Вероятностный алгоритм для обновления файлов через линию связи

А.В. Евфимьевский*

Реферат

Пусть **P** и **Q** — два абонента, соединенные двусторонней линией связи; абонент **P** имеет старую версию некоторого текстового файла и хочет получить обновленную версию этого файла от абонента **Q**. Предлагаемый алгоритм осуществляет операцию обновления, передавая $poly(\log|x|, \log|y|, d(x,y), \log(1/\varepsilon))$ битов по линии связи (в обоих направлениях); здесь x и y — соответственно старая и новая версии файла, $d(x,y)$ — количество операций редактирования текста, необходимое для превращения x в y, ε — вероятность ошибки. Время работы алгоритма не превосходит $poly(|x|, |y|, \log(1/\varepsilon))$.

1 Введение

Рассмотрим двух абонентов, **P** и **Q**, соединенных друг с другом посредством двустороннего канала связи. Пусть у абонента **P** есть двоичное слово x, а у абонента **Q** — двоичное слово y, «близкое» к x. Абоненту **P** требуется получить слово y; однако при осуществлении пересылки слова необходимо, чтобы по линии связи передавалось как можно меньшее количество битов. Для уменьшения этого количества разрешается воспользоваться тем, что абонент **P** уже имеет файл x, содержащий некоторую информацию об y. Степень «сходства» $d(x,y)$ между файлами x и y определяется как количество операций текстового редактора, переводящих x в y (см. определение 1). Отметим, что это отношение не является симметричным.

Описанную задачу естественно рассматривать как частный случай более общих вопросов, изучаемых в рамках теории коммуникационной сложности. Хорошо известен алгоритм Рабина и Яо [6], осуществляющий сравнение двух слов, принадлежащих различным абонентам **P** и **Q**, допускающий малую вероятность ошибки и пересылающий по линии связи логарифмическое количество битов. В то же время передача слова y от **Q** к **P** (без каких-либо дополнительных предположений) не может быть осуществлена быстрее, чем посредством пересылки по линии связи всех битов слова y. Мы же рассматриваем промежуточную ситуацию: абонент **P** уже имеет слово x, содержащее некоторую информацию о слове y; как мы покажем далее, в этом случае вероятностная коммуникационная сложность оценивается полиномом от $\log|x|+\log|y|+d(x,y)+\log\varepsilon^{-1}$. Наш алгоритм весьма прост (хотя доказательство оценки несколько сложнее).

На практике слово y может быть новой версией некоторого файла, который необходимо переслать по сети с сервера **Q** пользователю **P**; при этом у пользователя часто уже имеется устаревшая версия. Если сервер знает, какая именно версия файла хранится у пользователя и какие изменения с тех пор были внесены в файл, он может послать лишь эти изменения; однако в противном случае наш алгоритм может оказаться полезным.

*Кафедра математической логики и теории алгоритмов, механико-математический факультет, Московский Государственный университет им. М.В. Ломоносова. Электронный адрес: *alexand@evfimyev.mccme.rssi.ru*

Свойства подобных нашему отношений «сходства» на словах изучались в работах Ханненхалли и Певзнера [2], где построены полиномиальные алгоритмы для вычисления кратчайших последовательностей операций редактирования, переводящих одно заданное слово в другое. Также они исследовались Шульманом и Цукерманом в статье [4] в рамках построения кодов, исправляющих ошибки.

Решаемую в данной статье задачу мне сообщил А.Х. Шень. Я глубоко признателен А.Х. Шеню, Н.К. Верещагину и А. Разборову за оказанную помощь.

Предварительная версия этой статьи (на английском языке) будет опубликована в Proceedings of the Ninth Annual Symposium on Discrete Algorithms [1].

2 Основная теорема

Определение 1. *Расстоянием* $d(x,y)$ от двоичного слова y до двоичного слова x называется наименьшее количество операций текстового редактора, достаточное для превращения слова x в слово y. Допускаются операции следующих типов:

- вставка бита: $AB \to A0B$ или $A1B$;
- удаление бита: $A0B \to AB$ или $A1B \to AB$;
- перестановка блока: $AXBC \to ABXC$;
- удаление блока: $AXB \to AB$;
- удвоение (копирование) блока: $AXBC \to AXBXC$.

