.2. Вычисление n-го числа Фибоначчи с использованием рекурсивного алгоритма

```
function fib(n) {
    if (n === 0) return 0;
    if (n === 1) return 1;

    return fib(n - 1) + fib(n - 2);
}

function measureTime(n) {
    const startTime = Date.now();
    const result = fib(n);
    const endTime = Date.now();

      console.log(`fib(${n}) = ${result}, время выполнения:
${endTime - startTime} мс`);
}

measureTime(4);
measureTime(10);
measureTime(14);
measureTime(23);
measureTime(30);
```

Данный код представляет собой простую реализацию функции для вычисления чисел Фибоначчи и измерения времени, необходимого для этого вычисления.

Функция fib(n)

Эта функция вычисляет n-е число Фибоначчи используя рекурсивный подход.

Как она работает:

Базовые случаи:

Если n равно 0, функция возвращает 0, поскольку первое число в последовательности Фибоначчи равно 0.

Если n равно 1, функция возвращает 1, поскольку второе число в последовательности Фибоначчи равно 1.

Рекурсивный шаг:

Для любого другого значения n, функция вызывает сама себя с аргументами n-1 и n-2 и возвращает сумму этих двух вызовов.

Функция measureTime(n)

Эта функция измеряет время, необходимое для вычисления n-го числа Фибоначчи используя функцию fib(n).

Как она работает:

Запись времени начала:

const startTime = Date.now(); записывает текущее время в миллисекундах перед вызовом функции fib(n).

Вычисление числа Фибоначчи:

const result = fib(n); вызывает функцию fib(n) и сохраняет результат в переменной result.

Запись времени окончания:

const endTime = Date.now(); записывает текущее время в миллисекундах после завершения вызова функции fib(n).

Вывод результатов:

console.log(fib(${n}) = ${result}, время выполнения: ${endTime - startTime} мс); выводит результат вычисления числа Фибоначчи и время, затраченное на это вычисление.

В конце кода вызываются несколько раз функция measureTime(n) с разными значениями n для измерения времени вычисления чисел Фибоначчи для этих значений. [2]

1.3. Вычисление n-го числа Фибоначчи с использованием цикла

```
function fib(n) {
  if (n < 1 || n > 32) {
    return "Число должно быть от 1 до 32";
  }

  if (n <= 2) return 1;

  let prev = 1;
  let current = 1;

  for (let i = 3; i <= n; i++) {
    let temp = current;
    current = prev + current;
    prev = temp;
  }

  return current;
}

function measureTime(n) {
  const startTime = Date.now();
  const result = fib(n);
  const endTime = Date.now();

  console.log(
      `fib(${n}) = ${result}, время выполнения: ${endTime - 
startTime} мс`
  );
}

console.log("Тестирование производительности:");
measureTime(5);
```

```
measureTime(10);
measureTime(15);
measureTime(20);
measureTime(30);
```

Этот код представляет собой более эффективную реализацию функции для вычисления чисел Фибоначчи и измерения времени, необходимого для этого вычисления, используя итерационный подход.

На этот раз функция fib(n) вычисляет n-е число Фибоначчи используя итерационный подход, который намного более эффективен, чем рекурсивный.

Сначала идет проверка, находится ли входное число n в диапазоне от 1 до 32. Если нет, функция возвращает сообщение об ошибке.

Базовые случаи.

Если n равно 1 или 2, функция возвращает 1, поскольку первые два числа в последовательности Фибоначчи равны 1.

Итерационный цикл

Инициализируются две переменные prev и current, обе равные 1, которые представляют первые два числа в последовательности Фибоначчи.

Цикл for начинается с i = 3 и продолжается до n.

В каждой итерации цикла:

Временная переменная temp сохраняет текущее значение current.

Новое значение current вычисляется как сумма предыдущего и текущего значений (prev + current).

Значение prev обновляется до предыдущего значения current, сохраненного в temp.

