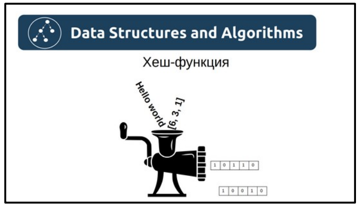
Поиск подстроки.
Алгоритм Рабина — Карпа

Список лекций необходимых для занятия

Перед просмотром этого занятия нужно просмотреть следующие лекции.



Хеш-функция

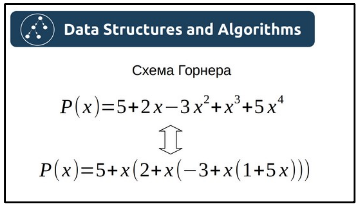


Схема Горнера

Алгоритм Рабина — Карпа

Алгоритм Рабина-Карпа алгоритм который в своей работе использует хеширование. Данный алгоритм отлично подходит для одновременного поиска множества подстрок (одинаковой длины) в строке. В тоже время этот алгоритм довольно редко используется для поиска одиночного образца. Он был разработан в 1987 году Михаэлем Рабином и Ричардом Карпом.

Сведение о алгоритме

Сложность по времени в наихудшем случае O(n·m)

n — длина строки

m — длина подстроки

Стоит отметить, что худший случай реализуется довольно редко. В среднем и лучшем случае сложность составляет O(n)

Принцип работы алгоритма

1) Вычисляется хеш-код искомой подстроки (предположим длина подстроки m).
2) Начиная с первого символа выделяем в базовой строке часть длиной m.
    - Хеш части строки и подстроки совпадает. Проверяем на равенство искомою подстроку и часть строки. В случае равенства возвращаем положительный результат.
    - Не совпадают. Сдвигаем выделяемую часть на один символ вправо. Если часть строки достигла границы базовой строки, возвращаем отрицательный результат. Поиск не успешен. Если не достигла переходим к первому подпункту.

Графическое пояснение алгоритма

A	w	e	r	s	o	m	e		a	p	p	l	e

- Базовая строка

s	o	m	e

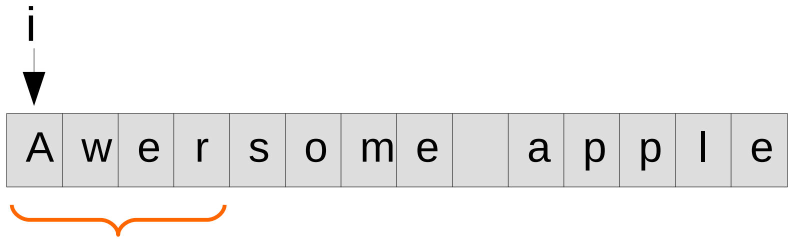
- Искомая строка

s	o	m	e

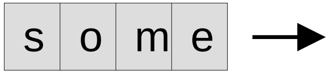
→ hash = 3536116

На первом этапе вычисляем хеш искомой строки.

