

Строки

Никита Гаевой (102)
Иван Казменко (101, 103)
Владислав Макаров (104)
Семён Петров (106)
Лиана Хазалия (105)

Санкт-Петербургский Государственный Университет

Четверг, 16 декабря 2021 года

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфилда: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфилда: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Определения

- *Подстрока* — это последовательность соседних символов строки.

Определения

- *Подстрока* — это последовательность соседних символов строки.
- Например, у строки «abcd» есть такие подстроки:
 - «» (пустая строка),
 - «abcd» (вся строка),
 - «abc»,
 - «ca».

Определения

- *Подстрока* — это последовательность соседних символов строки.
- Например, у строки «abcd» есть такие подстроки:
 - «» (пустая строка),
 - «abcd» (вся строка),
 - «abc»,
 - «ca».
- А следующие строки её подстроками не являются:
 - «e» (нет такой буквы),
 - «aa» (не соседние),
 - «bcd».

Определения

- *Подстрока* — это последовательность соседних символов строки.
- Например, у строки «abcd» есть такие подстроки:
 - «» (пустая строка),
 - «abcd» (вся строка),
 - «abc»,
 - «ca».
- А следующие строки её подстроками не являются:
 - «e» (нет такой буквы),
 - «aa» (не соседние),
 - «bcd».
- *Префикс* — это подстрока, начинающаяся в начале строки.
- *Суффикс* — подстрока, заканчивающаяся в конце строки.
- *Собственный префикс* или *суффикс* — это префикс или суффикс, не совпадающий с самой строкой.

Задача

Задача:

- Даны строка S и текст T .
- Содержится ли S в T как подстрока?

Задача

Задача:

- Даны строка S и текст T .
- Содержится ли S в T как подстрока?
- Варианты:
 - установить факт,
 - найти первое или любое вхождение,
 - найти все вхождения.

Задача

Задача:

- Даны строка S и текст T .
- Содержится ли S в T как подстрока?
- Варианты:
 - установить факт,
 - найти первое или любое вхождение,
 - найти все вхождения.
- Примеры:
 - $S = \text{«aba»}$, $T = \text{«ababa»}$ (2 вхождения).
 - $S = \text{«aba»}$, $T = \text{«aabba»}$ (0 вхождений).
 - $S = \text{«aba»}$, $T = \text{«abaababaababa»}$ (5 вхождений).

Наивное решение

Пусть $|S| = m$ и $|T| = n$. Наивное решение за $O(mn)$:

- Сравним $S_1S_2 \dots S_m$ и $T_1T_2 \dots T_m$.
- Сравним $S_1S_2 \dots S_m$ и $T_2T_3 \dots T_{m+1}$.
- Сравним $S_1S_2 \dots S_m$ и $T_3T_4 \dots T_{m+2}$.
- ...
- Сравним $S_1S_2 \dots S_m$ и $T_{n-m+1}T_{n-m+2} \dots T_n$.

Наивное решение

Пусть $|S| = m$ и $|T| = n$. Наивное решение за $O(mn)$:

- Сравним $S_1S_2 \dots S_m$ и $T_1T_2 \dots T_m$.
- Сравним $S_1S_2 \dots S_m$ и $T_2T_3 \dots T_{m+1}$.
- Сравним $S_1S_2 \dots S_m$ и $T_3T_4 \dots T_{m+2}$.
- ...
- Сравним $S_1S_2 \dots S_m$ и $T_{n-m+1}T_{n-m+2} \dots T_n$.

Пример 1:

- $T = \text{«ababa»}$
- $S = \text{«aba»}$
- $T_1T_2T_3 = \text{«aba»}$, нашли
- $T_2T_3T_4 = \text{«bab»}$
- $T_3T_4T_5 = \text{«aba»}$, нашли

Наивное решение

Пусть $|S| = m$ и $|T| = n$. Наивное решение за $O(mn)$:

- Сравним $S_1S_2 \dots S_m$ и $T_1T_2 \dots T_m$.
- Сравним $S_1S_2 \dots S_m$ и $T_2T_3 \dots T_{m+1}$.
- Сравним $S_1S_2 \dots S_m$ и $T_3T_4 \dots T_{m+2}$.
- ...
- Сравним $S_1S_2 \dots S_m$ и $T_{n-m+1}T_{n-m+2} \dots T_n$.