Если $k \geqslant d(x,y)$, мы будем говорить, что слово y *k–близко* к слову x.

Теорема 1. *Существуют вероятностные алгоритмы для абонентов* $\mathbf{P}$ *и* $\mathbf{Q}$*, которые для любых слов x и y и для любого $\varepsilon > 0$ осуществляют передачу слова y от абонента* $\mathbf{Q}$ *к абоненту* $\mathbf{P}$ *с вероятностью ошибки ε. При этом число фактически пересылаемых битов ограничено $poly(d(x,y), \log|x|, \log|y|, \log\varepsilon^{-1})$, а время работы не превышает $poly(|x|, |y|, \log\varepsilon^{-1})$.*

Более точно, коммуникационная сложность этой передачи ограничена сверху величиной $O(\, d(x,y)^2 \, \log|y| \cdot (\log|x| + 5\log|y| + \log\varepsilon^{-1})\,)$ (см. (2)).

3 Описание алгоритма

Не ограничивая общности, можно предположить, что длина слова y есть некоторая степень двойки: $|y| = 2^r$, $r \in \mathbf{N}$ (в противном случае абонент $\mathbf{Q}$ может добавить необходимое количество нулей к y и сообщить их число абоненту $\mathbf{P}$). Вначале абонент $\mathbf{Q}$ посылает абоненту $\mathbf{P}$ длину слова y, а $\mathbf{P}$ выделяет память для хранения получаемой в дальнейшем информации. Затем один из абонентов, например $\mathbf{Q}$, выбирает и сообщает другому произвольное простое число $p = \Theta(|x| \cdot |y|^5 \varepsilon^{-1})$; более точно,

$$501 \cdot |x||y|^5\varepsilon^{-1} \leqslant p \leqslant 100 \cdot 501 \cdot |x||y|^5\varepsilon^{-1}. \tag{1}$$

Поиск простого числа нетрудно выполнить за полиномиальное время, воспользовавшись одним из хорошо известных вероятностных алгоритмов проверки простоты (см. [3, 5]).

Основной процедурой нашего алгоритма является «попытка переслать подслово z слова y от $\mathbf{Q}$ к $\mathbf{P}$»; обозначим ее через $\mathtt{try}(z)$. При выполнении $\mathtt{try}(z)$ предполагается, что как $\mathbf{P}$, так и $\mathbf{Q}$ уже знают длину подслова z и его местонахождение в y и, кроме того, что абонент **P** уже получил часть y, лежащую слева от z (хотя существует малая вероятность, что переданная абоненту **P** часть слова y содержит ошибку).

Чтобы выполнить операцию $\mathtt{try}(z)$, абонент **Q** сначала выбирает случайным образом элемент α из поля вычетов $\mathbf{Z}/p\mathbf{Z}$ и вычисляет «контрольную сумму» подслова z, равную значению следующего многочлена над полем $\mathbf{Z}/p\mathbf{Z}$:

$$\beta = z_0 + z_1\alpha + z_2\alpha^2 + \ldots + z_{|z|-1}\alpha^{|z|-1} = \\ = \sum_{j=0}^{|z|-1} z_j\alpha^j, \quad \text{где } z = \overline{z_0 z_1 \ldots z_{|z|-1}}.$$

После этого числа α и β посылаются абоненту **P**. Тот, в свою очередь, перебирает все подслова длины $|z|$ в слове x, а также в уже имеющейся у него левой части слова y, и вычисляет для этих подслов контрольные суммы (по той же формуле, и пользуясь тем же вычетом α). Если ни одна из этих контрольных сумм не совпала с β, то **P** сообщает **Q** о неудаче в операции $\mathtt{try}(z)$. Если же **P** нашел подслово с контрольной суммой β, он копирует это подслово и приписывает его копию справа к ранее полученной части y на место подслова z; в этом случае он сообщает **Q**, что операция прошла успешно (рис. 1).

