Алгоритмы на строках

Алексей Владыкин

СПбГУ ИТМО

16 ноября 2009

- Основная задача: поиск вхождений образца в текст.

  Варианты:
- поиск одного образца или многих сразу;
- поиск по точному образцу или по регулярному выражению;
- для точного образца: точный или неточный поиск.

- Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И. В. Романовского. — СПб.: Невский Диалект; БХВ-Петербург, 2003. — 654 с.: ил.

Общие соглашения

- Pattern — строка длины m, её ищем.
- Text — строка длины n, в ней ищем.
- Для всех найденных вхождений образца вызываем переданную в алгоритм функцию (паттерн Callback):

```
typedef int (*pattern_found_callback)(
        char* text, char* pattern, int start_pos);
```

- Пример реализации:

```
int print_callback(char* text, char* pattern, int start_pos) {
    printf("== FOUND ==\n");
    printf("Text:    %s\n", text);
    printf("Pattern: %*s%s\n\n", start_pos, "", pattern);
    return 1;
}
```

Наивный поиск подстроки

- Время работы — $O(mn)$.

```
void naive_search(char* text, char* pattern, pattern_found_callback c
    char *s, *t, *p;
    for (s = text; *s; ++s) {
        for (t = s, p = pattern; *p && *t == *p; ++t, ++p);
        if (!*p && !(*callback)(text, pattern, s - text)) {
            return;
        }
    }
}
```

Алгоритм Бойера — Мура

- Сравнение с конца образца.
- Сильное/слабое правило плохого символа.
- Сильное/слабое правило хорошего суффикса.
- Каждое правило говорит, на сколько позиций сдвинуть образец. Выбираем из двух сдвигов наибольший.

- Время на препроцессинг — $O(m)$
- Одно слабое правило плохого символа дает время поиска $O(n/m)$ в лучшем случае, но в худшем те же $O(mn)$.
- Два правила вместе в сильном виде дают время поиска $O(n)$ в худшем случае.

Правило плохого символа

- После несовпадения сдвигаем образец так, чтобы совместить несовпавший символ текста с таким же символом образца. (Если такого символа в образце нет, сдвигаем на всю длину образца.)

```
text:    a b r a c a d a b r a
pattern: r a b a c
           r a b a c
```

- В слабом варианте для каждого символа алфавита запоминаем его крайнее правое вхождение в образец. В сильном варианте для каждого символа алфавита запоминаем все его вхождения в образец. В следующем примере сработает только сильный вариант:

```
text:    a b r a c a d a b r a
pattern:     r b c a d
```

Правило хорошего суффикса

- После несовпадения сдвигаем образец так, чтобы совместить совпавшую подстроку s текста со следующим вхождением этой подстроки в образец.

text:	a	b	r	a	c	a	d	a	b	r	a
pattern:		b	a	d	b	a	d				
					b	a	d	b	a	d	

- Если такой подстроки в образце больше нет, выбираем наибольший префикс образца, являющийся суффиксом совпавшей подстроки s.

text:	a	b	r	a	c	a	d	a	b	r	a
pattern:			a	d	c	a	d				
						a	d	c	a	d	

- В сильном варианте дополнительно требуется, чтобы после сдвига в образце перед s встал другой символ. В первом примере образец «badbad» был бы сдвинут на всю длину.

Алгоритм Бойера — Мура

```
void boyer_moore_search(char* text, char* pattern,
                        pattern_found_callback callback) {
    int i, j, char_pos[UCHAR_MAX + 1];
    memset(char_pos, -1, sizeof(char_pos));
    for (i = 0; pattern[i]; ++i) {
        char_pos[(unsigned char) pattern[i]] = i;
    }
    int n = strlen(text), m = strlen(pattern);
    for (i = 0; i + m <= n; ) {
        for (j = m - 1; 0 <= j; --j) {
            if (text[i + j] != pattern[j]) {
                i += max(1, j - char_pos[text[i + j]]);
                break;
            }
        }
        if (j < 0) {
            if (!(*callback)(text, pattern, i)) { return; }
            ++i;
        }
    }
}
```

Алгоритм Кнута — Морриса — Пратта

- Сравнение с начала образца.
- Передвигаем образец по правилу, похожему на правило хорошего суффикса из алгоритма Бойера — Мура: пре несовпадении символа ищем наибольший суффикс совпавшей части, являющийся префиксом образца.

- Время на препроцессинг + поиск: $O(m + n)$ в худшем случае

Примеры

- В следующем примере максимальный суффикс совпавшей части, являющийся её же префиксом — «ab».

text:	a	b	r	a	c	a	d	a	b	r	a
pattern:	a	b	r	a	c	a	d	a	b	b	
								a	b	r	...

