Поиск палиндромов в потоковой модели*

Олег Меркурьев

Уральский федеральный университет

1 Введение

Свежим трендом в строковых алгоритмах является построение эффективных алгоритмов в потоковой модели, где строку можно читать как поток и алгоритму доступно сублинейная относительно входных данных память. Обычно в таких условиях используется рандомизированность и требуется решить задачу с высокой вероятностью. В этой работе рассматривается задача *Longest Palindromic Substring Problem* о нахождении наибольшего палиндрома (подстроки которая читается одинаково в обоих направлениях) в такой модели. Эта задача хорошо изучена в классических не потоковых моделях [4, 5, 6, 7]. И была рассмотрена в некоторых недавних работах в потоковой модели [2, 3]. В этих работах были доказаны некоторые нижние оценки на размер необходимой памяти, для решения этой задачи с аддитивной погрешностью E и мультипликативной погрешностью ε, и представлены алгоритмы, которые используют близкий к нижним оценкам размер памяти. Так в [3] доказано, что для алгоритмов Монте-Карло требуется $\Omega(\frac{n}{E})$ бит, для достижения аддитивной ошибки $E \leqslant 0.49n$. И $\Omega(\frac{\log n}{\log 1+\varepsilon})$ бит для достижения мультипликативной ошибки с множителем $1+\varepsilon$, при $n^{-0.99} \leqslant \varepsilon \leqslant n^{0.49}$. Также в [3] представлены алгоритмы для обоих случаев, которые по используемой памяти отклоняются от нижних оценок только на логарифмические множители, и при этом используют $O(n \log n)$ времени.

В этой работе представлены линейные алгоритмы, которые используют $O(1)$ времени на обработку каждого символа входного потока. При этом размер используемой памяти каждого из алгоритмов отличается от нижних границ лишь на логарифмические множители : биты в оценках памяти заменяются на слова длиной $\log n$ бит.

Работа состоит из трех основных частей, в которых приводятся алгоритмы нахождения в потоке самого длинного палиндрома с аддитивной ошибкой (раздел 4), с малой мультипликативной ошибкой (раздел 6) и с большой мультипликативной ошибкой (раздел 7).

Стоит также отметить, что все алгоритмы из этой работы прямолинейным образом обобщаются на палиндромы Уотсона-Крика, которые имеют важное значение в вычислительной биологии [9].

2 Модель и определения

Пусть $S \in \Sigma^n$ обозначает входной поток над алфавитом $\Sigma \subset \mathbb{N}$. Обозначим через $S[i]$ символ на позиции $1 \leqslant i \leqslant n$ и $S[i \dots j] = S[i]S[i+1] \dots S[j]$. В этой работе рассматривается потоковая модель: В один *проход* алгоритм *идёт* через весь входной поток S, считывая $S[i]$ в *итерацию* i прохода. В этой работе подразумевается, что алгоритму доступна сублинейная относительно входных данных память. Используется так называемая *word model*, в которой память равна количеству $O(\log n)$-битовых регистров. Будем говорить что алгоритм является *real-time* алгоритмом, если каждую итерацию прохода он совершает за $O(1)$ времени.

S содержит *палиндром* длины l в позиции $a \in \{1, \dots, n-l\}$, если $S[a+i-1] = S[a+l-i]$, для всех $a \in \{1, \dots l\}$. Длину наибольшего палиндрома в S обозначим через $l(S)$. В этой работе используется полиномиальный хэш, впервые введённый Карпом и Рабином [8] для сжатия строк

*часть результатов данной работы вошла в статью Paweł Gawrychowski, Oleg Merkurev, Arseny M. Shur, Przemysław Uznański «Tight Tradeoffs for Real-Time Approximation of Longest Palindromes in Streams», принятую на конференцию Combinatorial Pattern Matching 2016.

и использовавшийся впоследствии в потоковых задачах поиска шаблона. Для строки S определим прямой и обратный хэш следующим образом:

$$\phi^F_{r,p}(S) = \sum_{i=1}^{n} s[i] \cdot r^i \mod p$$

$$\phi^R_{r,p}(S) = \sum_{i=1}^{n} s[i] \cdot r^{n-i+1} \mod p$$

где p — это любое фиксированное простое число из интервала $[n^{4+\alpha}, n^{5+\alpha}]$ и r случайно выбрано из $\{1, \ldots, p-1\}$. Будем писать ϕ^F (ϕ^R соответственно) вместо $\phi^F_{k,p}$ ($\phi^R_{k,p}$ соответственно), если r и p зафиксированы. Определим для $1 \leqslant i \leqslant j \leqslant n$ хэш $F^F(i,j)$ как прямой хэш от $S[i \ldots j]$, т.е. $F^F(i,j) = \phi^F(S[i \ldots j])$, и хэш $F^R(i,j)$ как обратный хэш от $S[i \ldots j]$, т.е. $F^R(i,j) = \phi^R(S[i \ldots j])$. Следующие утверждения неоднократно использовались ранее, например в [1].

Утверждение 2.1 $F^F(i,j) = r^{-(i-1)} \left(\phi^F(S[1 \ldots j]) - \phi^F(S[1 \ldots i-1])\right) \mod p$.

Утверждение 2.2 $F^R(i,j) = \phi^F(S[1 \ldots j]) - r^{j-i+1}\phi^R(S[1 \ldots i-1]) \mod p$.

За $I(i)$ обозначим упорядоченный набор $\{i, F^F(1,i-1), F^R(1,i-1), r^{-(i-1)} \mod p, r^i \mod p\}$.

Утверждение 2.3 *Зная $I(i)$ и $I(j+1)$ можно за время $O(1)$ проверить, является ли $S[i \ldots j]$ палиндромом.*

Доказательство. Для этого необходимо вычислить и проверить на равенство $F^F(i,j)$ и $F^R(i,j)$. Согласно утверждениям 2.1, 2.2 для этого требуется константное число операций. ■

Развёрнутую строку S будем обозначать $\bar{S}$. Представленные в этой работе алгоритмы основываются на предположении, что при сравнении хэшей не произойдёт ошибок, то есть для любых двух различных строк, хэши от которых будут сравниваться в процессе работы алгоритма, хэши будут различными.
Будем говорить что в S нет коллизий, если нет двух различных подстрок, каждая из которых принадлежит S или $\bar{S}$, таких что прямые хеши от них совпадают.