В конце также замеряется время выполнения алгоритма. [2]

1.4. Вычисление n-го числа Фибоначчи с записью числового ряда в массив

```
function fib(n) {
  if (n < 1 || n > 40) {
     throw new Error("Число n должно быть в диапазоне 1 ≤ n ≤ 40");
  }

  const fibArray = [0, 1];

  for (let i = 2; i <= n; i++) {
    if (fibArray[i] === undefined) {
      fibArray[i] = fibArray[i - 1] + fibArray[i - 2];
    }
  }

  console.log(fibArray.slice(0, n + 1));
  return fibArray.slice(0, n + 1);
}
fib(8);
```

Данный код представляет собой реализацию функции для вычисления чисел Фибоначчи с использованием динамического программирования и хранения результатов в массиве. [2] Обьяснение кода:

1. Вторая строка кода проверяет, находится ли входное число n в диапазоне от 1 до 40. Если нет, функция генерирует ошибку.
2. Создается массив fibArray с первыми двумя числами Фибоначчи: 0 и
3. Цикл for начинается с i = 2 и продолжается до n.

В каждой итерации цикла:

Проверяется, уже вычислено значение fibArray[i]. Если нет, оно вычисляется как сумма двух предыдущих значений (fibArray[i - 1] + fibArray[i - 2]).

4. Результат сохраняется в массиве fibArray.
5. Выводится срез массива fibArray от 0 до n (включительно) в консоль.

1.5. Вычисление n-го числа Фибоначчи при помощи формулы Бине

```
function fib(n) {
  if (n < 1 || n > 64) {
    throw new Error("Число n должно быть в диапазоне 1 ≤ n ≤ 64");
  }

  const sqrt5 = Math.sqrt(5);
  const phi = (1 + sqrt5) / 2;
  const psi = (1 - sqrt5) / 2;

  const fibNumber = Math.round((Math.pow(phi, n) - Math.pow(psi, n)) / sqrt5);

  console.log(fibNumber);
  return fibNumber;
}

fib(32);
```

Код представляет собой реализацию функции для вычисления чисел Фибоначчи используя математическую формулу, известную как формула Бине.

1. Условие на второй строке как и всегда проверяет находится ли входное число в правильном диапазоне.
2. Вычисляются необходимые константы:

sqrt5: квадратный корень из 5.

phi: золотое сечение (приблизительно 1.618).

psi: отрицательное обратное золотое сечение (приблизительно -0.618).

3. Затем используется формула Бине для вычисления n-го числа Фибоначчи
4. Вычисленное число Фибоначчи выводится в консоль.
5. Функция возвращает вычисленное число Фибоначчи.
6. В конце кода вызывается функция fib(n) с аргументом 32. [2]

1.6. Определение четности n-го большого числа Фибоначчи

```
function fib_eo(n) {
  if (n < 1 || n > 1e6) {
     throw new Error("Число n должно быть в диапазоне 1 ≤ n ≤
10^6");
  }

  if (n === 1) {
    console.log("odd");
    return "odd";
  }

  if (n === 2) {
    console.log("odd");
    return "odd";
  }

  let prev = 0;
  let curr = 1;

  for (let i = 2; i <= n; i++) {
    const next = (prev + curr) % 10;
    prev = curr;
    curr = next;
  }

  const result = curr % 2 === 0 ? "even" : "odd";
  console.log(result);
  return result;
}

fib_eo(841645);
```

Предоставленный код имеет функцию fib_eo для определения, является ли n-е число Фибоначчи четным или нечетным, без необходимости вычислять само число Фибоначчи полностью.

1. Вторая строка кода проверяет, находится ли входное число n в диапазоне от 1 до 1 миллиона. Если нет, функция генерирует ошибку.
2. Если n равно 1 или 2, функция выводит и возвращает "odd", поскольку первые два числа Фибоначчи (0 и 1) нечетны.
3. Инициализируются переменные prev и curr с первыми двумя числами Фибоначчи (0 и 1).

4. Цикл for начинается с i = 2 и продолжается до n.

В каждой итерации цикла:

Вычисляется следующее число Фибоначчи по модулю 10 (next = (prev + curr) % 10), что позволяет избежать больших чисел и сохранять только последнюю цифру.

5. Переменные prev и curr обновляются для следующей итерации.
6. После завершения цикла определяется, является ли последняя цифра (curr) четной или нечетной.
7. Результат выводится в консоль и возвращается функцией.
8. В конце кода вызывается функция fib_eo(n) с аргументом 841645. [2]

2.Алгоритмы Хаффмана

2.1. Кодирование строки по алгоритму Хаффмана

Алгоритм Хаффмана является методом без потерь сжатия данных, основанным на построении двоичного дерева, где каждому символу текста соответствует уникальный код переменной длины. Этот алгоритм основан на принципе присвоения более коротких кодов символам, которые появляются чаще в тексте, а более длинных кодов символам, которые появляются реже. Это позволяет минимизировать среднюю длину кодового слова и, таким образом, уменьшить общий объем данных. [4]

Разбор кода

1. **Построение Частотного Словаря**

Сначала код строит частотный словарь, где каждому уникальному символу в тексте соответствует его частота появления. Это делается с помощью карты (Map) в функции buildHuffmanTree.