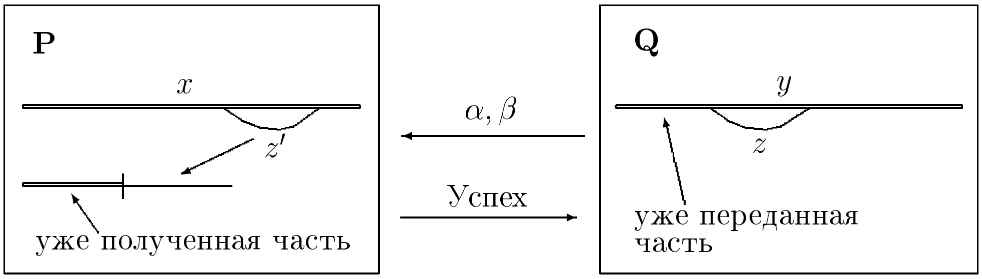


Рис. 1: Выполнение процедуры $\mathtt{try}(z)$.

Если передача подслова z не удалась, мы делим z пополам и повторяем эту операцию с левой половиной z посредством рекурсивного вызова. Более формально, мы используем следующую процедуру $\mathtt{send}(z)$ для пересылки подслова z:

```
procedure send(z);
begin
   if |z| = 1 then передаем z напрямую;
   else begin
      try(z); if неудача then begin
         send(левая половина z);
         send(правая половина z);
      end;
   end;
end.
```

Наш алгоритм сохраняет (с большой вероятностью) следующее инвариантное отношение: передавая некоторое подслово z, мы уже передали всю часть слова y, лежащую слева от z. Операция пересылки всего слова y соответствует выполнению инструкции $\mathtt{send}(y)$.

Таким образом, алгоритм достаточно прост; тем не менее, доказательство верхней границы на количество пересылаемых битов требует несколько лемм технического характера. Однако пока мы отложим доказательства лемм и перейдем к доказательству основной теоремы.

4 Доказательство основной теоремы

Чтобы доказать теорему, надо оценить сверху количество битов, передаваемых по линии связи, и вероятность ошибочной передачи. Для этого нам сперва необходимо оценить количество операций $\mathtt{try}(z)$, исполняемых нашим алгоритмом.

Согласно лемме 2 (см. раздел 5), слово y можно разделить на $M \leqslant 112\, d(x,y)^2$ блоков таким образом, что каждый блок будет совпадать либо с некоторым блоком в x, либо с некоторым блоком в y, лежащим слева от него. Поэтому всякая операция $\mathtt{try}(z)$, аргумент которой содержится в таком блоке, будет обязательно успешной (хотя необязательно правильной). Отсюда следует, что операция $\mathtt{send}(z)$ может производить рекурсивные вызовы только в том случае, если подслово z пересекает границу между какими-нибудь двумя этими блоками.

Какие подслова z могут оказаться аргументами операции $\mathtt{send}(z)$ во время работы алгоритма? Только те, длина которых есть степень двойки и чья начальная координата кратна их длине. Если $|y| = 2^n$, то для каждого $i = 1, \ldots, n$ не более чем M таких подслов длины 2^i могут пересечь границу между блоками. Это значит, что общее количество подслов z, на которых операция $\mathtt{send}(z)$ производит рекурсивные вызовы, не превышает $M \log |y|$. Каждая такая операция $\mathtt{send}(z)$ сперва вызывает $\mathtt{try}(z)$, а затем делает два рекурсивных вызова. Следовательно, общее количество $N_{\mathtt{try}}$ вызовов операции $\mathtt{try}(z)$ можно оценить следующим образом:

$$N_{\mathtt{try}} \leqslant 3M \log |y| \leqslant 336\, d(x,y)^2 \log |y|.$$

(Мы не рассматриваем отдельно случай $|z| = 1$, поскольку пересылка одного бита — еще более простая операция.)