Графическое пояснение алгоритма



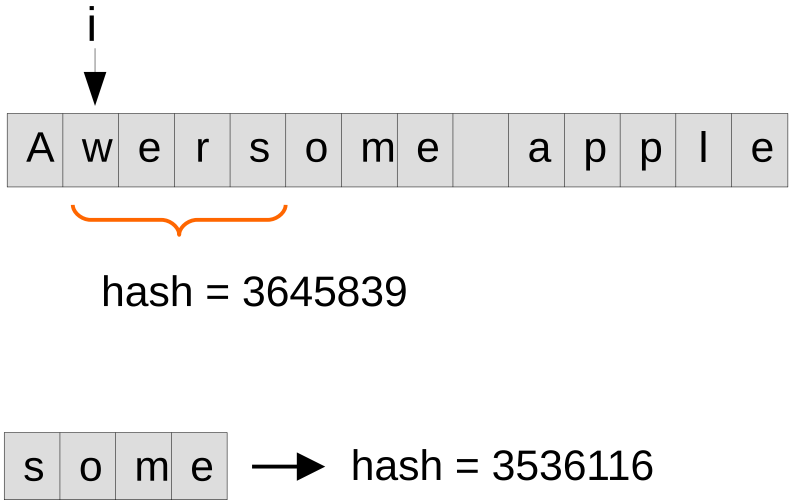
hash = 2054019



hash = 3536116

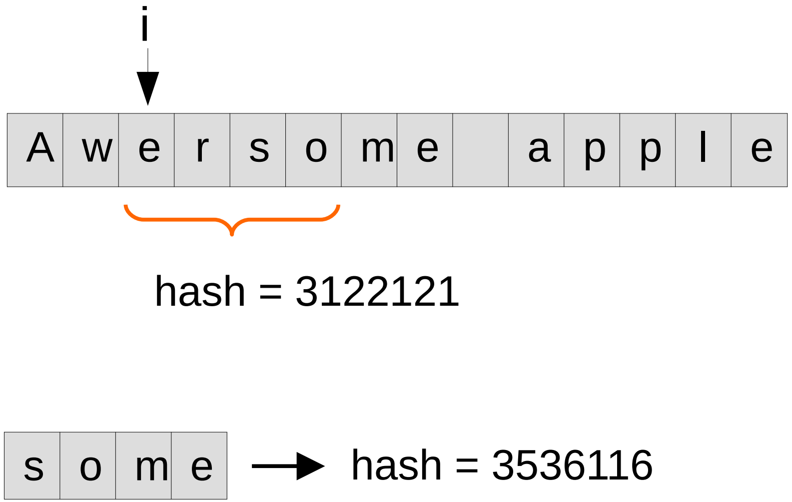
Вычисляем хеш-код части базовой строки (длиной равной длине искомой строки). При сравнении оказывается, что эти хеши не равны. Сдвигаем расположение части базовой строки на один символ.

Графическое пояснение алгоритма



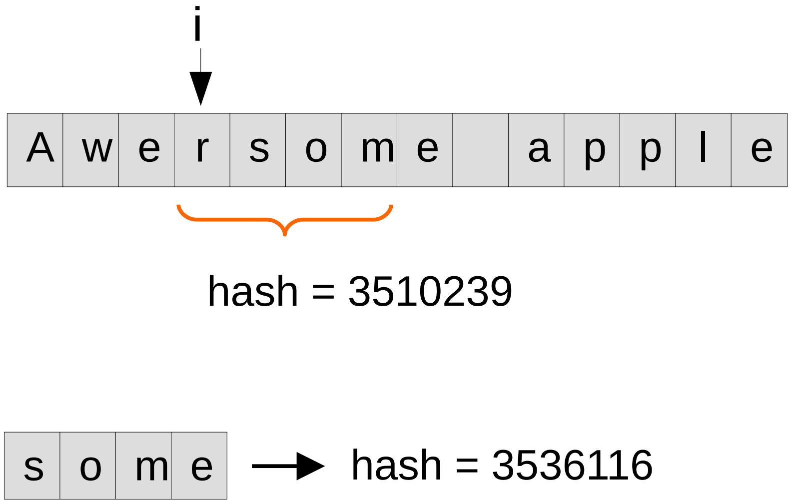
Вычисляем хеш-код части базовой строки (длиной равной длине искомой строки). При сравнении оказывается, что эти хеши не равны. Сдвигаем расположение части базовой строки на один символ.

Графическое пояснение алгоритма



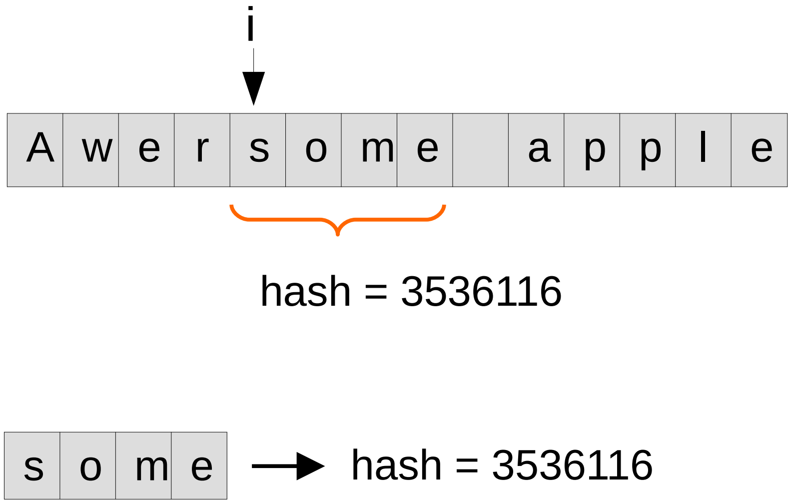
Вычисляем хеш-код части базовой строки (длиной равной длине искомой строки). При сравнении оказывается, что эти хеши не равны. Сдвигаем расположение части базовой строки на один символ.

Графическое пояснение алгоритма



Вычисляем хеш-код части базовой строки (длиной равной длине искомой строки). При сравнении оказывается, что эти хеши не равны. Сдвигаем расположение части базовой строки на один символ.

Графическое пояснение алгоритма



Вычисляем хеш-код части базовой строки (длиной равной длине искомой строки). При сравнении оказывается, что эти хеши равны. Заканчиваем алгоритм, возвращаем индекс начала части строки как результат.