Утверждение 2.4 *Пусть u и v это две различные строки длины l, где $l \leqslant n$ и p произвольное простое число. Тогда количество $1 < r < p$ таких, что $\phi^F_{r,p}(u) = \phi^F_{r,p}(u)$, не превосходит l.*

Доказательство. Аналогичное утверждение было доказано в [1, Theorem2.1].
Равенство хэшей $\sum_{i=1}^{l} v(i) \cdot r^i \mod p = \sum_{i=1}^{l} u(i) \cdot r^i \mod p$ означает, что многочлен $\sum_{i=1}^{l} (u(i) - v(i))\, x^i$ имеет корень r в F_p. Но так как F_p это поле, любой многочлен степени l имеет не более чем l корней. ■

Утверждение 2.5 *Для любого простого p есть не менее $p - n^4$ значений $r \in [1, p-1]$ таких, что при использовании хешей с параметрами p и r в S не окажется коллизий.*

Доказательство.
Пар различных подстрок длины i не более $(n-i+1)(n-i) + (n-i+1)^2$, каждая пара даёт i плохих значений для r. Тогда всего плохих значений для r не более
$\sum_{i=1}^{n} i\left((n-i+1)(n-i) + (n-i+1)^2\right) = \frac{n^4}{6} + \frac{n^3}{2} + \frac{n^2}{3} < n^4$. ■

Из этого следует, что для любой строки $S \in \Sigma^n$ вероятность того, что в S есть коллизия, при выборе параметров p и r описанным выше способом, не превосходит $\frac{1}{n^\alpha}$.

Утверждение 2.6 *Пусть алгоритм совершает K сравнений подстрок с помощью хэшей, тогда для любого простого p есть не менее $p - n \cdot K$ значений $r \in [1, p-1]$ таких, что при использовании хешей с параметрами p и r не произойдёт ошибок.*

В частности из этого утверждения следует, что при выборе параметров p и r описанным выше способом, *real-time* алгоритм, совершающий на каждом шаге не более C сравнений, столкнётся с ошибками хешей с вероятностью не превосходящей $\frac{C}{n^{2+\alpha}}$.

3 Алгоритм простой приближённый корневой

В этом разделе приведён алгоритм, который за один проход приближённо решает задачу *Longest Palindromic Substring Problem* с аддитивной погрешностью $E \in [1, n]$, используя $O(\frac{n}{E})$ памяти и $O(\frac{n^2}{E})$ времени. Алгоритм найдёт палиндром длины $\tilde{l}(S)$, такой что $l(S) - E \leqslant \tilde{l}(S) \leqslant l(S)$.

В этом и следующих алгоритмах понадобится двусвязный список, в каждом узле которого хранится $I(j)$, для некоторого префикса j, будем обозначать его SP. В некоторых алгоритмах также будет поддерживаться текущий узел в SP, он будет обозначаться *sp*. Длина наибольшего уже найденного палиндрома будет обозначаться *answer*. При этом ответом на задачу является его позиция, подразумевается что она также запоминается. Обозначим $t_E = \lfloor \frac{E}{2} \rfloor$, в коде алгоритмов будет обозначаться как t.

Основная идея алгоритма заключается в том, что для каждого j, делящегося на t_E, в SP добавляется $I(j)$. И с помощью SP на i-ой итерации проверяются на палиндромность все строки вида $S[t_E k \dots i]$, где $1 \leqslant kt_E \leqslant i$.

Замечание 3.1 *В этом и следующих алгоритмах поддерживается инвариант: к началу итерации i в памяти хранится $I(i)$.*

Алгоритм 3.1 *Алгоритм простой приближённый корневой, i-ая итерация:*

```
1  Если  i mod t = 0
2          Добавить I(i) в начало SP
3  Считать S[i], преобразовать I(i) в I(i + 1)
4  Для всех узлов v в SP:
5          Если S[v.i...i] является палиндромом
6                  Обновить ответ палиндромом S[v.i...i]
```

Утверждение 3.1 *Алгоритм использует $O(\frac{n}{t_E}) = O(\frac{n}{E})$ памяти и $O(\frac{n^2}{t_E}) = O(\frac{n^2}{E})$ времени.*

Доказательство. На каждую итерацию тратится $O(\frac{n}{t_E})$ времени, всего n итераций. ■

Утверждение 3.2 *Алгоритм корректен.*

Доказательство. Рассмотрим любой наибольший палиндром $S[a \dots b]$ в строке S.
Обозначим $x = \lceil \frac{a}{t_E} \rceil t_E$. Тогда $a \leqslant x < a + t_E$. На итерации x в SP была добавлена $I(x)$.
На итерации $b - (x - a)$ был найден палиндром $S[x \dots b - (x - a)]$. Его длина
$b - (x - a) - x + 1 = b - a - 2(x - a) + 1 > (b - a + 1) - 2t_E = l(S) - 2\lfloor \frac{E}{2} \rfloor \geqslant l(S) - E$. ■

4 Алгоритм приближённый корневой

В этом разделе мы улучшим конструкцию из раздела 3, получая алгоритм, который за один проход приближённо решает задачу *Longest Palindromic Substring Problem* с аддитивной погрешностью E, используя $O(\frac{n}{E})$ памяти и всего $O(n)$ времени, при этом каждая итерация выполняется за $O(1)$. Алгоритм найдёт палиндром длины $\tilde{l}(S)$, такой что $l(S) - E \leqslant \tilde{l}(S) \leqslant l(S)$

Для улучшения простого приближённого корневого алгоритма достаточно заметить один факт:

Утверждение 4.1 *За одну итерацию простого приближённого корневого алгоритма длина наибольшего из найденных палиндромов не может увеличиться больше чем на $2 \cdot t_E$.*

Доказательство. Рассмотрим итерацию i. Самый длинный палиндром, найденный на этой итерации обозначим $S[j \dots i]$. Если $i - j + 1 \leqslant 2t_E$ то утверждение выполняется. Иначе рассмотрим $S[j + t_E \dots i - t_E]$ — это тоже палиндром, и он должен был быть найден на итерации $i - t_E$. Его длина $i - j + 1 - 2t_E$. Значит и в этом случае утверждение выполняется. ■

Из этого утверждения следует, что на каждой итерации есть только два узла в SP, с помощью которых ответ может улучшиться.