Теперь рассмотрим работу процедуры $\mathtt{try}(z)$. В процессе ее выполнения абонент **Q** передает абоненту **P** числа α и β, длина каждого из которых не превышает $\lceil \log p \rceil$; **P** пересылает лишь один бит. Количество битов, пересылаемых во время работы всего алгоритма, не превосходит таким образом

$$\begin{aligned} N_{bits} &\leqslant \log |y| + \lceil \log p \rceil + N_{\mathtt{try}} \cdot (2\lceil \log p \rceil + 1) \leqslant \\ &\leqslant O(\, d(x,y)^2 \log |y| \cdot \lceil \log p \rceil\,). \end{aligned}$$

Используя верхнюю оценку на число p (из (1)), получаем:

$$\begin{aligned} p \leqslant O(|x||y|^5 \varepsilon^{-1}) \quad &\Rightarrow \\ \Rightarrow \quad N_{bits} \leqslant O(\, d(x,y)^2 \log |y| \cdot \lceil \log p \rceil\,) &\leqslant \\ \leqslant O(\, d(x,y)^2 \log |y| \cdot (\log |x| + 5 \log |y| &+ \\ + \log \varepsilon^{-1})\,) \leqslant poly(\, d(x,y), \log |x|, \log |y|, \log \varepsilon^{-1}\,)&; \end{aligned} \tag{2}$$

полиномиальная оценка на время работы алгоритма очевидна.

Нам осталось оценить вероятность ошибки. Чтобы определить, совпадает ли подслово z с подсловом z', наш алгоритм вычисляет их контрольные суммы, т.е. многочлены вида

$$z_0 + z_1 \alpha + z_2 \alpha^2 + \ldots + z_{|z|-1} \alpha^{|z|-1}$$

в случайной точке $\alpha \in \mathbf{Z}/p\mathbf{Z}$, и сравнивает их значения. Алгоритм может совершить ошибку только тогда, когда $z \neq z'$ и когда вычет α оказался корнем многочлена

$$\begin{aligned} (z_0 - z'_0) + (z_1 - z'_1)\alpha + (z_2 - z'_2)\alpha^2 + \ldots &+ \\ +(z_{|z|-1} - z'_{|z|-1})\alpha^{|z|-1}&. \end{aligned} \tag{3}$$

Этот многочлен не равен тождественно нулю, а его степень меньше $|z|$; поэтому вероятность ошибки при сравнении не превосходит $\frac{|z|}{p}$. Во время выполнения одной операции

$\mathtt{try}(z)$ алгоритм делает не более $|x| + |y|$ сравнений, а количество таких операций не превышает $N_{\mathtt{try}}$, так что общая вероятность ошибки ограничена сверху выражением

$$N_{\mathtt{try}} \cdot (|x| + |y|) \cdot |y| p^{-1} \leqslant 500\, d(x,y)^2 |x||y|^3 p^{-1}.$$

Принимая во внимание, что $d(x,y) \leqslant |y|$ и $p > 500 \cdot |x||y|^5 \varepsilon^{-1}$ (см. (1)), получаем такую верхнюю оценку:

$$\begin{aligned} 500\, d(x,y)^2 |x||y|^3 p^{-1} < \\ < 500 |x||y|^5 \cdot (500|x||y|^5 \varepsilon^{-1})^{-1} = \varepsilon. \end{aligned}$$

□

5 Вспомогательные леммы

Определение 2. Назовем *операцией копирования–приписывания* (*КП–операцией*) преобразование двоичного слова ABC в слово ABCB (где A, B и C — произвольные слова). В слове ABCB мы будем называть первое вхождение блока B *блоком–источником*, а второе вхождение — *блоком–копией.*

Лемма 1. *Если слово y k–близко к слову x, то существует последовательность из не более чем $4k$ КП–операций, преобразующая слово x в слово xuy (для некоторого промежуточного слова u). Каждый бит в слове xuy может входить в блоки–источники не более чем двух КП–операций в этой последовательности.*