Пример 2:

- $T = \text{«}aa \dots abaa \dots a\text{»}$
- $S = \text{«}aa \dots a\text{»}$
- $T_1T_2 \dots T_m = \text{«}aa \dots ab\text{»}$
- $T_2T_3 \dots T_{m+1} = \text{«}aa \dots ba\text{»}$
- ...
- $T_mT_{m+1} \dots T_n = \text{«}baa \dots a\text{»}$

Наивный алгоритм: код

```
1  vector <int> findNaive (string s, string t) {
2      vector <int> res;
3      int m = s.size (), n = t.size ();
4      for (int i = 0; i + m <= n; i++) {
5          bool found = true;
6          for (int j = 0; j < m; j++)
7              if (s[j] != t[i + j]) {
8                  found = false;
9                  break;
10             }
11         if (found)
12             res.push_back (i + 1);
13     }
14     return res;
15 }
```

ВВОД:

aba
ababaaba

ВЫВОД:

1 3 6

Наивный алгоритм: код

```
1  vector <int> findNaive (string s, string t) {
2      vector <int> res;
3      int m = s.size (), n = t.size ();
4      for (int i = 0; i + m <= n; i++) {
5          bool found = true;
6          for (int j = 0; j < m; j++)
7              if (s[j] != t[i + j]) {
8                  found = false;
9                  break;
10             }
11         if (found)
12             res.push_back (i + 1);
13     }
14     return res;
15 }
```

ВВОД:

abb

ababaaba

ВЫВОД:

none

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфилда: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Идея

Идея:

- Сопоставим каждой возможной строке случайное число.
- Вместо того, чтобы сравнивать строки, сравним числа.

Идея

Идея:

- Сопоставим каждой возможной строке случайное число.
- Вместо того, чтобы сравнивать строки, сравним числа.
- Если числа не равны, строки точно не равны.

Идея

Идея:

- Сопоставим каждой возможной строке случайное число.
- Вместо того, чтобы сравнивать строки, сравним числа.
- Если числа не равны, строки точно не равны.
- Если числа равны, ..

Идея

Идея:

- Сопоставим каждой возможной строке случайное число.
- Вместо того, чтобы сравнивать строки, сравним числа.
- Если числа не равны, строки точно не равны.
- Если числа равны,..

Числа сравниваются быстрее, чем строки, только если они небольшие.

- Пусть наши случайные числа — целые из $[0, q)$.

Идея

Идея:

- Сопоставим каждой возможной строке случайное число.
- Вместо того, чтобы сравнивать строки, сравним числа.
- Если числа не равны, строки точно не равны.
- Если числа равны, ..

Числа сравниваются быстрее, чем строки, только если они небольшие.

- Пусть наши случайные числа — целые из $[0, q)$.
- Если числа равны, строки не равны с вероятностью $1/q$.

Полиномиальный хеш

Как выбирать случайные числа?

- Настоящие случайные числа сложно использовать.

Полиномиальный хеш

Как выбирать случайные числа?

- Настоящие случайные числа сложно использовать.
- Построим псевдослучайные: полиномиальный хеш.
- $h(S_1 S_2 \dots S_k) = (((S_1 \cdot p + S_2) \cdot p + \dots) \cdot p + S_k) \bmod q$

Полиномиальный хеш

Как выбирать случайные числа?

- Настоящие случайные числа сложно использовать.
- Построим псевдослучайные: полиномиальный хеш.
- $h(S_1 S_2 \dots S_k) = (((S_1 \cdot p + S_2) \cdot p + \dots) \cdot p + S_k) \bmod q$
- $h(S_1 S_2 \dots S_k) = (S_1 \cdot p^{k-1} + S_2 \cdot p^{k-2} + \dots + S_k \cdot p^0) \bmod q$

Хеш подстроки

Как быстро получить хеши всех подстрок T длины $|S|$?

Хеш подстрок

Как быстро получить хеши всех подстрок T длины $|S|$?

- $$h(T_1 T_2 \dots T_m) = (T_1 \cdot p^{m-1} + T_2 \cdot p^{m-2} + \dots + T_m \cdot p^0) \bmod q$$

Хеш подстроки

Как быстро получить хеши всех подстрок T длины $|S|$?

- $h(T_1 T_2 \dots T_m) = (T_1 \cdot p^{m-1} + T_2 \cdot p^{m-2} + \dots + T_m \cdot p^0) \bmod q$
- $h(T_2 \dots T_m T_{m+1}) = (T_2 \cdot p^{m-1} + \dots + T_m \cdot p^1 + T_{m+1} \cdot p^0) \bmod q$

Хеш подстрок

Как быстро получить хеши всех подстрок T длины $|S|$?

- $h(T_1 T_2 \dots T_m) = (T_1 \cdot p^{m-1} + T_2 \cdot p^{m-2} + \dots + T_m \cdot p^0) \bmod q$
- $h(T_2 \dots T_m T_{m+1}) = (T_2 \cdot p^{m-1} + \dots + T_m \cdot p^1 + T_{m+1} \cdot p^0) \bmod q$
- $h(T_2 \dots T_{m+1}) = (h(T_1 \dots T_m) \cdot p - T_1 \cdot p^m + T_{m+1}) \bmod q$

Хеш подстрок

Как быстро получить хеши всех подстрок T длины $|S|$?