Используемый вид хеш-функции

Для использования этого алгоритма используются хеш-функции определенного класса. Они относятся к скользящим (кольцевым) хеш-функциям.

Скользящая (кольцевая) хеш-функция — хеш-функция вычисляемая на основе части диапазона входных данных. При этом при сдвиге диапазона хеш вычисляется на основе ранее ранее вычисленного хеша. Вычисление в этом случае выполняется намного быстрее.

Для этого алгоритма была предложена полиномиальная хеш-функция.

$$h(s[1..m])=\left(\sum_{i=1}^{m} s[i]\cdot x^{m-i}\right) mod\, q$$

$s[i]$ - целочисленное представление символа

x - положительное целое число

q - положительное целое число

Для выбора q и x есть ряд рекомендаций. Число q — должно быть простым. Для ускорения вычисления рекомендуют использовать числа Мерсенна. Так для 32 битных хешей рекомендуется выбрать q = 2^{31} - 1 = 2147483647, для 64 битных 2^{61} — 1. В качестве x можно выбрать любое число в диапазоне 0..q - 1.

Графическое пояснение алгоритма полиномиального хеша

a	p	p	l	e

$$q=2147483647$$

$$x=31$$

$$h(app)=\left(\sum_{i=1}^{3} s[i]\cdot 31^{3-i}\right) mod\, 2147483647=\left(97\cdot 31^{2}+112\cdot 31^{1}+112\cdot 31^{0}\right) mod\, 2147483647=96801$$

a	p	p	l	e

$$h(ppl)=\left(\sum_{i=1}^{3} s[i]\cdot 31^{3-i}\right) mod\, 2147483647=\left(112\cdot 31^{2}+112\cdot 31^{1}+108\cdot 31^{0}\right) mod\, 2147483647=111212$$

Вычисление нового хеша на основе предыдущего

-

$$h(app)=\left(\sum_{i=1}^{3} s[i]\cdot 31^{3-i}\right) mod\, 2147483647=(97\cdot 31^{2}+112\cdot 31^{1}+112\cdot 31^{0})\, mod\, 2147483647=96801$$

+

$$h(ppl)=\left(\sum_{i=1}^{3} s[i]\cdot 31^{3-i}\right) mod\, 2147483647=(112\cdot 31^{2}+112\cdot 31^{1}+108\cdot 31^{0})\, mod\, 2147483647=111212$$

Таким образом для вычисления нового хеша на основе старого требуется

1) Из старого хеша вычесть значение $s[1]\cdot x^{m-1}$
2) Полученный результат умножить на x
3) Прибавить значение $s[m+1]$
4) Вычислить остаток от деления на q

Вычисление нового хеша на основе предыдущего

В общем случае новый хеш код вычисляется через предыдущий следующим образом

$$h(s[i+1 \ldots i+m]) = \left(\left(h(s[i \ldots i+m-1]) - s[i] \cdot x^{m-1}\right) \cdot x + s[i+m]\right) mod\, q$$

Удобство полиномиального хеша

Полиномиальный хеш удобен также тем, что для его вычисления можно (и нужно) использовать схему Горнера. Тут числовые коды символов - коэффициенты полинома, причем они идут в порядке уменьшения степени.

a	p	p

$$h(app)=(97\cdot 31^2+112\cdot 31^1+112\cdot 31^0)\,mod\,2147483647=((97\cdot 31+112)\cdot 31+112)\,mod\,2147483647$$

Реализация алгоритма на Python

Вычисление полиномиального хеша по схеме Горнера

```
def gorner_scheme(text):
    result = ord(text[0])
    base = 31
    for i in range(len(text) - 1):
        result = result * base + ord(text[i + 1])
    return result

def calculate_hash(text):
    q = 2147483647
    return gorner_scheme(text) % q
```

Python

Реализация поиска подстроки по алгоритму Рабина - Карпа

```
def search_text(text, sub_text):
    base = 31
    q = 2147483647
    sub_hash = calculate_hash(sub_text)
    m = len(sub_text)
    current_hash = calculate_hash(text[0:m])
    i = 0
    while True:
        if sub_hash == current_hash:
            if sub_text == text[i:i+m]:
                return i
        if i + m >= len(text):
            break
        current_hash = ((current_hash - ord(text[i]) * base ** (m-1)) * base + ord(text[i + m])) % q
        i = i + 1
    return None
```

Поиск множества подстрок

Этот алгоритм очень удобен для поиска множества подстрок (одинаковой длины). Для каждой подстроки вычисляется хеш и на каждой итерации сравнивают этот хеш с хешом полученным на основе части основной строки. В таком случае этот алгоритм становится наиболее оптимальным для задач такого плана.