Доказательство. Все блоковые операции из определения k–близости могут быть переведены на язык операций копирования–приписывания (используя не более 4-х КП–операций вместо одной блоковой операции) следующим образом:

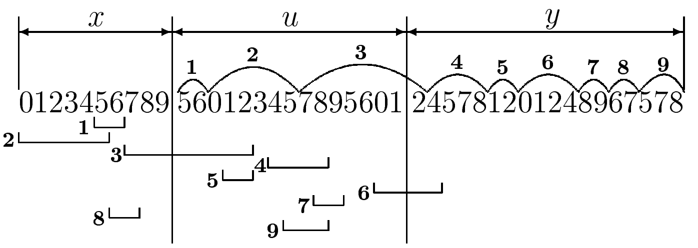
Удаление:	Перестановка:	Копирование:
ABC ABC A ABC AC	ABCD ABCD A ABCD AC ABCD ACB ABCD ACBD	ABCD ABCD ABC ABCD ABCB ABCD ABCBD

Операции с битами можно рассматривать как операции с блоками (при условии, что все слова содержат и нули, и единицы).

Получившееся слово xuy оказывается составленным из подслов, где каждое последующее подслово 1–близко к предыдущему. Поскольку каждый бит в xuy принадлежит одному из этих подслов, он может оказаться в блоке–источнике только для тех КП–операций, которые участвуют в порождении следующего подслова, т.е. только дважды. □

Лемма 2. *Если слово y k–близко к слову x, то существует последовательность из не более $112 \cdot k^2$ КП–операций, преобразующая x в слово xy.*

Доказательство. По лемме 1 найдется такое промежуточное слово u, что x можно преобразовать в слово $w = xuy$ посредством $m \leqslant 4k$ КП–операций. Для каждой КП–операции определены ее блок–источник $[p,q]$ и блок-приемник $[p',q']$ (где p есть координата первого бита в блоке $[p,q]$, а q — координата последнего бита); при этом $p \leqslant q < p' \leqslant q'$, $q - p = q' - p'$, и подслова $\overline{w_p w_{p+1} \ldots w_q}$ и $\overline{w_{p'} w_{p'+1} \ldots w_{q'}}$ совпадают. Поскольку участок uy образован КП–операциями, он состоит из m блоков–копий, расположенных друг за другом (рис. 2).



КП–операции:
(в случае, если цифры соответствуют битам)
1. [5,6] $\to$ [10,11];
2. [0,5] $\to$ [12,17];
3. [7,14] $\to$ [18,25];
4. [16,19] $\to$ [26,29];
5. [13,14] $\to$ [30,31];
6. [23,26] $\to$ [32,35];
7. [19,20] $\to$ [36,37];
8. [6,7] $\to$ [38,39];
9. [17,19] $\to$ [40,42].

Рис. 2: Пример слова xuy, полученного из x с помощью нескольких КП–операций. Здесь цифры обозначают произвольные двоичные подслова (одинаковые цифры соответствуют одинаковым подсловам); блоки–источники показаны под словом, блоки–копии — над словом.

Наша задача состоит в устранении слова u путем такого изменения последовательности КП–операций, что новая последовательность будет преобразовывать x уже не в xuy, а в xy. Это изменение мы будем производить пошагово, устраняя на каждом шаге правый блок–копию в слове u; остальные блоки–копии, образующие u (лежащие слева от удаляемого блока), не затрагиваются на этом шаге, однако некоторые КП–операции, образующие y, изменяются.

Перед началом преобразования посмотрим, не проходит ли граница между словами u и y по какому-нибудь блоку–копии. Если проходит, заменим соответствующую КП–операцию на две КП–операции так, чтобы граница между u и y стала границей между двумя новыми блоками–копиями.

ШАГ преобразования:

Перед выполнением шага преобразования подлежащий удалению блок–копия $[p', q']$ в слове u пересекается с некоторыми блоками–источниками $[p_j, q_j]$; соответствующие им блоки–копии расположены в слове y (рис. 3).