- $h(T_1 T_2 \dots T_m) = (T_1 \cdot p^{m-1} + T_2 \cdot p^{m-2} + \dots + T_m \cdot p^0) \bmod q$
- $h(T_2 \dots T_m T_{m+1}) = (T_2 \cdot p^{m-1} + \dots + T_m \cdot p^1 + T_{m+1} \cdot p^0) \bmod q$
- $h(T_{2\dots m+1}) = (h(T_{1\dots m}) \cdot p - T_1 \cdot p^m + T_{m+1}) \bmod q$
- В общем случае:
- $h(T_{k+1\dots k+m}) = (h(T_{k\dots k+m-1}) \cdot p - T_k \cdot p^m + T_{k+m}) \bmod q$
- Число $p^m \bmod q$ можно найти один раз заранее.

Алгоритм Рабина–Карпа: код

```
1  int const p = 821, q = 999999937;
2
3  vector <int> findRabinKarp (string s, string t) {
4      vector <int> res;
5      int m = s.size (), n = t.size ();
6      int hs = 0, ht = 0, pm = 1;
7      for (int j = 0; j < m; j++) {
8          hs = (hs * 1LL * p + s[j]) % q;
9          pm = (pm * 1LL * p) % q;
10     }
11     for (int i = 0; i < n; i++) {
12         ht = (ht * 1LL * p + t[i]) % q;
13         if (i >= m) {
14             ht = (ht - t[i - m] * 1LL * pm) % q;
15             if (ht < 0) ht += q;
16         }
17         if (ht == hs)
18             res.push_back (i - m + 2);
19     }
20     return res;
21 }
```

ВВОД:

aba
ababaaba

ВЫВОД:

1 3 6

Алгоритм Рабина–Карпа: код

```
1  int const p = 821, q = 999999937;
2
3  vector <int> findRabinKarp (string s, string t) {
4      vector <int> res;
5      int m = s.size (), n = t.size ();
6      int hs = 0, ht = 0, pm = 1;
7      for (int j = 0; j < m; j++) {
8          hs = (hs * 1LL * p + s[j]) % q;
9          pm = (pm * 1LL * p) % q;
10     }
11     for (int i = 0; i < n; i++) {
12         ht = (ht * 1LL * p + t[i]) % q;
13         if (i >= m) {
14             ht = (ht - t[i - m] * 1LL * pm) % q;
15             if (ht < 0) ht += q;
16         }
17         if (ht == hs)
18             res.push_back (i - m + 2);
19     }
20     return res;
21 }
```

ВВОД:

abb

ababaaba

ВЫВОД:

none

Анализ

- Общее время работы:
- Вероятность ошибки:

Анализ

- Общее время работы: $O(m + n)$.
- Вероятность ошибки:

Анализ

- Общее время работы: $O(m + n)$.
- Вероятность ошибки: $1 - (1 - 1/q)^{n-m+1}$ в предположении, что полученные числа по-настоящему случайные.
- Математическое ожидание количества ошибок равно $(n - m + 1)/q$, можно использовать это (или даже n/q) как грубую оценку сверху для вероятности хотя бы одной ошибки.

Выбор констант

Как выбрать p и q ?

Выбор констант

Как выбрать p и q ?

- q должны быть как можно больше...
- ...но чтобы вычисления оставались быстрыми.
- p должно быть больше размера алфавита.
- p и q должны быть взаимно просты.
- q не должно быть степенью двойки.
- Можно выбирать q случайно во время выполнения программы.

Выбор констант

Как выбрать p и q ?

- q должны быть как можно больше...
- ...но чтобы вычисления оставались быстрыми.
- p должно быть больше размера алфавита.
- p и q должны быть взаимно просты.
- q не должно быть степенью двойки.
- Можно выбирать q случайно во время выполнения программы.
- Подойдут два больших простых числа.
- Если вероятность ошибки всё ещё велика, можно считать несколько разных хешей с разными константами p и q .

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфилда: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции i вычислим *префикс-функцию*: длину наибольшего собственного суффикса $S_1S_2 \dots S_i$, равного префиксу.

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции i вычислим *префикс-функцию*: длину наибольшего собственного суффикса $S_1S_2 \dots S_i$, равного префиксу.
- Пример 1:

S	=	а	а	b	а	а	b	а	а	с
p	=	0	1	0	1	2	3	4	5	0

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции i вычислим *префикс-функцию*: длину наибольшего собственного суффикса $S_1S_2 \dots S_i$, равного префиксу.
- Пример 2:

S	=	а	б	а	с	а	б	а
p	=	0	0	1	0	1	2	3

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ префикс-функция равна нулю.

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ префикс-функция равна нулю.
- При $i > 0$ мы знаем $k = p(i - 1)$.
- Это значит, что на предыдущей позиции $S_0S_1 \dots S_{k-1} = S_{i-k}S_{i-k+1} \dots S_{i-1}$ — самый длинный суффикс, равный префиксу.

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ префикс-функция равна нулю.
- При $i > 0$ мы знаем $k = p(i - 1)$.
- Это значит, что на предыдущей позиции $S_0 S_1 \dots S_{k-1} = S_{i-k} S_{i-k+1} \dots S_{i-1}$ — самый длинный суффикс, равный префиксу.
- Если $S_k = S_i$, то $p(i) = k + 1$.
- Если $k = 0$, просто сравним S_0 и S_i .