Алгоритм 4.1 *Алгоритм приближённый корневой, i-ая итерация:*

```
1   Если  i mod t = 0
2           Добавить I(i) в начало SP
3           Если i = t
4                   Присвоить sp указатель на начало SP
5   Считать S[i], преобразовать I(i) в I(i + 1)
6   sp = предыдущий(sp)
7   Пока (i - sp.i + 1 <= answer) и (sp не последний узел в SP)
8           sp = следующий(sp)
9   Для v существующих из {sp, следующий(sp) }
10          Если S[v.i...i] является палиндромом
11                  Обновить ответ палиндромом S[v.i...i]
```

Теорема 4.1 *Алгоритм корректен, использует* $O(\frac{n}{t_E}) = O(\frac{n}{E})$ *памяти, всего* $O(n)$ *времени, при этом каждая итерация выполняется за* $O(1)$.

Доказательство. На итерации i проверяются на палиндромность все подстроки вида $S[t_E k \dots i]$, где $answer < i - t_E k + 1 \leqslant answer + 2t_E$, где $answer$ — это длина наибольшего палиндрома, найденного до итерации i. Из этого факта, утверждения 4.1 и корректности простого приближённого корневого алгоритма следует корректность рассматриваемого алгоритма.
По используемой памяти рассматриваемый алгоритм не отличается от простого приближённого корневого алгоритма, а значит оценка на количество используемой памяти может быть взята из утверждения 3.1. Для оценки времени работы одной итерации алгоритма, достаточно заметить что цикл сдвигающий *sp* не может совершить более двух итераций. ■

5 Алгоритм простой приближённый логарифмический

В этом разделе приведён алгоритм, который за один проход приближённо решает задачу *Longest Palindromic Substring Problem* с мультипликативной погрешностью $\varepsilon \in (0, 1]$, используя $O(\frac{\log n}{\varepsilon})$ памяти и $O(\frac{n \log n}{\varepsilon})$ времени.
Алгоритм найдёт палиндром длины $\tilde{l}(S)$, такой что $\tilde{l}(S) \geqslant \frac{l(S)}{1+\varepsilon}$.

Замечание 5.1 *Ниже* log *везде обозначает двоичный логарифм.*

Определим $q_\varepsilon = \lceil -\log \frac{\varepsilon}{2} \rceil$.

Утверждение 5.1 *При* $\varepsilon \leqslant 1$, *достаточно чтобы выполнялось:* $\tilde{l}(S) \geqslant l(S) - l(S)\frac{\varepsilon}{2}$.

Доказательство. $\tilde{l}(S) \geqslant l(S) - l(S)\frac{\varepsilon}{2} \geqslant l(S) - \tilde{l}(S)\varepsilon \Rightarrow \tilde{l}(S) + \tilde{l}(S)\varepsilon \geqslant l(S) \Rightarrow \tilde{l}(S) \geqslant \frac{l(S)}{1+\varepsilon}$ ■

Основное отличие логарифмических алгоритмов от корневых: в SP попадают $I(i)$ для всех i, но со временем они оттуда удаляются. И основная сложность - подобрать хорошую функцию от i, она будет обозначаться $ttl(i)$, которая обозначает, сколько итераций $I(i)$ хранится в SP.
Эта функция должна обеспечивать корректность алгоритма и логарифмический размер SP в каждый момент времени.

Алгоритм 5.1 *Алгоритм простой приближённый логарифмический, i-ая итерация:*

```
1   Добавить I(i) в начало SP
2   Для всех узлов v в SP:
3           Если v.i + ttl(v.i) = i
4                   Удалить v из SP
5   Считать S[i], преобразовать I(i) в I(i + 1)
6   Для всех узлов v в SP:
7           Если S[v.i...i] является палиндромом
8                   Обновить ответ палиндромом S[v.i...i]
```

Осталось определить функцию $ttl(i)$.

Определим функцию $\beta(i)$ как номер младшего единичного бита i. Определим $ttl(i) = 2^{q_\varepsilon+2+\beta(i)}$.
То есть по сути $ttl(i)$ это младший единичный бит i, домноженный на $2^{q_\varepsilon+2}$.

Стоит обратить внимание на содержание SP после итерации i.
В таблице отмечены j, для которых $I(j)$ хранится в SP после итерации 22, при $q_\varepsilon = 1$:

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
$ttl(j)$	8	16	8	32	8	16	8	64	8	16	8	32	8	16	8	128	8	16	8	32	8	16
SP	−	−	−	+	−	−	−	+	−	+	−	+	−	+	+	+	+	+	+	+	+	+

Для описания общего случая используются следующие утверждения:

Утверждение 5.2 *Для любых целых $a > 1$ и $b \geqslant 0$, существует единственный $j \in [a, a + 2^b)$, такой что $ttl(j) \geqslant 2^{q_\varepsilon+2+b}$.*

Доказательство. $ttl(j) \geqslant 2^{q_\varepsilon+2+b} \Leftrightarrow 2^{\beta(j)} \geqslant 2^b \Leftrightarrow x = 2^b y$, для некоторого целого y.
На отрезке целых чисел длины 2^b существует единственный элемент, делящийся на 2^b. ■

Утверждение 5.3 *После итерации i, SP как множество совпадает с множеством*

$$\{I(j)|j \in (i - 2^{q_\varepsilon+2}, i]\} \cup \bigcup_{x=1}^{\infty} \{I(j)|j > 0, j \in (i - 2^{q_\varepsilon+2+x}, i - 2^{q_\varepsilon+2+x-1}], 2^x \vdots j\} \quad (1)$$

Утверждение 5.4 *Размер SP после любой итерации есть $O(2^{q_\varepsilon} \log n)$.*