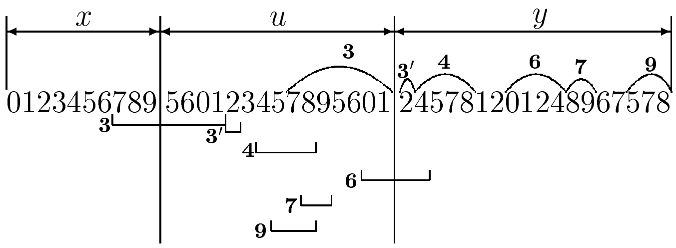


Рис. 3: Слово из рис. 2 перед шагом преобразования. КП–операция 3 разбита на две; те блоки, которые не изменятся во время выполнения этого шага, не показаны.

Подшаг 1: Каждый блок–источник, пересекающийся с правым блоком–копией $[p', q']$, разбивается на части; точками деления будут левая и правая границы блока $[p', q']$, а также все те границы всех блоков–источников, которые находятся внутри $[p', q']$. Разбивая блоки–источники, мы также разбиваем и соответствующие им блоки–копии (расположенные в слове y), после чего заменяем каждую из соответствующих КП–операций на несколько последовательных КП–операций (рис. 4).

Подшаг 2: Теперь любые два блока–источника внутри $[p', q']$ либо не пересекаются, либо совпадают. В каждой группе совпадающих блоков–источников выберем тот

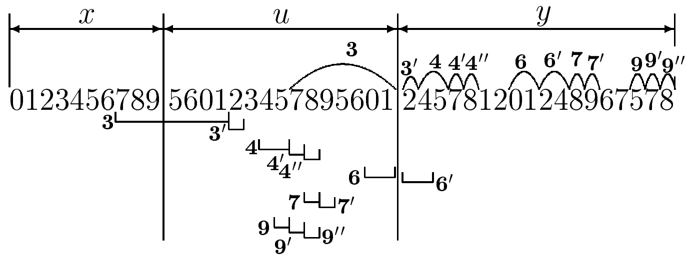


Рис. 4: Результат выполнения подшага 1.

из них, которому соответствует самый левый блок–копия в y. Затем изменим КП–операции, соответствующие остальным блокам–источникам из группы, используя упомянутый выше блок–копию в качестве их нового блока–источника (рис. 5).

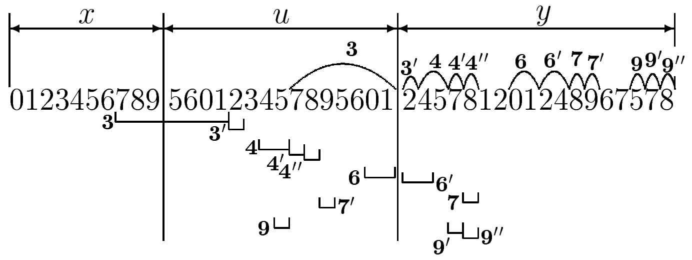


Рис. 5: Результат выполнения подшага 2.

Подшаг 3: Все блоки–источники, оставшиеся внутри $[p', q']$, попарно не пересекаются. Подлежащий удалению блок–копия $[p', q']$ соответствует блоку–источнику $[p, q]$, расположенному слева от $[p', q']$ (где-нибудь в u или в x). Поскольку блоки $[p, q]$ и $[p', q']$ содержат одинаковые подслова, мы можем переместить в $[p, q]$ все блоки–источники из $[p', q']$. После этого внутри $[p', q']$ не остается ни одного блока–источника; этот участок слова u становится бесполезным. Мы удаляем это подслово, а также и производящую его КП–операцию «$[p, q] \to [p', q']$» из последовательности КП-операций (рис. 6).

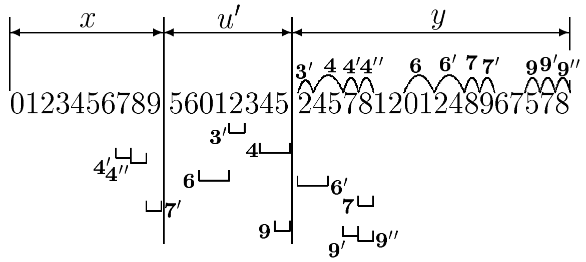


Рис. 6: Результат выполнения всего шага.