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ префикс-функция равна нулю.
- При $i > 0$ мы знаем $k = p(i - 1)$.
- Это значит, что на предыдущей позиции $S_0 S_1 \dots S_{k-1} = S_{i-k} S_{i-k+1} \dots S_{i-1}$ — самый длинный суффикс, равный префиксу.
- Если $S_k = S_i$, то $p(i) = k + 1$.
- Если $k = 0$, просто сравним S_0 и S_i .
- Если же $S_k \neq S_i$, нужно рассмотреть следующий по длине суффикс в позиции $i - 1$, равный префиксу.
- Для этого достаточно сделать $k := p(k - 1)$.
- А если $k = 0$?

Префикс-функция

- Вычисляем префикс-функцию p слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ префикс-функция равна нулю.
- При $i > 0$ мы знаем $k = p(i - 1)$.
- Это значит, что на предыдущей позиции $S_0 S_1 \dots S_{k-1} = S_{i-k} S_{i-k+1} \dots S_{i-1}$ — самый длинный суффикс, равный префиксу.
- Если $S_k = S_i$, то $p(i) = k + 1$.
- Если $k = 0$, просто сравним S_0 и S_i .
- Если же $S_k \neq S_i$, нужно рассмотреть следующий по длине суффикс в позиции $i - 1$, равный префиксу.
- Для этого достаточно сделать $k := p(k - 1)$.
- А если $k = 0$? Тогда $p(i) = 0$.

Пример

Пример: $S = a b a b a b a a b a b a b b$
 $p = 0 0 1 2 3 4 5 1 2 3 4 5 6 0$

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём префикс-функцию строки R .
- Те позиции, где она равна $|S|$, это концы вхождений S в T .

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём префикс-функцию строки R .
- Те позиции, где она равна $|S|$, это концы вхождений S в T .
- Пример 1:

R	=	a	b	a	#	a	b	a	b	a
p	=	0	0	1	0	1	2	3	2	3

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём префикс-функцию строки R .
- Те позиции, где она равна $|S|$, это концы вхождений S в T .
- Пример 2:

R	=	a	a	b	#	a	a	a	b	a	b	a
p	=	0	1	0	0	1	2	2	3	1	0	1

Алгоритм Кнута–Морриса–Пратта: код

```
1  vector <int> findKnuthMorrisPratt (string s, string t) {
2      vector <int> res;
3      string r = s + "#" + t;
4      int m = s.size (), len = r.size ();
5      int k = 0;
6      vector <int> p (len);
7      p[0] = k;
8      for (int i = 1; i < len; i++) {
9          while (k > 0 && r[i] != r[k])
10             k = p[k - 1];
11             if (r[i] == r[k])
12                 k += 1;
13             if (k == m)
14                 res.push_back (i - m * 2 + 1);
15             p[i] = k;
16         }
17     return res;
18 }
```

ВВОД:

aba

ababaaba

ВЫВОД:

1 3 6

Алгоритм Кнута–Морриса–Пратта: код

```
1  vector <int> findKnuthMorrisPratt (string s, string t) {
2      vector <int> res;
3      string r = s + "#" + t;
4      int m = s.size (), len = r.size ();
5      int k = 0;
6      vector <int> p (len);
7      p[0] = k;
8      for (int i = 1; i < len; i++) {
9          while (k > 0 && r[i] != r[k])
10             k = p[k - 1];
11             if (r[i] == r[k])
12                 k += 1;
13             if (k == m)
14                 res.push_back (i - m * 2 + 1);
15             p[i] = k;
16         }
17     return res;
18 }
```

ВВОД:

abb

ababaaba

ВЫВОД:

none

Анализ

- Общее время работы:

Анализ

- Общее время работы: $O(m + n)$.

Анализ

- Общее время работы: $O(m + n)$.
- Действительно, границы самого длинного суффикса с каждым действием движутся слева направо.

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфила: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции $i > 0$ вычислим z -функцию: длину наибольшего совпадения с префиксом, начиная с этой позиции.

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции $i > 0$ вычислим z -функцию: длину наибольшего совпадения с префиксом, начиная с этой позиции.
- Пример 1:

S	=	а	а	b	а	а	b	а	а	с
z	=	0	1	0	5	1	0	2	1	0

Идея

- Идея: ускорим наивное решение.
- Пусть мы ищем $S = \text{«аабаас»}$ в $T = \text{«аабаабаад»}$.
- Выяснили, что $\text{«аабаа»} = \text{«аабаа»}$, но $\text{«аабааб»} \neq \text{«аабаас»}$.
- Хочется, глядя только на найденное «аабаа» , сказать, что со второй или с третьей буквы вхождение «аабаас» начинаться не может, можно сразу перейти к четвёртой.
- Как это узнать?
- Рассмотрим пока только строку S .
- Для каждой позиции $i > 0$ вычислим z -функцию: длину наибольшего совпадения с префиксом, начиная с этой позиции.
- Пример 2:

S	=	а	б	а	с	а	б	а
z	=	0	0	1	0	3	0	1

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ определим значение функции равным нулю.