Доказательство. Оценим размер SP, воспользовавшись формулой 1.
Отдельно оценим мощность множества, соответствующего отрезку, на котором $I(j)$ хранится в SP для каждого j из отрезка: $|\{I(j)|j \in (i - 2^{q_\varepsilon+2}, i]\}| = 2^{q_\varepsilon+2}$.
Теперь оценим суммарную мощность оставшихся множеств. Для x, таких что $i - 2^{q_\varepsilon+2+x-1} \leqslant 0$, мощность соответствующего множества равна 0. Мощность одного произвольного множества из объединения по x: $|\{I(j)|j > 0, j \in (i - 2^{q_\varepsilon+2+x}, i - 2^{q_\varepsilon+2+x-1}], 2^x \vdots j\}| \leqslant \frac{2^{q_\varepsilon+2+x-1}}{2^x} = 2^{q_\varepsilon+1}$.
Есть всего $O(\log n)$ таких x, что $i - 2^{q_\varepsilon+2+x-1} > 0$. А значит, суммарный размер множеств есть $O(2^{q_\varepsilon+1} \log n) = O(2^{q_\varepsilon} \log n)$ ■

Утверждение 5.5 *Алгоритм использует $O(\frac{\log n}{\varepsilon})$ памяти и $O(\frac{n \log n}{\varepsilon})$ времени.*

Доказательство. Памяти используется $O(2^{q_\varepsilon} \log n) = O(\frac{\log n}{\varepsilon})$, каждая итерация выполняется за размер SP, то есть за $O(\frac{\log n}{\varepsilon})$, суммарно по всем итерациям $O(\frac{n \log n}{\varepsilon})$. ■

Утверждение 5.6 *Алгоритм корректен.*

Доказательство. Рассмотрим любой наибольший палиндром $S[a \dots b]$ в строке S.
Определим $d = \max\{x|2^x \leqslant l(S)\}$.
Если $d < q_\varepsilon + 2$, то палиндром $S[a \dots b]$ будет найден точно, так как $a + ttl(a) \geqslant a + 2^{q_\varepsilon+2} \geqslant a + 2^{d+1} > a + l(S) > b$.
Иначе, по утверждению 5.2 существует единственный $c \in [a, a + 2^{d-q_\varepsilon-1})$, для которого $ttl(c) \geqslant 2^{d+1}$.
На итерации $b-(c-a)$ будет обнаружен палиндром $S[a+(c-a) \dots b-(c-a)]$, так как это палиндром, а префикс c хранится до итерации $c + ttl(c) \geqslant a + ttl(c) \geqslant a + 2^{d+1} > b \geqslant b - (c - a)$.
Длина этого палиндрома $b - (c - a) - (a + (c - a)) + 1 = b - a + 1 - 2(c - a) = l(S) - 2(c - a) \geqslant l(S) - 2^{d-q_\varepsilon} \geqslant l(S) - \frac{l(S)}{2^{q_\varepsilon}} = l(S)(1 - 2^{q_\varepsilon}) \geqslant l(S)(1 - \frac{\varepsilon}{2})$. ■

6 Алгоритм приближённый логарифмический

В этом разделе мы улучшим конструкцию из раздела 5, получая алгоритм, который за один проход приближённо решает задачу *Longest Palindromic Substring Problem* с мультипликативной погрешностью $\varepsilon \in (0, 1]$, используя $O(\frac{\log n}{\varepsilon})$ памяти и всего $O(n)$ времени, при этом каждая итерация выполняется за $O(1)$.
Алгоритм найдёт палиндром длины $\tilde{l}(S)$, такой что $\tilde{l}(S) \geqslant \frac{l(S)}{1+\varepsilon}$.

У простого приближённого логарифмического алгоритма есть две медленные части на каждой итерации: удаление узлов из SP и проверка некоторых подстрок на палиндромность.
Для ускорения проверки на палиндромность, можно использовать ту же идею, что и при ускорении корневого алгоритма: на самом деле нужно проверять только $O(1)$ подстрок.

Утверждение 6.1 *Для любых $a < b < c < d$, таких что $I(a), I(b), I(c), I(d)$ расположены подряд в SP после некоторой итерации, выполняется: $b - a \leqslant d - b$.*

Доказательство. Рассмотрим итерацию i:
Если $a \in (i - 2^{q_\varepsilon+2}, i]$, то $b - a = 1$, а $d - b \geqslant 2$, и утверждение выполняется. Иначе, пусть $a \in (i - 2^{q_\varepsilon+2+x}, i - 2^{q_\varepsilon+2+x-1}]$, тогда $\beta(a) \geqslant x$; кроме того, для любого k, для которого выполняется $a < k \leqslant i$ и $\beta(k) \geqslant x$, $I(k)$ лежит в SP. Следовательно $b - a \leqslant 2^x$. Каждый из b, c, d принадлежит либо интервалу $(i-2^{q_\varepsilon+2+x}, i-2^{q_\varepsilon+2+x-1}]$, либо $(i-2^{q_\varepsilon+2+x-1}, i-2^{q_\varepsilon+2+x-2}]$. Значит $d-b \geqslant 2 \cdot 2^{x-1} = 2^x \geqslant b - a$. ■

Утверждение 6.2 *На одной итерации достаточно проверить 3 подстроки на палиндромность.*

Доказательство. Это следствие из утверждения 6.1. Пусть $S[a \dots i]$ оказалась палиндромом. Надо показать что $S[d \dots i]$ не длиннее, чем ответ перед i-ой итерацией.
Подстрока $S[a + (b - a) \dots i - (b - a)]$ также палиндром, и на шаге $i - (b - a)$, $I(b)$ лежало в SP. А значит ответ до i-ой итерации не меньше длины этого палиндрома, которая равна:
$i - a + 1 - 2(b - a) \geqslant i - a + 1 - (b - a) - (d - b) = i - d + 1$. ■

Для ускорения удаления из SP нужно отметить ещё одно замечательное свойство функции $ttl(i)$.

Утверждение 6.3 *Функция $x \to x + ttl(x)$ инъективна.*

Доказательство. $\beta(x + ttl(x)) = \beta(x)$, а значит можно построить в некотором смысле обратную к $i + ttl(i)$ функцию: $dh(x) = x - ttl(x)$, и для всех $i > 0$ выполняется: $dh(i + ttl(i)) = i$. ■

Из утверждения 6.3 следует, что на каждом шаге из SP удаляется не больше одного префикса.
Для того, чтобы удаление выполнялось за $O(1)$, потребуется дополнительная структура.