- В следующем примере максимальный подходящий префикс — «a», однако следующий за ним символ «b» равен несовпавшему «b» в конце образца. Поэтому образец можно сдвинуть ещё дальше.

text:	a	b	r	a	c	a	d	a	b	r	a
pattern:	a	b	r	a	c	a	b				
							a	b	r	a	...

Алгоритм Кнута — Морриса — Пратта

```
void knuth_morris_pratt_search(char* text, char* pattern,
                               pattern_found_callback callback) {
    int m = strlen(pattern), n = strlen(text), jump[m + 1];
    int i = 0, j = jump[0] = -1;
    while (i < m) {
        while (0 <= j && pattern[i] != pattern[j]) {
            j = jump[j];
        }
        i++;
        j++;
        if (pattern[i] == pattern[j]) {
            jump[i] = jump[j];
        } else {
            jump[i] = j;
        }
    }

    // see next slide
```

Алгоритм Кнута—Морриса—Пратта

```
    // see previous slide

    i = j = 0;
    while (i < n) {
        while (0 <= j && pattern[j] != text[i]) {
            j = jump[j];
        }
        i++;
        j++;
        if (m <= j) {
            if (!(*callback)(text, pattern, i - j)) {
                return;
            }
            j = jump[j];
        }
    }
}
```

Алгоритм Рабина — Карпа

- Использование хеш-функции и сравнение чисел вместо строк.
- Быстрый пересчет значения хеш-функции для $text[i+1..i+m]$ из её значения для $text[i..i+m-1]$.

```
unsigned long hash_calc(char* s, int n);
unsigned long hash_recalc(char* s, int n, unsigned long h);
```

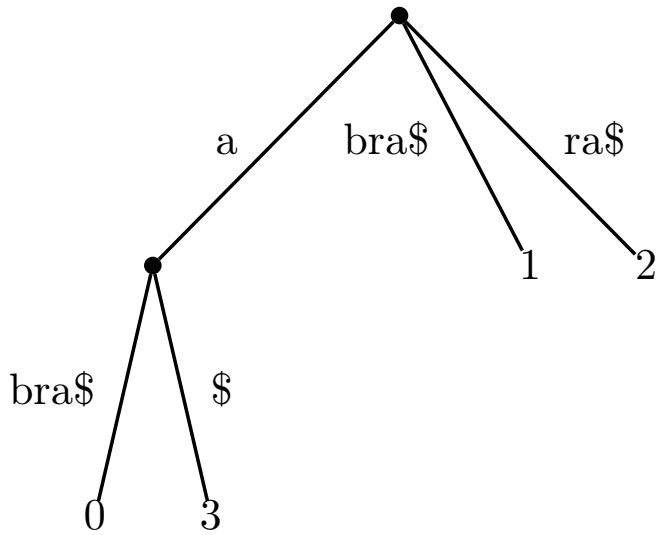
- Примеры хеш-функций:
  - сумма кодов символов;
  - произведение кодов символов;
  - интерпретация строки как числа в некоторой системе счисления.
- Время работы в среднем — $O(m+n)$.

Алгоритм Рабина—Карпа

```
void rabin_karp_search(char* text, char* pattern,
                       pattern_found_callback callback) {
    int i, j, m = strlen(pattern), n = strlen(text);
    unsigned long pattern_hash = hash_calc(pattern, m);
    unsigned long text_hash;
    for (i = 0; i + m <= n; ++i) {
        if (i == 0) {
            text_hash = hash_calc(text, m);
        } else {
            text_hash = hash_recalc(text + i - 1, m, text_hash);
        }
        if (text_hash == pattern_hash) {
            if (memcmp(text + i, pattern, m) == 0) {
                if (!(*callback)(text, pattern, i)) { return; }
            }
        }
    }
}
```

Суффиксные деревья

- Суффиксное дерево — структура данных, позволяющая эффективно решать многие строковые задачи.
- Суффиксное дерево строки s содержит все её суффиксы.
- Строка должна заканчиваться символом, больше нигде в ней не встречающимся. Его специально приписывают и обозначают \$.



Построение суффиксного дерева

- Наивный алгоритм — $O(n^2)$.

Линейные алгоритмы:

- алгоритм Вайнера (1973 г.);
- алгоритм Мак-Крейта (1976 г.);
- алгоритм Укконена (1995 г.).

- Требования к памяти — $O(n)$.

Применение суффиксных деревьев

- Поиск вхождений одного или нескольких образцов.
- Поиск наибольшей общей подстроки двух или более строк.
- Поиск наименьшего общего предшественника.
- ...