2. **Создание Узлов Дерева**

Для каждого символа в частотном словаре создается узел дерева (HuffmanNode), содержащий символ, его частоту, и ссылки на левый и правый дочерние узлы. Эти узлы добавляются в массив.

3. **Построение Дерева Хаффмана**

Алгоритм сортирует узлы по их частотам и объединяет два узла с наименьшими частотами в новый узел, пока не останется один корневой узел. Это делается в цикле while в функции buildHuffmanTree.

4. **Построение Кодов Хаффмана**

После построения дерева Хаффмана, функция buildCodes рекурсивно проходит по дереву и присваивает каждому символу уникальный код. Код формируется путем добавления '0' для левого дочернего узла и '1' для правого дочернего узла.

5. **Кодирование Текста**

Функция huffmanEncode использует построенное дерево и карту кодов для кодирования входного текста. Она проходит по тексту, заменяя каждый символ его соответствующим кодом Хаффмана.

6. **Вывод Результатов**

В конце, функция выводит длину кодовой карты, длину закодированной строки, сами коды для каждого символа и саму закодированную строку.

7. **Пример Вызова**

Вызов huffmanEncode("Errare humanum est.") демонстрирует работу алгоритма на конкретном тексте. [4] см. Приложение 1

2.2. Декодирование строки по алгоритму Хаффмана

Декодирование Хаффмана является процессом восстановления текста из закодированной строки, которая была сжата с помощью алгоритма Хаффмана.

Разбор кода

1. **Входные Данные**

Функция принимает входные данные в виде массива строк. Первая строка содержит количество уникальных символов и длину закодированной строки. Следующие строки представляют собой карту кодов Хаффмана, где каждая строка является парой "символ: код". Последняя строка содержит саму закодированную строку.

2. **Построение Карты Кодов**

Функция начинает с построения карты кодов (codeMap) из входных данных. Она проходит по каждой строке, содержащей пару "символ: код", и добавляет эти пары в карту. Символы и коды извлекаются из строк, очищаются от лишних символов (например, кавычек), и затем добавляются в карту.

3. **Декодирование Закодированной Строки**

После построения карты кодов, функция приступает к декодированию закодированной строки. Она проходит по каждому символу в закодированной строке, накапливая текущий код в переменной currentCode. Когда накопленный код соответствует ключу в карте кодов, соответствующий символ добавляется к декодированной строке, а переменная currentCode сбрасывается.

4. **Вывод Декодированной Строки**

После завершения декодирования, функция выводит декодированную строку в консоль. [4]

см. Приложение 2

Заключение

В ходе работы над проектом по алгоритмам Хаффмана и числам Фибоначчи, я приобрёл глубокие знания и навыки в нескольких ключевых областях. Ниже приведены доводы о том, чему я научился и что полезного узнал:

Алгоритмы Хаффмана

Понимание Основного Принципа: Я узнал, что алгоритм Хаффмана основан на использовании префиксных кодов для сжатия данных. Это позволяет эффективно сжимать текст, присваивая более короткие коды более частым символам.

Построение Дерева Хаффмана: Я изучил процесс построения дерева Хаффмана, который включает в себя объединение узлов с наименьшими частотами. Это помогло мне понять, как структурировать данные для эффективного сжатия.

Кодирование и Декодирование: Я научился кодировать и декодировать строки с помощью дерева Хаффмана. Это включает в себя проход по дереву для присвоения кодов символам и восстановление исходного текста из закодированной строки.

Практическая Реализация: Я реализовал функции для кодирования и декодирования строк Хаффмана на JavaScript, что помогло мне понять практические аспекты алгоритма.

Числа Фибоначчи

Определение и Структура: Я узнал о последовательности Фибоначчи, где каждое число является суммой двух предыдущих. Это помогло мне понять базовую структуру и закономерности в этой последовательности.

Алгоритмы Вычисления: Я изучил несколько алгоритмов для вычисления чисел Фибоначчи, включая рекурсивный, итерационный, динамическое

программирование и матричный методы. Каждый метод имеет свои преимущества и недостатки, и я понял, когда использовать каждый из них.