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ определим значение функции равным нулю.
- Из найденных подстрок, совпадающих с префиксами, будем помнить ту, которая кончается как можно правее.
- Пусть это подстрока $S[l_0...h_i)$.

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ определим значение функции равным нулю.
- Из найденных подстрок, совпадающих с префиксами, будем помнить ту, которая кончается как можно правее.
- Пусть это подстрока $S[l_0...h_i)$.
- Если позиция i лежит в $[l_0...h_i)$, посмотрим на позицию $i - l_0$.
- Вся подстрока $S[i - l_0...h_i - l_0)$ такая же, как $S[i...h_i)$.
- Так что оценка снизу на $z(i)$ — это $\min(h_i - i, z(i - l_0))$.

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ определим значение функции равным нулю.
- Из найденных подстрок, совпадающих с префиксами, будем помнить ту, которая кончается как можно правее.
- Пусть это подстрока $S[l_0 \dots h_i)$.
- Если позиция i лежит в $[l_0 \dots h_i)$, посмотрим на позицию $i - l_0$.
- Вся подстрока $S[i - l_0 \dots h_i - l_0)$ такая же, как $S[i \dots h_i)$.
- Так что оценка снизу на $z(i)$ — это $\min(h_i - i, z(i - l_0))$.
- Теперь, пока буквы $S[z(i)]$ и $S[i + z(i)]$ совпадают, будем увеличивать $z(i)$.

Z-функция

- Вычисляем функцию z слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- При $i = 0$ определим значение функции равным нулю.
- Из найденных подстрок, совпадающих с префиксами, будем помнить ту, которая кончается как можно правее.
- Пусть это подстрока $S[l_0...h_i)$.
- Если позиция i лежит в $[l_0...h_i)$, посмотрим на позицию $i - l_0$.
- Вся подстрока $S[i - l_0...h_i - l_0)$ такая же, как $S[i...h_i)$.
- Так что оценка снизу на $z(i)$ — это $\min(h_i - i, z(i - l_0))$.
- Теперь, пока буквы $S[z(i)]$ и $S[i + z(i)]$ совпадают, будем увеличивать $z(i)$.
- Проверим, не оказался ли конец строки — это позиция $i + z(i)$ — правее h_i .

Пример

Пример: $S = a a b a a b a a a b a a b b$
 $z = 0 1 0 5 1 0 2 6 1 0 3 1 0 0$

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём z -функцию строки R .
- Те позиции, где она равна $|S|$, это начала вхождений S в T .

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём z -функцию строки R .
- Те позиции, где она равна $|S|$, это начала вхождений S в T .
- Пример 1:

R	=	a	b	a	#	a	b	a	b	a
z	=	0	0	1	0	3	0	3	0	1

Поиск подстроки

- Как это помогает искать подстроку S в строке T ?
- Например, так: запишем $R = S\#T$.
- Найдём z -функцию строки R .
- Те позиции, где она равна $|S|$, это начала вхождений S в T .

• Пример 2:

R	=	a	a	b	#	a	a	a	b	a	b	a
z	=	0	1	0	0	2	3	1	0	1	0	1

Алгоритм Гасфилда: код

```

1  vector <int> findGusfield (string s, string t) {
2      vector <int> res;
3      string r = s + "#" + t;
4      int m = s.size (), len = r.size ();
5      vector <int> z (len);
6      z[0] = 0;
7      int lo = 0, hi = 0;
8      for (int i = 1; i < len; i++) {
9          if (i < hi)
10             z[i] = min (z[i - lo], hi - i);
11             while (i + z[i] < len && r[z[i]] == r[i + z[i]])
12                 z[i] += 1;
13             if (hi < i + z[i]) {
14                 lo = i;
15                 hi = i + z[i];
16             }
17             if (z[i] == m)
18                 res.push_back (i - m);
19         }
20     return res;
21 }

```

ВВОД:

aba
ababaaba

ВЫВОД:

1 3 6

Алгоритм Гасфила: код

```
1  vector <int> findGusfield (string s, string t) {
2      vector <int> res;
3      string r = s + "#" + t;
4      int m = s.size (), len = r.size ();
5      vector <int> z (len);
6      z[0] = 0;
7      int lo = 0, hi = 0;
8      for (int i = 1; i < len; i++) {
9          if (i < hi)
10             z[i] = min (z[i - lo], hi - i);
11             while (i + z[i] < len && r[z[i]] == r[i + z[i]])
12                 z[i] += 1;
13             if (hi < i + z[i]) {
14                 lo = i;
15                 hi = i + z[i];
16             }
17             if (z[i] == m)
18                 res.push_back (i - m);
19         }
20     return res;
21 }
```

ВВОД:

abb

ababaaba

ВЫВОД:

none

Анализ

- Общее время работы:

Анализ

- Общее время работы: $O(m + n)$.