Связный список макимальных по включению отрезков из едениц в битовой записи числа x будем обозначать $BS(x)$.

К примеру, для числа 12345 двоичное представление и BS выглядят следующим образом:

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	0	0	0	1	1	1	0	0	1

$BS(12345) = \{[0, 0], [3, 5], [10, 10], [12, 13]\}$

Утверждение 6.4 *$BS(x)$ занимает $O(\log x)$ памяти.*

Утверждение 6.5 *$BS(x)$ можно преобразовать в $BS(x + 1)$ за $O(1)$.*

Доказательство. Путь первый отрезок в $BS(x)$ это $[a, b]$.
Если $BS(x)$ пусто или $a > 2$, то $BS(x+1)$ получается из $BS(x)$ добавлением в начало отрезка $[0, 0]$.
Если $a = 1$, то $BS(x + 1)$ получается из $BS(x)$ заменой первого отрезка на отрезок $[0, b]$.

Рассмотрим теперь случай $a = 0$. Пусть второй отрезок в $BS(x)$ это $[c, d]$. Если в $BS(x)$ только один отрезок или $c > b + 2$, то $BS(x + 1)$ получается из $BS(x)$ заменой первого отрезка на отрезок $[b + 1, b + 1]$. Если же $c = b + 2$, то $BS(x + 1)$ получается из $BS(x)$ заменой первых двух отрезков на отрезок $[c - 1, d]$. ■

Утверждение 6.6 *Из $BS(x)$ можно получить $\beta(x)$ за $O(1)$.*

Имея на i-ой итерации алгоритма значение $BS(i)$ можно за $O(1)$ вычислить $\beta(i)$, а значит, и β от удаляемого из SP элемента.

Утверждение 6.7 *Если $a < b$ и $ttl(a) = ttl(b)$, то $I(a)$ удаляется из SP раньше, чем $I(b)$.*

Другими словами, информация о префиксах с одинаковым ttl или, что то же самое с одинаковым β добавляются и удаляются из SP в одном и том же порядке. А значит для каждого возможного β можно хранить очередь, в которой будут ссылки на узлы SP соответсвующие префиксам с таким β. Всего очередей понадобится $O(\log n)$. Назовём массив этих очередей QU, тогда $QU(x)$ это очередь ссылок на узлы SP, в которых лежат $I(j)$ такие, что $\beta(j) = x$.

Алгоритм 6.1 *Алгоритм приближённый логарифмический, i-ая итерация:*

```
1   Добавить I(i) в начало SP
2   Если i = 1
3           Присвоить sp указатель на начало SP
4   Построить BS(i), используя BS(i - 1).
5   Посчитать beta(i), используя BS(i).
6   Если QU(beta(i) ) не пуста
7           Удалить из SP узел на который ссылается первый элемент QU(beta(i) )
8                   Если удаляется элемент, на который ссылается sp, перед удалением:
9                           sp = следующий(sp)
10          Удалить из очереди QU(beta(i) ) первый элемент.
11  Добавить указатель на начало SP к очереди QU(beta(i) ).
12  Считать S[i], преобразовать I(i) в I(i + 1)
13  sp = предыдущий(sp)
14  Пока (i - sp.i + 1 <= answer) и (sp не последний узел в SP)
15          sp = следующий(sp)
16  Для v имеющих смысл из {sp, следующий(sp), следующий(следующий(sp) ) }
17          Если S[v.i...i] является палиндромом
18                  Обновить ответ палиндромом S[v.i...i]
```

Теорема 6.1 *Алгоритм корректен, использует $O(\frac{\log n}{\varepsilon})$ памяти, всего $O(n)$ времени, при этом каждая итерация выполняется за $O(1)$.*

Доказательство. Модифицированный алгоритм также поддерживает инвариант описанный в утверждении 5.3. При этом утверждение 6.2 показывает, что рассматриваемый алгоритм найдёт такой же ответ как и простой приближённый логарифмический алгоритм. А значит, по утверждению 5.6 он корректен.
Размер $BS(x)$ есть $O(\log n)$. Всего очередей QU используется $O(\log n)$, при этом их суммарный размер равен размеру SP. Следовательно, согласно утверждению 5.4, всего алгоритм использует $O(\frac{\log n}{\varepsilon})$ памяти.
Цикл изменяющий sp не может совершить более трёх итераций. Пересчёт $BS(i)$ по утверждению 6.5 выполняется за $O(1)$. Вычисление $\beta(i)$ выполняется за $O(1)$ по утверждению 6.6. А значит в целом каждая итерация алгоритма выполняется за $O(1)$. ■

7 Алгоритм приближённый логарифмический для больших ε.

В этом разделе мы модифицируем конструкцию из раздела 6, получая алгоритм, который за один проход приближённо решает задачу *Longest Palindromic Substring Problem* с мультипликативной погрешностью $\varepsilon \in [15, n]$, используя $O(\frac{\log n}{\log(1+\varepsilon)})$ памяти и всего $O(n)$ времени, при этом каждая

итерация выполняется за $O(1)$.
Алгоритм найдёт палиндром длины $\tilde{l}(S)$, такой что $\tilde{l}(S) \geqslant \frac{l(S)}{1+\varepsilon}$.

Заметим, что $\frac{\log n}{\log(1+\varepsilon)} = \log_{1+\varepsilon} n$.

В алгоритме из раздела 6 важную роль играет структура SP. В ней хранится информация о некоторых префиксах, при этом для какого-то отрезка хранится информация о всех префиксах, для следующего отрезка такой же длины информация хранится о каждом втором префиксе, и потом длина отрезка увеличивается в 2 раза и плотность хранимых префиксов в 2 раза уменьшается. Для того чтобы добиться размера SP порядка $\log_{1+\varepsilon} n$, длина отрезка должна увеличиваться примерно в $1+\varepsilon$ раз, а плотность префиксов уменьшаться во столько же раз.

