

ЛИГА ЛЕКТОРОВ

Борис Алексеевич
Золотов

Тема:

Задача поиска
подстроки в строке





Задача поиска подстроки в строке

Дан **текст** T и **образец** S ; проверить, содержится ли S в T — как «кад» содержитя в «абракадабра».



Задача поиска подстроки в строке

Дан **текст** T и **образец** S ; проверить, содержится ли S в T — как «кад» содержится в «абракадабра».

Вы скажете: я просто нажимаю **Ctrl** + **F**, в чём проблема?



Задача поиска подстроки в строке

Дан **текст** T и **образец** S ; проверить, содержится ли S в T — как «кад» содержитя в «абракадабра».

Вы скажете: я просто нажимаю **Ctrl** + **F**, в чём проблема?

Вы-то понятно, но что в этот момент думает компьютер?



Задача поиска подстроки в строке

Дан **текст** T и **образец** S ; проверить, содержится ли S в T — как «кад» содержится в «абракадабра».

Вы скажете: я просто нажимаю **Ctrl** + **F**, в чём проблема?

Вы-то понятно, но что в этот момент думает компьютер?

Будем считать, что текст **очень длинный** и, возможно, дописывается прямо сейчас.



Наивный алгоритм

Начиная с каждой позиции в T , сравнивать символы в T и S ;
если удаётся дойти до конца S — мы нашли вхождение.

$T = \begin{matrix} 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 \end{matrix}$

$S = \begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

$\begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

$\begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

$\begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

$\begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

$\begin{matrix} 1 & 2 & 1 & 2 & 3 \end{matrix}$

Проблема наивного алгоритма



В худшем случае придётся сделать примерно $|S| \cdot |T|$ индивидуальных сравнений символов — представьте, просмотреть длинный текст много раз?



Проблема наивного алгоритма

В худшем случае придётся сделать примерно $|S| \cdot |T|$ индивидуальных сравнений символов — представьте, просмотреть длинный текст много раз?

Можно ли быстрее? Например, чтобы количество операций составляло примерно $|S| + |T|$.

Наука об алгоритмах



В этом заключается суть *науки об алгоритмах* —
с помощью умных наблюдений и правильной
последовательности выполнения добиваться
оптимального времени работы.

Классические задачи: сортировка списка чисел, поиск
в упорядоченном наборе, поиск путей (в т. ч. на карте),...

Хэш



Проблема: чтобы выяснить, что S не равно куску T , нужно много сравнений.

Что, если сопоставить строке число, которое будет характеризовать строку в целом и почти всегда различаться для различных строк?
Такое число называется **хэшом**.



Алгоритм Рабина – Карпа

Значение хэша: многочлен, коэффициенты которого — символы строки; взять остаток от деления на простое число.

Пусть мы ищем подстроку 131:

$$1 \cdot 3^2 + 3 \cdot 3^1 + 1 \cdot 3^0 = 5 \pmod{7}.$$

Последовательно считаем хэши кусков T нужной длины, если хэш совпал с хэшом S — сравниваем посимвольно.

Рабин – Карп: пересчёт хэша



3 2 2 1 3 1 1

$$3 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 0 \pmod{7}$$

$$2 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 = 4 \pmod{7}$$

$$2 \cdot 3^2 + 1 \cdot 3^1 + 3 \cdot 3^0 = 3 \pmod{7}$$

$$1 \cdot 3^2 + 3 \cdot 3^1 + 1 \cdot 3^0 = 5 \pmod{7}$$

$$3 \cdot 3^2 + 1 \cdot 3^1 + 1 \cdot 3^0 = 3 \pmod{7}$$

Рабин – Карп: проблема совпадения хэшей



2

2

2

2

2

2

2

$$2 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 5 \pmod{7}$$

$$2 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 5 \pmod{7}$$

$$2 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 5 \pmod{7}$$

$$2 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 5 \pmod{7}$$

$$2 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 = 5 \pmod{7}$$

Алгоритм Кнута – Морриса – Пратта



Попробуем снова посимвольно сравнивать S и T ,
но придумаем, как избегать ненужных сравнений,
если строка S правильно предобработана.

Алгоритм Кнута – Морриса – Пратта



Попробуем снова посимвольно сравнивать S и T ,
но придумаем, как избегать ненужных сравнений,
если строка S правильно предобработана.

С каких позиций в T может начинаться S ?