Анализ

- Общее время работы: $O(m + n)$.
- Действительно, при росте совпадения число hi тоже растёт.

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфилда: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abc»}$.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 4: $S = \text{«abaabaab»}$, $G = \text{«aba»}$.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	b	a	b
p	=	0	0	1	2	3	4
z	=	0	0	4	0	2	0

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.
- Префикс-функция и z -функция:

$$\begin{array}{rcl} S & = & \text{x} \quad \text{y} \quad \text{z} \\ p & = & 0 \quad 0 \quad 0 \\ z & = & 0 \quad 0 \quad 0 \end{array}$$

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abc»}$.
- Префикс-функция и z -функция:

S	=	a	b	c	a	b	c	a
p	=	0	0	0	1	2	3	4
z	=	0	0	0	4	0	0	1

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 4: $S = \text{«abaabaab»}$, $G = \text{«aba»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	a	b	a	a	b
p	=	0	0	1	1	2	3	4	5
z	=	0	0	1	5	0	1	2	0

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGG\dots$
- Ответ: $|S| - p_{last}$.
- Ответ: $\min i : i + z_i = |S|$, или $i = n$, если такого нет.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	b	a	b
p	=	0	0	1	2	3	4
z	=	0	0	4	0	2	0
- Ответ: $|S| - p_{last}$.
- Ответ: $\min i : i + z_i = |S|$, или $i = n$, если такого нет.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGG\dots$
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.
- Префикс-функция и z -функция:
$$\begin{array}{r} S = \text{ x y z} \\ p = 0 \ 0 \ 0 \\ z = 0 \ 0 \ 0 \end{array}$$
- Ответ: $|S| - p_{last}$.
- Ответ: $\min i : i + z_i = |S|$, или $i = n$, если такого нет.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGG\dots$
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abc»}$.
- Префикс-функция и z -функция:

S	=	a	b	c	a	b	c	a
p	=	0	0	0	1	2	3	4
z	=	0	0	0	4	0	0	1
- Ответ: $|S| - p_{last}$.
- Ответ: $\min i : i + z_i = |S|$, или $i = n$, если такого нет.

Генератор строки

Задача:

- Дана строка S .
- Найдите её генератор: кратчайшую строку G , для которой S является префиксом бесконечной строки $G^* = GGGGG\dots$
- Пример 4: $S = \text{«abaabaab»}$, $G = \text{«aba»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	a	b	a	a	b
p	=	0	0	1	1	2	3	4	5
z	=	0	0	1	5	0	1	2	0
- Ответ: $|S| - p_{last}$.
- Ответ: $\min i : i + z_i = |S|$, или $i = n$, если такого нет.

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abcabca»}$.

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 4: $S = \text{«аааааа»}$, $G = \text{«а»}$.

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	b	a	b
p	=	0	0	1	2	3	4
z	=	0	0	4	0	2	0

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.
- Префикс-функция и z -функция:

$$S = \text{ x y z}$$

$$p = \text{ 0 0 0}$$

$$z = \text{ 0 0 0}$$

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abcabca»}$.
- Префикс-функция и z -функция:

S	=	a	b	c	a	b	c	a
p	=	0	0	0	1	2	3	4
z	=	0	0	0	4	0	0	1

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 4: $S = \text{«аааааа»}$, $G = \text{«а»}$.
- Префикс-функция и z -функция:

S	=	а	а	а	а	а	а
p	=	0	1	2	3	4	5
z	=	0	5	4	3	2	1

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Генератор: $g = |S| - p_{last}$.
- Генератор: $g = \min i : i + z_i = |S|$, или $i = n$, если такого нет.
- Если $|S|$ делится на g , то ответ — префикс S длины g .
- Если $|S|$ не делится на g , то ответ — вся строка S .

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 1: $S = \text{«ababab»}$, $G = \text{«ab»}$.
- Префикс-функция и z -функция:

S	=	a	b	a	b	a	b
p	=	0	0	1	2	3	4
z	=	0	0	4	0	2	0
- Генератор: $g = |S| - p_{last}$.
- Генератор: $g = \min i : i + z_i = |S|$, или $i = n$, если такого нет.
- Если $|S|$ делится на g , то ответ — префикс S длины g .
- Если $|S|$ не делится на g , то ответ — вся строка S .

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 2: $S = \text{«хуз»}$, $G = \text{«хуз»}$.
- Префикс-функция и z -функция:

S	$=$	x	y	z
p	$=$	0	0	0
z	$=$	0	0	0
- Генератор: $g = |S| - p_{last}$.
- Генератор: $g = \min i : i + z_i = |S|$, или $i = n$, если такого нет.
- Если $|S|$ делится на g , то ответ — префикс S длины g .
- Если $|S|$ не делится на g , то ответ — вся строка S .