Определим $k_\varepsilon = \lfloor \log(1+\varepsilon) \rfloor - 2$. Из ограничения $\varepsilon \geqslant 15$ следует, что $k_\varepsilon \geqslant 2$.

Роль функции $\beta(i)$ будет выполнять функция $\gamma_\varepsilon(i) = \lfloor \frac{\beta(i)}{k_\varepsilon} \rfloor$, которая по сути вычисляет номер младшего ненулевого разряда в 2^{k_ε}-ичной записи числа i.
Опеределим также функцию $\chi_\varepsilon(i) = \lfloor \frac{i}{2^{\gamma_\varepsilon(i) k_\varepsilon}} \rfloor \mod 2^{k_\varepsilon}$, которая вычисляет значение младшего ненулевого разряда в 2^{k_ε}-ичной записи числа i. Функцию ttl переопределим следующим образом:

$$ttl(i) = \begin{cases} 2^{2+k_\varepsilon \gamma_\varepsilon(i)}, & \text{если } \chi_\varepsilon(i) \neq 2^{k_\varepsilon} - 4; \\ 2^{3+k_\varepsilon \gamma_\varepsilon(i)}, & \text{иначе.} \end{cases}$$

Поясним эту формулу: $2^{2+k_\varepsilon \gamma_\varepsilon(i)}$ это 4 сдвинутое в младший ненулевой разряд i. Таким образом условие $\chi_\varepsilon(i) \neq 2^{k_\varepsilon} - 4$ говорит, что прибавление 4 в младший ненулевой разряд не занулит его. В противном случае прибавляется 8, и в самом разряде остаётся 4. Отметим, что $\chi_\varepsilon(i + ttl(i)) = 4$ тогда и только тогда, когда $\chi_\varepsilon(i) = 2^{k_\varepsilon} - 4$.

Утверждение 7.1 $\gamma_\varepsilon(i + ttl(i)) = \gamma_\varepsilon(i)$.

Утверждение 7.2 *Функция $x \to x + ttl(x)$ инъективна.*

Доказательство. Как и в утверждении 6.3 продемонстрируем способ вычислить x по значению $x + ttl(x)$.
Определим функцию dh следующим образом:

$$dh(x) = \begin{cases} x - 2^{2+k_\varepsilon \gamma_\varepsilon(x)}, & \text{если } \chi_\varepsilon(x) \neq 4; \\ x - 2^{3+k_\varepsilon \gamma_\varepsilon(x)}, & \text{иначе.} \end{cases}$$

Тогда для любого $x > 0$ справедливо равенство $dh(x + ttl(x)) = x$. ■

Утверждение 7.3 *Для любых целых $a > 1$ и $b \geqslant 0$, существует единственный $j \in [a, a + 2^{k_\varepsilon b})$, такой что $ttl(j) \geqslant 2^{2+k_\varepsilon b}$.*

Доказательство. $ttl(j) \geqslant 2^{2+k_\varepsilon b} \Leftrightarrow k_\varepsilon \lfloor \frac{\beta(j)}{k_\varepsilon} \rfloor \geqslant k_\varepsilon b \Leftrightarrow \beta(j) \geqslant k_\varepsilon b$.
На отрезке целых чисел длины $2^{k_\varepsilon b}$ существует единственный элемент, делящийся на $2^{k_\varepsilon b}$. ■

Утверждение 7.4 *Для любых целых $a > 1$ и $b \geqslant 0$, существует не более одного $j \in [a, a + 2^{2+k_\varepsilon b})$, такого что $ttl(j) \geqslant 2^{3+k_\varepsilon b}$.*

Доказательство. Из неравенства $ttl(j) \geqslant 2^{3+k_\varepsilon b}$ следует, что либо $\gamma_\varepsilon(j) > b$, либо $\gamma_\varepsilon(j) = b$ и при этом $\chi_\varepsilon(i) = 2^{k_\varepsilon} - 4$. В обоих случаях $2^{2+k_\varepsilon b}$ делит j. А значит двух различных подходящих j на отрезке длины $2^{2+k_\varepsilon b}$ быть не может. ■

Покажем, что алгоритм 5.1 с изменённой функцией ttl будет работать в условиях, рассматриваемых в этом разделе. Возможность эффективной реализации функции ttl оставим пока вне рассмотрения.

Рассмотрим содержание SP после итерации i. Для SP как для множества выполняется:

$$SP = \{I(j) | j > 0, j + ttl(j) > i\} = \bigcup_{b=0}^{\lfloor \log_{2^{k_\varepsilon}}(i) \rfloor} \{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i], j + ttl(j) > i\}$$

В этом объединении каждое следующее множество содержит все предыдущие. Исключим те элементы, которые точно попали в предыдущие множества:

$$SP = \bigcup_{b=0}^{\lfloor \log_{2^{k_\varepsilon}}(i) \rfloor} \{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i], j + ttl(j) > i, ttl(j) \geqslant 2^{2+k_\varepsilon b}\} \quad (2)$$

Отметим, что объединение в формуле 2 всё равно может быть не дизъюнктным.

Утверждение 7.5 *Мощность каждого из множеств, участвующих в объединении в формуле 2, не превосходит* 5.

Доказательство. Рассмотрим произвольное $b > 0$.
Покажем, что $|\{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i], j + ttl(j) > i, ttl(j) \geqslant 2^{2+k_\varepsilon b}\}| \leqslant 5$.
Разобьём имеющееся множество на два, и оценим мощность каждого из них.
По утверждению 7.3, $|\{I(j) | j > 0, j \in (i - 2^{2+k_\varepsilon b}, i], ttl(j) \geqslant 2^{2+k_\varepsilon b}\}| \leqslant 4$.
По утверждению 7.4, $|\{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i - 2^{2+k_\varepsilon b}], ttl(j) \geqslant 2^{3+k_\varepsilon b}\}| \leqslant 1$.
■

Утверждение 7.6 *Размер SP после любой итерации есть* $O(\log_{1+\varepsilon} n)$.