После одного шага преобразования количество блоков–копий внутри u уменьшается на единицу, поэтому для удаления всего слова u требуется не более m шагов.

Нам остается оценить количество блоков после выполнения преобразования. Обозначим переменной s количество точек в слове u (границ между соседними битами), которые содержат по крайней мере одну границу (левую или правую) какого-нибудь блока–источника (если на одну точку приходится несколько границ, мы ее считаем как одну). Переменной t обозначим максимальное количество блоков–источников, содержащих один и тот же бит из слова u. Нетрудно увидеть, что t не увеличивается в процессе преобразования. Действительно, разбиение блоков не меняет t; выведение некоторых блоков–источников за пределы слова u может лишь уменьшить t; при перемещении нескольких непересекающихся блоков–источников с блока $[p', q']$ на блок $[p, q]$ t может увеличиться на 1, но это возрастание будет тут же скомпенсировано удалением КП–операции «$[p, q] \to [p', q']$».

Значение s может увеличиваться, но не более чем на 1 за каждый шаг преобразования (на самом первом шаге s может увеличиться на 2). На подшаге 1 к множеству «граничных точек» могут добавиться лишь две новые точки — левая и правая границы блока–копии $[p', q']$; все остальные новые границы проводятся в точках, содержащих границы ранее. Более того, если этот шаг — не первый, то правая граница блока $[p', q']$ приходится на ту же точку, что и левая граница блока–копии, удаленного на предыдущем шаге; следовательно, эта точка уже посчитана. На подшаге 2 переменная s не меняется; не возрастает она и на подшаге 3, так как, во-первых, каждая новая граничная точка внутри $[p, q]$ соответствует некоторой удаляемой граничной точке внутри $[p', q']$, а во-вторых, обе граничные точки, перемещенные с левой и правой границ $[p', q']$, оказываются в точках, уже ранее содержавших границы.

Перед началом выполнения преобразования значение t не превышало 2 (см. лемму 1), а значение s не превышало $2m$. Поскольку всего производится не более m шагов, после каждого шага мы будем иметь

$$s \leqslant 3m + 1; \qquad t \leqslant 2.$$

На каждом шаге после подшага 1 внутри блока–копии $[p', q']$ будет не более $s \cdot t$ блоков–источников. Так как только эти блоки могут добавиться к множеству всех блоков–источников на этом шаге, общее количество блоков возрастет не более чем на $6m + 2$.

После окончания процесса преобразования количество КП–операций M можно оценить так:

$$M \leqslant m + m \cdot st \leqslant 3m + 6m^2 \leqslant 7m^2 \leqslant 112k^2$$

(при $m > 3$). □

Список литературы

[1] A.V. Evfimievski, *A probabilistic algorithm for updating files over a communication link.* Proc. ACM–SIAM Symposium on Discrete Algorithms, 1998 (to appear)

[2] Sridhar Hannenhalli, Pavel Pevzner, *Transforming men into mice (polynomial algorithm for genomic distance problem).* Tech. Report CSE–95–012, Dep. of Computer Science and Engineering, Penn. State Univ., May 1995

[3] M.O. Rabin, *Probabilistic algorithms for testing primality.* J. Number Theory, 12(1980), p. 128–138

[4] Leonard J. Schulman, David Zuckerman, *Asymptotically good codes correcting insertions, deletions, and transpositions.* Proc. ACM–SIAM Symposium on Discrete Algorithms, 1997

[5] R. Solovay, V. Strassen, *A fast Monte–Carlo test for primality.* SIAM J. Comput., 6(1977), p. 84–85; Erratum 7(1978), p. 118

[6] A.C. Yao, *Some complexity questions related to distributed computing.* Proc. 11th Ann. ACM Symp. on Theory of Computing (1979), p. 209–213