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 3: $S = \text{«abcabca»}$, $G = \text{«abcabca»}$.
- Префикс-функция и z -функция:

S	=	a	b	c	a	b	c	a
p	=	0	0	0	1	2	3	4
z	=	0	0	0	4	0	0	1
- Генератор: $g = |S| - p_{last}$.
- Генератор: $g = \min i : i + z_i = |S|$, или $i = n$, если такого нет.
- Если $|S|$ делится на g , то ответ — префикс S длины g .
- Если $|S|$ не делится на g , то ответ — вся строка S .

Период строки

Задача:

- Дана строка S .
- Найдите её период: кратчайшую строку G , для которой S является конкатенацией нескольких копий G .
- Пример 4: $S = \text{«аааааа»}$, $G = \text{«а»}$.
- Префикс-функция и z -функция:

S	=	а	а	а	а	а	а
p	=	0	1	2	3	4	5
z	=	0	5	4	3	2	1
- Генератор: $g = |S| - p_{last}$.
- Генератор: $g = \min i : i + z_i = |S|$, или $i = n$, если такого нет.
- Если $|S|$ делится на g , то ответ — префикс S длины g .
- Если $|S|$ не делится на g , то ответ — вся строка S .

Содержание

- 1 Введение
 - Определения
 - Задача
 - Наивное решение
 - Наивный алгоритм: код
- 2 Хеширование строк
 - Идея
 - Полиномиальный хеш
 - Хеш подстроки
 - Алгоритм Рабина–Карпа: код
 - Анализ
 - Выбор констант
- 3 Префикс-функция
 - Идея
 - Префикс-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Кнута–Морриса–Пратта: код
 - Анализ
- 4 Z-функция
 - Идея
 - Z-функция
 - Пример
 - Поиск подстроки
 - Алгоритм Гасфила: код
 - Анализ
- 5 Задачи
 - Генератор строки
 - Период строки
- 6 Палиндромы
 - Задача
 - Идея
 - Радиусы палиндромов
 - Пример
 - Алгоритм Манакера: код
 - Анализ

Задача

Задача:

- Дана строка S .
- Найдите все её подстроки, являющиеся палиндромами.
- Пример: $S = \text{«ababaab»}$.

Задача

Задача:

- Дана строка S .
- Найдите все её подстроки, являющиеся палиндромами.
- Пример: $S = \text{«ababaab»}$.
- Палиндромов в различных позициях 13.
- В общем случае их может быть порядка $|S|^2$.
- Как организовать информацию, чтобы узнать о них за линейное время?

Задача

Задача:

- Дана строка S .
- Найдите все её подстроки, являющиеся палиндромами.
- Пример: $S = \text{«ababaab»}$.
- Палиндромов в различных позициях 13.
- В общем случае их может быть порядка $|S|^2$.
- Как организовать информацию, чтобы узнать о них за линейное время?
- Если $S[x\dots y]$ палиндром, то $S[x+1\dots y-1]$ — тоже палиндром.
- Получается, что для каждого центра можно найти максимальный размер палиндрома с таким центром.
- Но палиндромы бывают чётной и нечётной длины!

Идея

- Поставим какой-нибудь символ (например, «*») между каждыми двумя буквами S , а также до и после:
«abaа» → «*а*b*а*а*».
- Теперь бывшие палиндромы чётной длины — это палиндромы нечётной длины с центром в «*».

Идея

- Поставим какой-нибудь символ (например, «*») между каждыми двумя буквами S , а также до и после:
«aba» → «*a*b*a*a*».
- Теперь бывшие палиндромы чётной длины — это палиндромы нечётной длины с центром в «*».
- Идея: ускорим наивное решение, которое для каждого центра i ищет максимальный размер палиндрома $w(i)$ — такое число, что для всех $x < w(i)$ символы $S[i - x]$ и $S[i + x]$ совпадают.
- Пусть $S = \text{«abacaba»}$, и мы ищем только палиндромы нечётной длины.
- Выяснили, что «abacaba» — палиндром.

Идея

- Поставим какой-нибудь символ (например, «*») между каждыми двумя буквами S , а также до и после:
«aba» → «*a*b*a*a*».
- Теперь бывшие палиндромы чётной длины — это палиндромы нечётной длины с центром в «*».
- Идея: ускорим наивное решение, которое для каждого центра i ищет максимальный размер палиндрома $w(i)$ — такое число, что для всех $x < w(i)$ символы $S[i - x]$ и $S[i + x]$ совпадают.
- Пусть $S = \text{«abacaba»}$, и мы ищем только палиндромы нечётной длины.
- Выяснили, что «abacaba» — палиндром.
- *Внутри этой строки* вокруг правой буквы «b» всё точно так же, как вокруг левой буквы «b».