Кнут – Моррис – Пратт: пример



1	2	1	3	1	2	1	2	1	3									
					*													
1	2	1	3	1	3	1	2	2	1	3	1	2	1	3	1	2	1	3



Кнут – Моррис – Пратт: пример

1 2 1 3 1 2 1 2 1 3
|| || || || || *
1 2 1 3 1 3 1 2 2 1 2 1 3 1 2 1 3 1 2 1 2 1 3



Кнут – Моррис – Пратт: пример

1	2	1	3	1	2	1	2	1	3											
*																				
1	2	1	3	1	3	1	2	2	1	2	1	3	1	2	1	3	1	2	1	3

Кнут – Моррис – Пратт: пример



1	2	1	3	1	3	1	2	2	1	2	1	3
									*			
1	2	1	3	1	3	1	2	2	1	3	1	2

Кнут – Моррис – Пратт: пример



1	2	1	3	1	3	1	2	2	1	2	1	3	1	2	1	3	2	1	3	
																*				
1	2	1	3	1	3	1	2	2	1	2	1	3	1	2	1	3	1	2	1	3



Кнут – Моррис – Пратт: пример

	1	2	1	3	1	3	1	2	2	1	2	1	3	1	2	1	3	1	2	1	2	1	3
1	2	1	3	1	3	1	2	2	1	2	1	3	1	2	1	3	1	2	1	2	1	3	



Префикс-функция

Какая информация нам была нужна? Про каждую позицию i строки S — наибольшая длина $\ell < i$ начального куска S , который совпадает с ℓ символами перед позицией i .

Это называется префикс-функцией, обозначается $\pi(i)$.

1 2 1 3 1 2 1 2 1 3



Префикс-функция

Какая информация нам была нужна? Про каждую позицию i строки S — наибольшая длина $\ell < i$ начального куска S , который совпадает с ℓ символами перед позицией i .

Это называется префикс-функцией, обозначается $\pi(i)$.

1 2 1 3 0 1 2 1 2 1 3
0 0 1 0 1 2 2 1 3



Префикс-функция

Какая информация нам была нужна? Про каждую позицию i строки S — наибольшая длина $\ell < i$ начального куска S , который совпадает с ℓ символами перед позицией i .

Это называется префикс-функцией, обозначается $\pi(i)$.

1	2	1	3	1	2	1	2	1	3
0	0	1	0	1	2	2	3		



Префикс-функция

Какая информация нам была нужна? Про каждую позицию i строки S — наибольшая длина $\ell < i$ начального куска S , который совпадает с ℓ символами перед позицией i .

Это называется префикс-функцией, обозначается $\pi(i)$.





Вычисление префикс-функции

$$k = \pi(i); \quad \begin{cases} \pi(i+1) = k+1, & s(i+1) = s(k+1), \\ \pi(i+1) = 0, & k = 0 \text{ и } s(i+1) \neq s(1), \\ k := \pi(k), & \text{иначе.} \end{cases}$$





Вычисление префикс-функции

$$k = \pi(i); \quad \begin{cases} \pi(i+1) = k+1, & s(i+1) = s(k+1), \\ \pi(i+1) = 0, & k = 0 \text{ и } s(i+1) \neq s(1), \\ k := \pi(k), & \text{иначе.} \end{cases}$$





Вычисление префикс-функции

$$k = \pi(i); \quad \begin{cases} \pi(i+1) = k+1, & s(i+1) = s(k+1), \\ \pi(i+1) = 0, & k = 0 \text{ и } s(i+1) \neq s(1), \\ k := \pi(k), & \text{иначе.} \end{cases}$$





Вычисление префикс-функции

$$k = \pi(i); \quad \begin{cases} \pi(i+1) = k+1, & s(i+1) = s(k+1), \\ \pi(i+1) = 0, & k = 0 \text{ и } s(i+1) \neq s(1), \\ k := \pi(k), & \text{иначе.} \end{cases}$$



Конечные автоматы



Мы привели алгоритм распознавания строк, содержащих S :

$* S *$

Бывает полезно распознавать электронные адреса:

[нет @] @ [нет @] . [com|ru| ...]

Конечные автоматы



Номера банковских карт: 16 цифр, но

$$\sum(2 \cdot a_{2i}) \bmod 9 + a_{2i+1} \vdash 10.$$

Корректные даты, пароли, номера телефонов,
пункты задач олимпиад...

Вычислительная модель, эффективно распознающая
такие строки — **конечный автомат**.



- ▶ Для решения задачи поиска подстроки есть наивный алгоритм, $|T| \cdot |S|$ сравнений символов;
- ▶ Алгоритм Рабина — Карпа, $|T| + |S|$ арифметических операций и сравнений *в среднем*, но иногда хэши неравных подстрок совпадают;
- ▶ Алгоритм Кнута — Морриса — Пратта, $|T| + |S|$ сравнений символов.

Спасибо за внимание!