Теорема 7.1 *Алгоритм 5.1 корректен в условиях, рассматриваемых в этом разделе.*

Доказательство. Определим $x = \lceil \frac{l(S)}{2^{k_\varepsilon+2}} \rceil$. Заметим, что $x \geqslant \frac{l(S)}{1+\varepsilon}$. Следовательно, если алгоритм найдём палиндром длины хотя бы x, он будет корректным. Определим y как минимальное целое число той же чётности что и $l(S)$, которое больше либо равно x. По выбору y, в строке S гарантированно есть палиндром длины y.
Рассмотрим отдельно случай $l(S) < 2^{k_\varepsilon+3}$. В этом случае $y \leqslant x + 1 \leqslant \frac{l(S)}{2^{k_\varepsilon+2}} + 2 \leqslant 3$. А так как $ttl(i) \geqslant 4$ для любого $i > 0$, палиндром длины y будет найден.
Рассмотрим теперь общий случай. Определим $d = \min\{z | 2^{k_\varepsilon z} \geqslant y\}$. По утверждению 7.3, найдётся $j \in (a + \frac{l(s)-y}{2} - 2^{k_\varepsilon d}, a + \frac{l(s)-y}{2}]$ такой, что $ttl(j) \geqslant 2^{2+k_\varepsilon d}$. Для доказательства корректности алгоритма достаточно показать, что $S[j \dots b - j + a]$ это палиндром, он будет найден алгоритмом, и его длина не меньше чем y.
Для того чтобы показать, что $S[j \dots b - j + a]$ это палиндром, достаточно показать, что его длина не превосходит $l(S)$, так как её центр совпадает с центром $S[a \dots b]$.
$b - j + a - j + 1 = l(S) - 2(j - a) < l(S) - 2(\frac{l(S)-y}{2} - 2^{k_\varepsilon d}) = y + 2^{k_\varepsilon d+1} < y + 2^{k_\varepsilon+1} y \leqslant$
$\leqslant (1 + 2^{k_\varepsilon+1})(\lceil \frac{l(S)}{2^{k_\varepsilon+2}} \rceil + 1) \leqslant (1 + 2^{k_\varepsilon+1})(\frac{l(S)}{2^{k_\varepsilon+2}} + 2) \leqslant [l(S) \geqslant 2^{k_\varepsilon+3} > \frac{2^{k_\varepsilon+3}(1+2^{k_\varepsilon+1})}{2^{k_\varepsilon+2}-2^{k_\varepsilon}-1}] \leqslant l(S)$.
Покажем теперь, что длина $S[j \dots b - j + a]$ больше либо равна y:
$b - j + a - j + 1 = l(S) - 2(j - a) \geqslant l(S) - (l(S) - y) = y$.
Осталось показать, что $j + tll(j) > b - j + a$. Это неравенство будет означать, что на итерации $b - j + a$ будет обнаружен искомый палиндром.
$b - j + a - j < l(S) - 1 - 2(j - a) < ((l(S) - y) - 2^{1+k_\varepsilon d}) = y + 2^{1+k_\varepsilon d} \leqslant 2^{k_\varepsilon d} + 2^{1+k_\varepsilon d} < 2^{2+k_\varepsilon d} \leqslant ttl(j)$.
■

Замечание 7.1 *Было доказано что алгоритм найдёт палиндром длины не меньше чем* $\lceil \frac{l(S)}{2^{k_\varepsilon+2}} \rceil$.

Теперь покажем как применить модификации, аналогичные описанным в разделе 6, для получения аналога алгоритма 6.1.

Утверждение 7.7 *Пусть перед итерацией i длина самого большого найденного палиндрома равна a, а после итерации i длина самого большого найденного палиндрома равна b. Тогда* $b \leqslant 2^{2+k_\varepsilon} a + 2$.

Доказательство. $b \leqslant l(S[1 \dots i]) \leqslant l(S[1 \dots i - 1]) + 2 \leqslant 2^{2+k_\varepsilon} a + 2$. ■

Утверждение 7.8 *На каждой итерации достаточно проверять* 15 *подстрок на палиндромность.*

Доказательство. Рассмотрим итерацию i. Пусть длина самого большого найденного палиндрома найденного до этой итерации равна a.
Если $a = 0$, то на i-ой итерации может быть найден только палиндром длины 1 или 2, то есть достаточно проверить 2 подстроки.

Рассмотрим теперь случай $a > 0$.
Оценим количество в SP элементов $I(j)$ таких, что $j \in [i - 2^{2+k_\varepsilon}a - 1, i - a]$. По утверждению 7.7 достаточно обновлять ответ только с помощью этих элементов. Воспользуемся формулой 2:

$$\bigcup_{b=0}^{\lfloor \log_{2^{k_\varepsilon}}(i) \rfloor} \{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i] \cap [i - 2^{2+k_\varepsilon}a - 1, i - a], j + ttl(j) > i, ttl(j) \geqslant 2^{2+k_\varepsilon b}\}$$

Определим $b_m = \min\{b | 2^{3+k_\varepsilon b} > a\}$. Заметим, что множества соответствующие $b < b_m$ пусты, так как j выбирается из пересечения непересекающихся областей.
Определим $b_M = \min\{b | 2^{3+k_\varepsilon b} > 2^{2+k_\varepsilon}a + 1\}$. Все множества соответсвующие $b > b_M$ являются подмножествами множества соответствующего b_M, так как области из которых выбирается j совпадают, но при больших b более жёсткое ограничение на делимость.
Таким образом все подходящие для обновления элементы содержатся в:

$$\bigcup_{b=b_m}^{b_M} \{I(j) | j > 0, j \in (i - 2^{3+k_\varepsilon b}, i], j + ttl(j) > i, ttl(j) \geqslant 2^{2+k_\varepsilon b}\}$$

Мощность каждого множества в объединении по утверждению 7.5 не превосходит 5. Оценим количество множеств в объединении:
$b_M - b_m + 1 < 1 + \frac{\log(2^{2+k_\varepsilon}a+1)-3}{k_\varepsilon} - \frac{\log a - 3}{k_\varepsilon} + 1 = 2 + \frac{1+\log(2^{2+k_\varepsilon}a) - \log a}{k_\varepsilon} = 2 + \frac{3+k_\varepsilon}{k_\varepsilon} < 4$.
Таким образом мы показали, что достаточно проверить $3 \cdot 5 = 15$ подстрок на палиндромность. ■

Определим структуру $ES(i)$ как связный список максимальных по включению отрезков разрядов с одинаковым значением в записи числа i в системе счисления с основанием 2^{k_ε}. Эта структура будет выполнять роль $BS(i)$ из алгоритма 6.1.