Идея

- Поставим какой-нибудь символ (например, «*») между каждыми двумя буквами S , а также до и после:
«abaа» → «*a*b*a*a*».
- Теперь бывшие палиндромы чётной длины — это палиндромы нечётной длины с центром в «*».
- Идея: ускорим наивное решение, которое для каждого центра i ищет максимальный размер палиндрома $w(i)$ — такое число, что для всех $x < w(i)$ символы $S[i - x]$ и $S[i + x]$ совпадают.
- Пусть $S = \text{«abacababa»}$, и мы ищем только палиндромы нечётной длины.
- Выяснили, что «abacaba» — палиндром.
- *Внутри этой строки* вокруг правой буквы «b» всё точно так же, как вокруг левой буквы «b».
- Пример 1:

S	=	a	b	a	c	a	b	a	b	a
w	=	1	2	1	4	1	2	3	2	1

Идея

- Поставим какой-нибудь символ (например, «*») между каждыми двумя буквами S , а также до и после: «abaа» → «*a*b*a*a*».
- Теперь бывшие палиндромы чётной длины — это палиндромы нечётной длины с центром в «*».
- Идея: ускорим наивное решение, которое для каждого центра i ищет максимальный размер палиндрома $w(i)$ — такое число, что для всех $x < w(i)$ символы $S[i - x]$ и $S[i + x]$ совпадают.
- Пусть $S = \text{«abacababa»}$, и мы ищем только палиндромы нечётной длины.
- Выяснили, что «abacaba» — палиндром.
- *Внутри этой строки* вокруг правой буквы «b» всё точно так же, как вокруг левой буквы «b».
- Пример 2:

S	=	a	b	a	c	a	b	b
w	=	1	2	1	3	1	1	1

Радиусы палиндромов

- Вычисляем радиусы палиндромов $w(i)$ слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.

Радиусы палиндромов

- Вычисляем радиусы палиндромов $w(i)$ слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- Из найденных палиндромов, будем помнить тот, который кончается как можно правее.
- Пусть это палиндром $S(lo...hi)$.

Радиусы палиндромов

- Вычисляем радиусы палиндромов $w(i)$ слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- Из найденных палиндромов, будем помнить тот, который кончается как можно правее.
- Пусть это палиндром $S(lo...hi)$.
- Если позиция i лежит в $(lo...hi)$, посмотрим на позицию $j = lo + hi - i$.
- Подстрока с центром в j и левым краем lo такая же, как подстрока с центром в i и правым краем hi .
- Так что оценка снизу на $w(i)$ — это $\min(hi - i, w(j))$.

Радиусы палиндромов

- Вычисляем радиусы палиндромов $w(i)$ слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- Из найденных палиндромов, будем помнить тот, который кончается как можно правее.
- Пусть это палиндром $S(lo...hi)$.
- Если позиция i лежит в $(lo...hi)$, посмотрим на позицию $j = lo + hi - i$.
- Подстрока с центром в j и левым краем lo такая же, как подстрока с центром в i и правым краем hi .
- Так что оценка снизу на $w(i)$ — это $\min(hi - i, w(j))$.
- Теперь, пока буквы $S[i - w(i)]$ и $S[i + w(i)]$ совпадают, будем увеличивать $w(i)$.

Радиусы палиндромов

- Вычисляем радиусы палиндромов $w(i)$ слева направо.
- Пусть мы находимся в позиции $i \in [0, n)$.
- Из найденных палиндромов, будем помнить тот, который кончается как можно правее.
- Пусть это палиндром $S(lo...hi)$.
- Если позиция i лежит в $(lo...hi)$, посмотрим на позицию $j = lo + hi - i$.
- Подстрока с центром в j и левым краем lo такая же, как подстрока с центром в i и правым краем hi .
- Так что оценка снизу на $w(i)$ — это $\min(hi - i, w(j))$.
- Теперь, пока буквы $S[i - w(i)]$ и $S[i + w(i)]$ совпадают, будем увеличивать $w(i)$.
- Проверим, не оказался ли конец найденного палиндрома — это позиция $i + w(i)$ — правее hi .

Пример

Пример: $S = a b c b a b d b a b a b c b$
 $w = 1 1 3 1 2 1 4 1 2 3 2 1 2 1$

Алгоритм Манакера: код

```
1  vector <int> palindromes (string t) {
2      int n = t.size ();
3      vector <int> w (n);
4      int lo = 0, hi = 0;
5      for (int i = 0; i < n; i++) {
6          if (i < hi)
7              w[i] = min (w[lo + hi - i], hi - i);
8          while (i - w[i] >= 0 && i + w[i] < n &&
9              t[i - w[i]] == t[i + w[i]])
10             w[i] += 1;
11         if (hi < i + w[i]) {
12             lo = i - w[i];
13             hi = i + w[i];
14         }
15     }
16     return w;
17 }
```

ВВОД:

ababaab

ВЫВОД:

13

Анализ

- Общее время работы:

Анализ

- Общее время работы: $O(m + n)$.

Анализ

- Общее время работы: $O(m + n)$.
- Действительно, при росте радиуса число hi тоже растёт.

Вопросы?

Вопросы?