Практическая Реализация: Я реализовал функции для вычисления чисел Фибоначчи на JavaScript, используя разные методы, что помогло мне понять практические аспекты этих алгоритмов.

Полезные Знания и Навыки

Алгоритмическое Мышление: Работа над этими проектами помогла мне развить алгоритмическое мышление и понять, как структурировать проблемы для эффективного решения.

Эффективность Алгоритмов: Я узнал о важности анализа сложности алгоритмов и выбора наиболее эффективного метода для решения конкретной задачи.

Практическая Реализация: Реализация алгоритмов на JavaScript помогла мне понять, как переводить теоретические знания в практические результаты.

Анализ и Оценка: Я научился анализировать и оценивать результаты, что включает в себя измерение времени выполнения и оценку эффективности различных методов.

В целом, работа над этими проектами была очень полезной и информативной. Я приобрёл глубокие знания о структурах данных, алгоритмах и их практическом применении. Эти навыки и знания будут полезны мне в будущих проектах и карьере в области информатики и компьютерных наук.

Список источников

1. Алгоритмы [Электронный ресурс] / - https://znanierussia.ru/articles/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC

https://math-it.petrsu.ru/users/semenova/Informatika/DOC/Sam_Izuch/Algoritm.pdf

2. Алгоритм для вычисления чисел Фибоначчи [Электронный ресурс] / -

https://yusmpgroup.ru/vychislenie-chisel-fibonachchi-cikl

3. Числа Фибоначчи [Электронный ресурс] / - https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8

http://e-maxx.ru/algo/export_fibonacci_numbers

4. Алгоритм Хаффмана [Электронный ресурс] / - https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0

https://cyberleninka.ru/article/n/szhatie-dannyh-algoritm-haffmana

http://www.codenet.ru/progr/alg/huffman.php

Приложение 1. Кодирование строки по алгоритму Хаффмана

```
class HuffmanNode {
  constructor(char, freq) {
    this.char = char;
    this.freq = freq;
    this.left = null;
    this.right = null;
  }
}

function buildHuffmanTree(text) {
  const freqMap = new Map();

  for (const char of text) {
    freqMap.set(char, (freqMap.get(char) || 0) + 1);
  }

  const nodes = [];
  for (const [char, freq] of freqMap.entries()) {
    nodes.push(new HuffmanNode(char, freq));
  }

  while (nodes.length > 1) {
    nodes.sort((a, b) => a.freq - b.freq);

    const left = nodes.shift();
    const right = nodes.shift();

    const merged = new HuffmanNode(null, left.freq + right.freq);
    merged.left = left;
    merged.right = right;

    nodes.push(merged);
  }

  return nodes[0];
}

function buildCodes(node, prefix = "", codeMap = {}) {
  if (!node) return;

  if (node.char !== null) {
    codeMap[node.char] = prefix;
  }

  buildCodes(node.left, prefix + "0", codeMap);
  buildCodes(node.right, prefix + "1", codeMap);
return codeMap;
```