Утверждение 7.9 *$ES(x)$ занимает $O(\log_{2^{k_\varepsilon}} x) = O(\log_{1+\varepsilon} x)$ памяти.*

Утверждение 7.10 *$ES(x)$ можно преобразовать в $ES(x+1)$ за $O(1)$.*

Доказательство. Путь первый отрезок в $ES(x)$ это $[0, a]$ и значение в этих разрядах равно b.
Рассмотрим сначала случай $b \neq 2^{k_\varepsilon} - 1$.
Если $a > 0$ разобьём первый отрезок на два: $[0, 0]$ со значением $b + 1$ и $[1, a]$ со значением b.
Если $a = 0$ изменим значение принимаемое на этом отрезке на $b+1$, и если значение на принимаемое следующем отрезке также $b + 1$, объединим первые два отрезка.
В случае $b = 2^{k_\varepsilon} - 1$ присвоем 0 значению на первом отрезке. Так как исходное значение на следующем отрезке отлично от $2^{k_\varepsilon} - 1$, применим к нему действия описанные в пером случае. ■

Утверждение 7.11 *Из $ES(i)$ можно получить $\gamma_\varepsilon(i)$ и $\chi_\varepsilon(i)$ за $O(1)$.*

Для быстрого удаления из SP нужного элемента, заведём для каждого возможного значения $ttl(i)$ очередь ссылок на элементы SP с таким значением ttl. Значение $ttl(i)$ определяется по $\gamma_\varepsilon(i)$ и булевому значению, которое показывает выполняется ли равенство $\chi_\varepsilon(i) = 2^{k_\varepsilon} - 4$. Проиндексируем описанные очереди этими двумя параметрами. Таким образом, на итерации i происходит добавление в очередь $QU(\gamma_\varepsilon(i), \chi_\varepsilon(i) = 2^{k_\varepsilon} - 4)$ и удаление из очереди $QU(\gamma_\varepsilon(i), \chi_\varepsilon(i) = 4)$.
Так как $\gamma_\varepsilon(i) \leqslant \log_{2^{k_\varepsilon}}(i)$, то количество очередей есть $O(\log_{2^{k_\varepsilon}} n) = O(\log_{1+\varepsilon} n)$.

Алгоритм 7.1 *Алгоритм приближённый логарифмический для больших ε, i-ая итерация:*

```
1  Добавить I(i) в начало SP
2  Если i = 1
3          Присвоить sp указатель на начало SP
4  Построить ES(i), используя ES(i - 1).
5  Посчитать gamma(i) и chi(i), используя ES(i).
6  Если QU(gamma(i), chi(i) = 4) не пуста
7          Удалить из SP узел на который ссылается первый элемент QU(gamma(i), chi(i) = 4)
8                  Если удаляется элемент, на который ссылается sp, перед удалением:
9                          sp = следующий(sp)
```

```
10          Удалить из очереди QU(gamma(i), chi(i) = 4) первый элемент.
11  Добавить указатель на начало SP к очереди QU(gamma(i), chi(i) = 2^k - 4).
12  Считать S[i], преобразовать I(i) в I(i + 1)
13  sp = предыдущий(sp)
14  Пока (i - sp.i + 1 <= answer) и (sp не последний узел в SP)
15          sp = следующий(sp)
16  Для v имеющих смысл из {sp, следующий(sp), ... , следующий^14(sp) }
17          Если S[v.i...i] является палиндромом
18                  Обновить ответ палиндромом S[v.i...i]
```

Теорема 7.2 *Алгоритм корректен, использует* $O(\log_{1+\varepsilon} n)$ *памяти, всего* $O(n)$ *времени, при этом каждая итерация выполняется за* $O(1)$.

Доказательство. Утверждение 7.8 показывает, что рассматриваемый алгоритм найдёт такой же ответ как и адаптированный под рассматриваемые ограничения алгоритм 5.1, корректность которого показана в теореме 7.1. Размер $ES(x)$ есть $O(\log_{1+\varepsilon} n)$. Всего очередей QU используется $O(\log_{1+\varepsilon} n)$, при этом их суммарный размер равен размеру SP. Следовательно, согласно утверждению 7.6, всего алгоритм использует $O(\log_{1+\varepsilon} n)$ памяти.
Цикл изменяющий *sp* не может совершить более чем 24 итераций. Пересчёт $ES(i)$ по утверждению 7.10 выполняется за $O(1)$. Вычисление $\gamma_\varepsilon(i)$ и $\chi_\varepsilon(i)$ выполняется за $O(1)$ по утверждению 7.11. А значит в целом каждая итерация алгоритма выполняется за $O(1)$. ■

Список литературы

[1] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In Raffaele Giancarlo and Giovanni Manzini, editors, Combinatorial Pattern Matching, volume 6661 of Lecture Notes in Computer Science, pages 162–172. Springer Berlin Heidelberg, 2011.

[2] Petra Berenbrink, Funda Ergun, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome Recognition In The Streaming Model. In STACS 2014, volume 25 of LIPIcs, pages 149–161, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] Pawel Gawrychowski and Przemyslaw Uznanski. Tight tradeoffs for approximating palindromes in streams. Technical Report 1410.6433, arxiv.org, 2015.

[4] Zvi Galil and Joel Seiferas. A linear-time on-line recognition algorithm for “palstar”. J. ACM, 25(1):102–111, January 1978.

[5] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. SIAM Journal on Computing, 6(2):323–350, 1977.

[6] Glenn K. Manacher. A new linear-time “on-line” algorithm for finding the smallest initial palindrome of a string. J. ACM, 22(3):346–351, 1975.

[7] Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. Theor. Comput. Sci., 141(1&2):163–173, 1995.

[8] R. Karp and M. Rabin. 1987. Efficient randomized pattern matching algorithms. IBM Journal of Research and Development 31, 2, 249–260.

[9] L. Kari and K. Mahalingam. Watson—Crick palindromes in DNA computing. Natural Computing, (9):297–316, 2010.