Реализация алгоритма на Java

Вычисление хеша по схеме Горнера

```java
public final static int BASE = 31;
public final static int q = 2147483647;

public static int gornerScheme(char[] sym, int start, int end) {
     int result = (int) (sym[start]);
     for (int i = start; i < end - 1; i++) {
          result = result * BASE + (int) sym[i + 1];
     }
     return result;
}

public static int calculateHash(char[] sym, int start, int end) {
     return gornerScheme(sym, start, end) % q;
}
public static int basePow(int n) {
     if (n == 0) {
          return 1;
     }
     int result = 1;
     for (int i = 0; i < n; i++) {
          result *= BASE;
     }
     return result;
}
```

Вспомогательный метод для проверки совпадения хешей

```java
public static int[] findSomeHash(int hash, int[] subHashs) {
      int n = 0;
      for (int i = 0; i < subHashs.length; i++) {
            if (subHashs[i] == hash) {
                  n += 1;
            }
      }
      int[] result = new int[n];
      int insertIndex = 0;
      for (int i = 0; i < subHashs.length; i++) {
            if (subHashs[i] == hash) {
                  result[insertIndex++] = i;
            }
      }
      return result;
}
```

Вспомогательный метод для проверки одинаковости длины искомых подстрок

```
public static boolean checkEqualsLength(String[] texts) {
      int l = texts[0].length();
      for (int i = 0; i < texts.length; i++) {
            if (texts[i].length() != l) {
                  return false;
            }
      }
      return true;
}
```

Метод поиска подстрок по алгоритму Рабина-Карпа

```
public static int[] substringSearch(String baseText, String... subStrings) {
        int[] result = new int[] { -1, -1 };
        if (!checkEqualsLength(subStrings)) {
                throw new IllegalArgumentException("substrings must be the same length");
        }
        int[] hashArray = new int[subStrings.length];
        for (int i = 0; i < hashArray.length; i++) {
                hashArray[i] = calculateHash(subStrings[i].toCharArray(), 0, subStrings[i].length());
        }
        char[] sym = baseText.toCharArray();
        int start = 0;
        int m = subStrings[0].length();
        int end = start + m;
        int textPartHash = calculateHash(sym, start, end);
        int mult = basePow(m - 1);
        for (;;) {
                int[] someHas = findSomeHash(textPartHash, hashArray);
                if (someHas.length > 0) {
                        String text = new String(sym, start, m);
                        for (int i = 0; i < someHas.length; i++) {
                                if (text.equals(subStrings[someHas[i]])) {
                                        result[0] = start;
                                        result[1] = someHas[i];
                                        return result;
                                }
                        }
                }
                start += 1;
                end += 1;
                if (end > sym.length) {
                        break;
                }
                textPartHash = ((textPartHash - mult * (int) sym[start - 1]) * BASE + sym[end - 1]) % q;
        }
        return result;
}
```

Реализация алгоритма на Fortran

Fortran

Константы и функции для вычисления хеш-кода части строки

```
integer, parameter::base = 31
integer, parameter::q = 2147483647

function gorner_scheme(text,s,e)
    character(len = *), intent(in)::text
    integer, intent(in)::s, e
    integer::gorner_scheme
    integer::i
    gorner_scheme = iachar(text(s:s))
    do i = s, e - 1
        gorner_scheme = gorner_scheme * base + iachar(text(i+1:i+1))
    end do
end function gorner_scheme

function calculate_hash(text,s,e)
    character(len = *), intent(in)::text
    integer, intent(in)::s, e
    integer::calculate_hash
    calculate_hash = MOD(gorner_scheme(text,s,e),q)
end function calculate_hash
```

Поиск подстроки по алгоритму Рабина-Карпа

```
function sub_text_search(text, sub_text)
    character(len=*), intent(in)::text
    character(len=*), intent(in)::sub_text
    integer::sub_text_search
    integer::s,e,m, pow_coeff
    integer::sub_text_hash
    integer::text_part_hash
    sub_text_search = -1
    m = len_trim(sub_text)
    sub_text_hash = calculate_hash(sub_text,1,m)
    text_part_hash = calculate_hash(text,1,m)
    pow_coeff = base ** (m - 1)
    s = 1
    e = s + m - 1
    do
        if (sub_text_hash == text_part_hash) then
            if(trim(sub_text)== text(s:e)) then
                sub_text_search = s
                exit
            end if
        end if
        s = s + 1
        e = e + 1
        if (e > len_trim(text)) then
            exit
        end if
        text_part_hash = (text_part_hash - iachar(text(s-1:s-1)) * pow_coeff) * base + iachar(text(e:e))
        text_part_hash = MOD (text_part_hash,q)
    end do
end function sub_text_search
```

Список литературы

1) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2