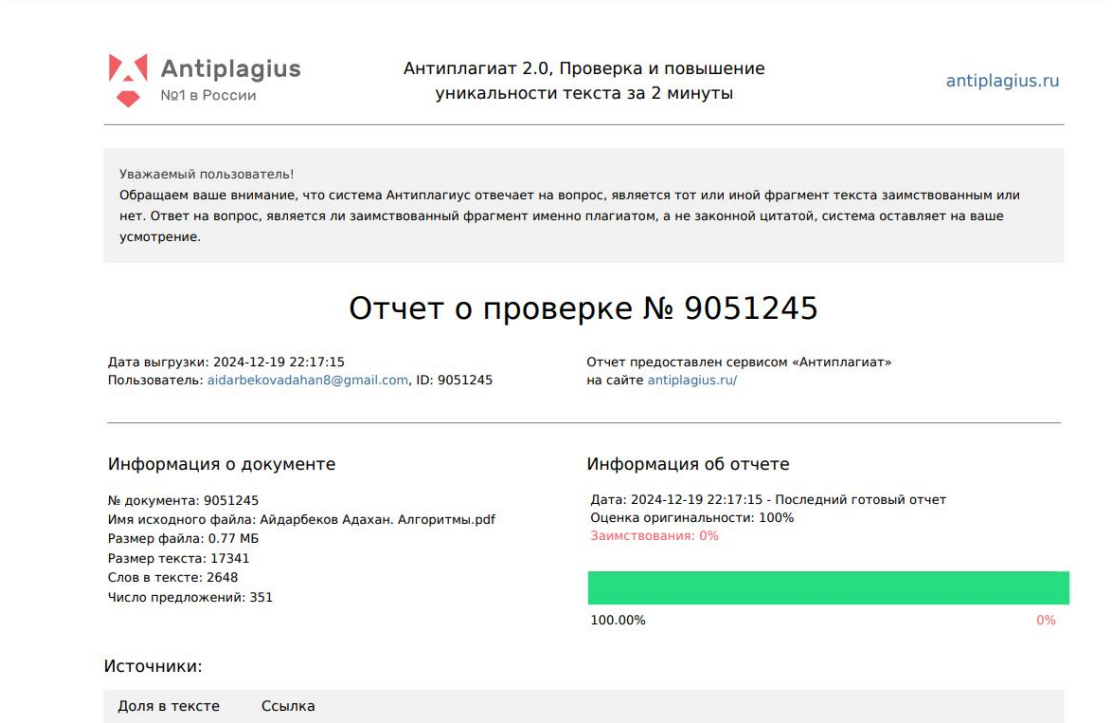
```
}

function huffmanEncode(text) {
  const root = buildHuffmanTree(text);
  const codeMap = buildCodes(root);

  let encodedString = "";
  for (const char of text) {
    encodedString += codeMap[char];
  }

                        console.log(Object.keys(codeMap).length,
encodedString.length);

  for (const [char, code] of Object.entries(codeMap)) {
    console.log(`'${char}': ${code}`);
  }

  console.log(encodedString);
  return { codeMap, encodedString };
}

huffmanEncode("Errare humanum est.");
```

Приложение 2. Декодирование строки по алгоритму Хаффмана

```
function huffmanDecode() {
  const input = [
    "12 60",
    "' ': 1011",
    "'.': 1110",
    "'D': 1000",
    "'c': 000",
    "'d': 001",
    "'e': 1001",
    "'i': 010",
    "'m': 1100",
    "'n': 1010",
    "'o': 1111",
    "'s': 011",
    "'u': 1101",
     "100011110001001101000111111011001010011000010110011010111
110",
  ];

    const   [uniqueCount,   encodedLength]   =   input[0].split("
").map(Number);
```

```
  const codeMap = {};
  for (let i = 1; i <= uniqueCount; i++) {
    const [symbol, code] = input[i]
      .split(":")
      .map((str) => str.trim().replace(/'/g, ""));
    codeMap[code] = symbol;
  }

  const encodedString = input[uniqueCount + 1];

  let decodedString = "";
  let currentCode = "";
  for (let i = 0; i < encodedString.length; i++) {
    currentCode += encodedString[i];
    if (currentCode in codeMap) {
      decodedString += codeMap[currentCode];
      currentCode = "";
    }
  }

  console.log(decodedString);
}

huffmanDecode();
```

Приложение 3. Антиплагиат



Antiplagius
№1 в России

Антиплагиат 2.0, Проверка и повышение уникальности текста за 2 минуты

antiplagius.ru

Уважаемый пользователь!
Обращаем ваше внимание, что система Антиплагиус отвечает на вопрос, является тот или иной фрагмент текста заимствованным или нет. Ответ на вопрос, является ли заимствованный фрагмент именно плагиатом, а не законной цитатой, система оставляет на ваше усмотрение.

Отчет о проверке № 9051245

Дата выгрузки: 2024-12-19 22:17:15
Пользователь: aidarbekovadahan8@gmail.com, ID: 9051245

Отчет предоставлен сервисом «Антиплагиат»
на сайте antiplagius.ru/

Информация о документе

№ документа: 9051245
Имя исходного файла: Айдарбеков Адахан. Алгоритмы.pdf
Размер файла: 0.77 МБ
Размер текста: 17341
Слов в тексте: 2648
Число предложений: 351

Информация об отчете

Дата: 2024-12-19 22:17:15 - Последний готовый отчет
Оценка оригинальности: 100%
Заимствования: 0%

100.00% 0%

Источники:

Доля в тексте Ссылка

Приложение 4. QR-код

Ссылка на репозиторий github: https://github.com/adeqoou/algorithms